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4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 4 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.

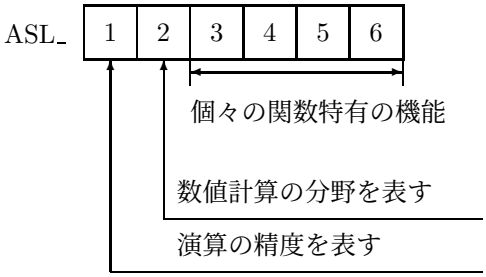


図 1−1　関数名の構成要素

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　微分方程式とその応用

2.1　概 要

本章は, 常微分方程式初期値問題, 境界値問題, 積分方程式と偏微分方程式からなる. 初期値問題は, 常微分方程式から離散点での近似解を順次求め, 常微分方程式 (境界値問題) と偏微分方程式は, 境界内の任意点での近似解を求め, 積分方程式は, 任意の点での近似解を求める.

本ライブラリの 2.2.1 ～ 2.2.4 および 2.3.1 ～ 2.3.2 の関数は, 高階連立常微分方程式に対応しているため, 連立 1 階常微分方程式に置き換える必要がなく, そのままの式で利用できる. また, 連立 1 階のときや単独 1 階のときでも, 階数や連立の数に対応する引数を 1 にすれば, 解くことができる.

初期値問題で自動刻み幅制御を行う場合で, 解の軌道を求めたいときには求める点での解を出力する以外に刻み幅ごとの解を出力することもできる.

境界値問題では, 境界条件を関数と導関数の値で求める (数値境界) の関数と, 境界での関数と導関数の関数で定める (関数境界) の関数の両方を利用できる.

このライブラリでは, 次の性質に対応した関数を用意している.

常微分方程式初期値問題

(1) **連立高階常微分方程式 (速度優先), 連立 1 階常微分方程式, 高階常微分方程式**
自動刻み幅制御のルンゲ・クッタ・バーナー法を利用して解を求める. 関数評価のコストが少なく要求精度が厳しくない場合の非 ～ 弱スティフ問題に対し最も効率がよい.

(2) **連立高階常微分方程式 (精度優先)**
自動刻み幅, 自動次数制御の線形多段階法を利用して解を求める. 関数評価コストが大きい場合や要求精度が厳しい場合の非 ～ 弱スティフ問題に対し最も効率が良い.

(3) **陰的連立常微分方程式**
陰的常微分方程式とは, $f_i\ (x,\ y_1,\ y_2,\ \cdots,\ y_n,\ y_1',\ y_2',\ \cdots,\ y_n',\ \cdots) = 0\ (\ i = 1,\ 2,\ \cdots,\ n)$ の形をしており $\partial f_i\ /\ \partial y_i\ (\ i = 1,\ 2,\ \cdots,\ n \quad j = 1,\ 2,\ \cdots,\ n)$ が非特異な方程式である. たとえば, 次式のように y_1' (または, y_2') が①式と②式の両方に依存し, 単独の式評価からは決定できない場合がこれにあたる.

$$\begin{cases} \mathrm{y_1'y_2'} - \mathrm{x^2} = 0 & \text{初期値} \quad \mathrm{y_1(x_0) = y_{10}} \cdots\cdots\cdots\ ① \\ \mathrm{y_1'} + \mathrm{y_2'} - \mathrm{y_1} - 2\mathrm{x} = 0 & \text{初期値} \quad \mathrm{y_2(x_0) = y_{20}} \cdots\cdots\cdots\ ② \end{cases}$$

さらに, この関数は代数方程式との連立した常微分方程式も解ける.
代数方程式との連立の場合とは, たとえば次に示すように 1 組が代数方程式または非線形方程式でもう 1 組が常微分方程式といったような場合である.

$$\begin{cases} y_1'y_2' - x^2 = 0 & \text{初期値} \quad y_1(x_0) = y_{10} \cdots\cdots\cdots\ ① \\ y_2' + y_3 + 2x = 0 & \text{初期値} \quad y_2(x_0) = y_{20} \cdots\cdots\cdots\ ② \\ 2y_1 + 3y_2 + y_3 - 1 = 0 \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\ ③ \end{cases}$$

このような問題はこの関数でしか解くことができない.
この関数はこの他に一般の連立高階常微分方程式も解くことができるが, 非線形連立方程式を解きながら積分を進めるので他の関数に比べ計算効率が悪い. また, この関数で非線形連立方程式や非線形方程式を解くことができる. この場合, 偏微係数や微係数の入力は不要である.

(4) **スティフ問題の連立高階常微分方程式**
スティフな問題とは解が独立変数の変化に対して異なったスケールで変化する 2 つまたはそれ以上の因子

から構成されているような問題である. たとえば微分方程式 $y'' = \lambda y(\lambda >> 0)$ を初期条件 $(y = 1, y'' = -\lambda \ \ at \ \ x = 0)$ で解く場合がこれにあたる. この微分方程式の一般解は独立変数 x の変化に対してスケールの異なる因子 $e^{\lambda x}$ と $e^{-\lambda x}$ の一次結合で与えられる. この問題の真解は $y = e^{-\lambda x}$ であるが, $x = x_1$ での解が

$y = e^{-\lambda x_1} + \varepsilon = e^{-\lambda x_1} + \varepsilon' e^{\lambda x_1}$ (ε, ε' : 数値計算上の誤差)

と求まった場合, x が大きくなるに従って

$y \to \varepsilon' e^{\lambda x}$

となり真解から離れてしまうという特徴をもつ.

詳細については, 参考文献 (3) を参照されたい.

本関数を用いて, このような強スティフ問題を効率良く解くことができる.

(5) $M\boldsymbol{y''} + C\boldsymbol{y'} + K\boldsymbol{y} = \boldsymbol{p}(x)$ **型常微分方程式**

運動方程式として知られる常微分方程式 $M\boldsymbol{y''} + C\boldsymbol{y'} + K\boldsymbol{y} = \boldsymbol{p}(x)$ を解くための関数である. ここで, M, C, K, は $n \times n$ 行列でそれぞれ質量行列, 減衰行列, 剛性行列と呼ばれ, $p(x)$ は外力ベクトルである. ただし, n は連立の数を表す.

この関数では M 行列の対角項に 0 を含む (質量 0 となる点がある) ような特異な場合でも解けるように処理を行っている. また入力された刻み幅ごとに解が得られる.

常微分方程式境界値問題

(1) **連立高階, 連立 1 階, 高階常微分方程式**

ルンゲ・クッタ・バーナー法をもとに, 多点射撃法で適切な初期値を見つけて解くもので, 一般の多点射撃法よりも効率よく, しかも確実に解が得られるように射撃点の自動設定や非線形計算部分のパラメータ化を行っている.

(2) **線形高階常微分方程式**

重み付き残差法の中の選点法と $B-$ スプライン関数を組み合わせて解く方法で, 線形であれば高速・高精度で解が得られる.

微分階数は任意で, 大きな階数の線形常微分方程式を解くのに有効である.

(3) **線形 2 階常微分方程式**

係数決定法を 2 階常微分方程式に対し一般化したものである. 線形 2 階常微分方程式のときに有効である.

積分方程式

(1) **第 2 種フレドホルム型積分方程式**

ガウスの積分法を用いて積分方程式を解き, 3 次スプライン関数を用いた補間により任意の点における解を求める.

(2) **第 1 種ボルテラ型積分方程式**

マクローリンの公式を用いて積分方程式を解き, 3 次スプライン関数を用いた補間により任意の点における解を求める.

偏微分方程式

(1) **非同次 Helmholtz 方程式 (2 次元)**

与えられた矩形領域内において 2 次元 5 点差分近似を用いて非同次の Helmholtz 方程式を解く.

(2) **非同次 Helmholtz 方程式 (3 次元)**

与えられた矩形領域内において 3 次元 7 点差分近似を用いて非同次の Helmholtz 方程式を解く.

2.1.1 使用上の注意

2.1.1.1 常微分方程式初期値問題

(1) この関数はいずれも不連続点を含む場合は, 正しい解を得ることができないので, 不連続点があるときは, そこで分割して別々に計算しなければならない.

(2) 強振動をする場合は, ステップ幅を非常に小さくしなければならないが, 解くべき区間が広くなると累積誤差が大きくなり精度が低下する.

(3) 方程式がスティフであるかどうかは不明なことが多い. この場合は, まず, 2.2.2 $\left\{\begin{matrix}\text{ASL\_dksnca}\\ \text{ASL\_rksnca}\end{matrix}\right\}$ で解いてみる. これで ierr=4000 が出力されれば, スティフであると考えられるので, 2.2.4 $\left\{\begin{matrix}\text{ASL\_dkssca}\\ \text{ASL\_rkssca}\end{matrix}\right\}$ で解きなおすとよい.

(4) 2.2.1 $\left\{\begin{matrix}\text{ASL\_dksncs}\\ \text{ASL\_rksncs}\end{matrix}\right\}$, 2.2.2 $\left\{\begin{matrix}\text{ASL\_dksnca}\\ \text{ASL\_rksnca}\end{matrix}\right\}$, 2.2.4 $\left\{\begin{matrix}\text{ASL\_dkssca}\\ \text{ASL\_rkssca}\end{matrix}\right\}$ は連立高階常微分方程式用であるが, 従来の連立 1 階常微分方程式で式が与えられたときは, 次のようにする.

従来	関数 f　(x, y, yp) $\begin{cases} yp[0] = f_1\ (x, y[0], \cdots, y[n-1]) \\ \quad\vdots \\ yp[n-1] = f_n\ (x, y[0], \cdots, y[n-1]) \end{cases}$
	引　数 区間 $x \sim x_f$: x, xf 初期値　y_i^0 : $y[0], y[1], \cdots, y[n-1]$ 連立数　n : n

本ルーチン	関数 f　(x, y, n) $\begin{cases} y(n) = f_1\ (x, y[0], \cdots, y[n-1]) \\ \quad\vdots \\ y[2*n-1] = f_n\ (x, y[0], \cdots, y[n-1]) \end{cases}$
	引　数 区間 $x \sim x_f$: x, xf 初期値　y_i^0 : $y[0], y[1], \cdots, y[n-1]$ 連立数　n : n
	追加引数 最大微分階数以上の値 : mx = 1 各方程式の微分階数 : $m[i-1] = 1 (i = 1, \cdots, n)$

すなわち, y が配列 y[n*(mx+1)] になり, 関数や初期値を, y_i' については y[i − 1 + n] (i = 1, ⋯, n), y_i については y[i − 1] (i = 1, ⋯, n) と与えなければならない. これに伴い, y_i' を計算する関数の引数も異なるので注意を要する. さらに追加引数として mx, m[i − 1] が必要で, それぞれに 1 をセットしておかねばならない.
なお, 高階常微分方程式を連立型に変換すると精度が低下する傾向があり, なるべくもとの高階のままで式を作る方がよい.

(5) 2.2.7 $\left\{\begin{matrix}\text{ASL_dkmncn}\\ \text{ASL_rkmncn}\end{matrix}\right\}$を除くすべての関数は, xf 点または自動きざみ幅制御のきざみ幅ごとの $y_i \sim y_i^{(m)}$ が出力されるようになっているので, いろいろな点での $y_i \sim y_i^{(m)}$ が必要なときは, xf を変更しながら連続してこの関数を利用するか, 自動きざみ幅ごとの出力を指定し, xf を最終点に固定し連続してこの関数を利用しなければならない. このとき出力引数は次の入力引数になるように設定しており, 引数値の変更は xf のみ, または変更なしでよい.

(6) 2.2.7 $\left\{\begin{matrix}\text{ASL_dkmncn}\\ \text{ASL_rkmncn}\end{matrix}\right\}$の関数は関数を係数行列の形で与え, 出力はきざみ幅ごとになっているので, 他とは使い方が異なる.

解の必要な点 x_f での解は, 初期値を与えた点を x として, $(x_f - x)$ を整数 k で分割し, これをきざみ幅 Δx として, x 点での $y_i^{(j)}$ ($i = 1, \cdots$, 連立数, $j = 0, 1$) を初期値に $y_i^{(j)}$ (j =0, 1, 2) を求め, またこれを初期値に Δx 進んだ点の $y_i^{(j)}$ を求めるといった作業を k 回反復しなければならない. このとき k の値が大きいほど精度が上がることになる.

(7) 関数 f の作り方は, 次に示すとおりである.

・関数 f

void FORTRAN f(double *x, double *y, int *n ···) ••• 主プログラムにおける第 1 引数 f と同じ名前とする

```
        ≀
{
        ≀
}
```

・主プログラム

```
        ≀
ierr=ASL_dk ··· (f, x, y, ···);
        ≀
```

2.1.1.2 常微分方程式境界値問題

(1) 不連続点が区間内にあると正確な解が得られないので, 不連続点で分割して解くこと.

(2) 非線形用の関数は, 非線形部分に α を乗じてパラメータ化するようになっている.
この非線形部分とは, ${y_i}^{(j)}$ (i : 配列番号, j : 微分階数) の 2 つ以上の乗除算 (例えば $y_1 \cdot y_2$, y_3'/y_4'', $y_1 \cdot y_1'$, $(y_1')^2$ など) や和, スカラ倍以外の関数 (例えば $\sin(y_1')$, $\|y_2\|$ など) になるところである.
例えば

$$\begin{cases} y_1'' = y_2 \\ y_2' = y_1' - y_1 \cdot y_2 \end{cases}$$

の場合は次のようにしてパラメータ化する.

$$\begin{cases} y_1'' = y_2 \\ y_2' = y_1' - \underline{\alpha} \cdot y_1 \cdot y_2 \end{cases}$$

なお, 以上の非線形部分がない場合は, α を乗じてパラメータ化する必要はない.
非線形部分があるにもかかわらずパラメータ化しなかった場合は, 一般の多点射撃法の算法と同じになり, 演算量の増加を招き, 場合により解けなくなることがある.
線形部分を間違ってパラメータ化したときは, 演算量の増加を招くが, 解への影響はない.

(3) 境界条件を数値で与える場合, in で始点, 終点の判別, ib で要素番号, ic で微分階数, bn でその境界での値を指定する．このとき, in, ib, ic, bn の要素番号は, それぞれの境界条件に対応しておく必要がある．なお, 境界条件の入力順は任意でよい.

(4) 境界条件を関数で与える場合, $y_i^{(j)}$ の始点側での値を $ya_i^{(j)}$, 終点側での値を $yb_i^{(j)}$ とすると, $g_k(ya_i^{(j)}, yb_i^{(j)}) = 0$ の関数で境界条件を定める．このとき, 境界条件の入力順は任意である.

(5) 微分階数, 連立数については, 関数内でエラーチェックができないため, 特に注意を払って入力すること.

2.1.1.3 積分方程式

(1) この関数はいずれも不連続点を含む場合は, 正しい解を得ることができないので, 不連続点があるときは, そこで分割して別々に計算しなければならない.

(2) 2.4.2 $\left\{\begin{matrix}\text{ASL_doiev1}\\ \text{ASL_roiev1}\end{matrix}\right\}$では, 積分区間の分割数が大きすぎると累積誤差が大きくなり精度が低下する.

2.1.2 使用しているアルゴリズム

2.1.2.1 常微分方程式初期値問題

(1) **ルンゲ・クッタ・バーナー法**

この方法は, 打切り誤差推定が可能であるので, きざみ幅の自動制御を要求局所絶対精度と要求局所相対精度のうち緩い方にしたがい目的の解を得る.

きざみ幅 h に対するバーナー法の 1 回反復は $y' = f(x, y)$ として次式で表される.

$$
\begin{aligned}
y_{n+1} &= y_n + \sum_{i=1}^{8} \gamma_i k_i \\
E &= \sum_{i=1}^{8} \gamma_i^* k_i \\
k_i &= hf\left(x_n + \alpha_i h, y_n + h \sum_{j=1}^{i-1} \beta_{ij} k_j \right) \quad (i = 1, 2, \cdots, 8)
\end{aligned}
$$

上式で y_{n+1} は 6 次の打切り精度をもつ近似解で, E はこれと 5 次の打切り精度をもつ近似解との差である.

表 2–1 係 数 表

i	α_i	β_{i1}	β_{i2}	β_{i3}	β_{i4}	β_{i5}	β_{i6}	β_{i7}	γ_i	γ_i^*
1	0								$\frac{57}{640}$	$\frac{33}{640}$
2	$\frac{1}{18}$	$\frac{1}{18}$							0	0
3	$\frac{1}{6}$	$-\frac{1}{12}$	$\frac{1}{4}$						$-\frac{16}{65}$	$-\frac{132}{325}$
4	$\frac{2}{9}$	$-\frac{2}{81}$	$\frac{4}{27}$	$\frac{8}{81}$					$\frac{1377}{2240}$	$\frac{891}{2240}$
5	$\frac{2}{3}$	$\frac{40}{33}$	$-\frac{4}{11}$	$-\frac{56}{11}$	$\frac{54}{11}$				$\frac{121}{320}$	$-\frac{33}{320}$
6	1	$-\frac{369}{73}$	$\frac{72}{73}$	$\frac{5380}{219}$	$-\frac{12285}{584}$	$\frac{2695}{1752}$			0	$-\frac{73}{700}$
7	$\frac{8}{9}$	$-\frac{8716}{891}$	$\frac{656}{297}$	$\frac{39520}{891}$	$-\frac{416}{11}$	$\frac{52}{27}$	0		$\frac{891}{8320}$	$\frac{891}{8320}$
8	1	$\frac{3015}{256}$	$-\frac{9}{4}$	$-\frac{4219}{78}$	$\frac{5985}{128}$	$-\frac{539}{384}$	0	$\frac{693}{3328}$	$\frac{2}{35}$	$\frac{2}{35}$

ここでは E を y_{n+1} の打切り誤差と推定する.

きざみ幅 h の自動調節方法は, 以下による.

ε を max(要求絶対精度, 要求相対精度 $\times \left| \frac{y_n + y_{n+1}}{2} \right|$) とする.

各連立する微分方程式で $\frac{\varepsilon}{|E|}$ の最小のものを求め, これを U とする.

(a) $U < 1$ (きざみ幅が大きすぎる) のとき,
$h = h \times \max(0.85 \times \sqrt[6]{U}, 0.1)$ として h を更新し, 再度 y_{n+1} を求める.

(b) $U \geq 1$ (きざみ幅は十分小さい) のとき,

i. $1.4 \leq U < 2.4$ のとき h をそのままにして次のステップ計算に移る.

ii. $U < 1.4$ または $U \geq 2.4$ のとき,
$h = h \times \min(0.9 \cdot \sqrt[6]{U}, 5)$, y_{n+1} を y_n として次のステップ計算に移る.

なお, h の初期値としては, 最初に求める $y' = f(x, y)$ より最終点 x_f の誤差を $y'(x_f - x)$ として, 次式より計算する.

$$\begin{aligned} \varepsilon &= \max(\text{要求絶対精度}, \text{要求相対精度} \times y) \\ h &= \sqrt[6]{\frac{\varepsilon(x_f - x)}{y'(x_f - x)}} \end{aligned}$$

このうち, 各連立する微分方程式で最小の h を採用する.
高階微分方程式:$y^{(d)} = f(x, y, y', \cdots, y^{(d-1)})$ では

$$y_1 = y, y_2 = y', y_3 = y'', \cdots, y_d = y^{(d-1)}$$

として

$$\left\{ \begin{array}{rcl} y_1' &=& y_2 \\ y_2' &=& y_3 \\ & \vdots & \\ y_d' &=& f(x, y_1, \cdots, y_d) \end{array} \right.$$

とした連立 1 階微分方程式を考えれば良い. 本ライブラリの関数では, この処理を自動的に行っている.
この方法は, 関数評価回数が多くなるので関数評価にかかるコストの少ない場合で, 要求精度が厳しくない場合は最も効率が良い (参考文献 (1), (2) 参照).

(2) **きざみ幅, 次数制御の商差法をもとにした線形多段階法**
微分方程式 $y^{(d)} = f(x, y, \cdots, y^{(d-1)})$ を考える.
いま, 簡単のため, $f(x_n) = f(x_n, y(x_n), \cdots, y^{(d-1)}(x_n))$, $y_n = y(x_n)$ と表す.

(a) $d = 1$ のとき予測子 p_{n+1}^0 は

$$p_{n+1}^0 = y_n + \int_{x_n}^{x_{n+1}} f(x)dx$$

(b) $d > 1$ のとき予測子 $p_{n+1}^{(d-k)}$ は $h_{n+1} = x_{n+1} - x_n$ として

$$\begin{aligned} p_{n+1}^{(d-k)} &= \sum_{i=0}^{k-1} \frac{h_{n+1}^i}{i!} y_n^{(d-k+i)} + \int_{x_n}^{x_{n+1}} \cdots \int_{x_n}^{s_2} f(s_1)ds_1 \cdots ds_k \\ & \qquad (k = 1, \cdots, d) \end{aligned} \tag{2.1}$$

ここで $\int ds_i$ は $y^{(d-i)}$ の x に対する積分を表す.

この $f(x)$ をすでに計算されている微係数を使って多項式近似する.
q 個の点 $(x_i, f(x_i))$ $(i = n, \cdots, n - q + 1)$ を通る $q - 1$ 次近似式を $P_{q-1}(x)$ とすると, これは差分商を使って次のように定義される.

$$\begin{aligned} f[x_n] &= f(x_n) \\ f[x_n, \cdots, x_{n-i}] &= \frac{f[x_n, \cdots, x_{n-i+1}] - f[x_{n-1}, \cdots, x_{n-i}]}{x_n - x_{n-i}} \end{aligned} \tag{2.2}$$

として

$$\begin{aligned} P_{q-1}(x) &= f[x_n] + \cdots + (x - x_n) \cdots (x - x_{n-i+1}) f[x_n, \cdots, x_{n-i}] \\ & \qquad (i = 0, \cdots, q - 1) \end{aligned} \tag{2.3}$$

また, 修正子は点 $(x_{n+1}, f(x_{n+1}))$ を通る q 次式 $P_q^*(x)$ を,

$$P_q^*(x) = P_{q-1}(x) + (x - x_n)\cdots(x - x_{n-q+1})f[x_n, \cdots, x_{n-q+1}] \tag{2.4}$$

として (2.2) 式と同様の積分をすれば得られる.
(2.2) 式の計算のため, 次の記号を定める.

$$\begin{aligned}
\tau &= \frac{x - x_n}{h_{n+1}} && (2.5)\\
\xi_i(n) &= h_n + \cdots + h_{n-i+1} = x_n - x_{n-i} && (2.6)\\
\eta_i(n) &= \frac{h_{n+1}}{\xi_i(n)} && (2.7)\\
\beta_0(n) &= 1 && (2.8)\\
\beta_i(n) &= \frac{\xi_1(n+1)\cdots\xi_i(n+1)}{\xi_1(n)\cdots\xi_i(n)} && (2.9)\\
\varphi_0(n) &= f[x_n] && (2.10)\\
\varphi_i(n) &= \xi_1(n)\cdots\xi_i(n)f[x_n, \cdots, x_{n-i}] && (2.11)
\end{aligned}$$

こうすれば (2.4) 式は次のように書ける.

$$P_{q-1}(x) = \sum_{i=0}^{q-1}\left\{\varphi_i(n)\sum_{j=1}^{i}A_{i,j}(n)\tau^j\right\}$$

ここで $A_{i,j}$ は次の漸化式で求められる.

$$\begin{cases}
A_{i,1}(n) = \eta_i(n) & (i = 1, \cdots, q)\\
A_{i+1,j+1}(n) = \eta_{i+1}(n)\displaystyle\sum_{l=j}^{i}A_{l,j}(n) & (j = 1, \cdots, q-2; i = j, \cdots, q-2)
\end{cases} \tag{2.12}$$

したがって

$$\gamma_{ki}(n) = \begin{cases}
\dfrac{1}{k!} & (i = 0;\ k = 1, \cdots, d)\\
\displaystyle\sum_{j=1}^{i}\frac{j!}{(j+k)!}A_{ij}(n) & (i = 1, \cdots, q-1;\ k = 1, \cdots, d)
\end{cases} \tag{2.13}$$

とすれば (2.2) 式は, 次のように表わされる.

$$p_{n+1}^{(d-k)} = \sum_{i=0}^{k-1}\frac{h_{n+1}^i}{i!}y_n^{(d-k+i)} + h_{n+1}^k\sum_{i=0}^{q-1}\gamma_{k,i}(n)\varphi_i(n) \tag{2.14}$$

同様に修正子 $y_{n+1}^{(d-k)}$ に対しては (2.4) 式より

$$\begin{aligned}
y_{n+1}^{(d-k)} &= p_{n+1}^{(d-k)} + h_{n+1}^k\gamma_{k,q}(n)\frac{\varphi_q(n+1)}{\beta_q(n)} \quad (k = 2, \cdots, d) && (2.15)\\
y_{n+1}^{(d-1)} &= p_{n+1}^{d-1} + h_{n+1}\sum_{i=0}^{q}\gamma_i^*(n)\varphi_i(n+1) && (2.16)
\end{aligned}$$

ここで, $\gamma_i^*(n)$ は以下の式から得られる.

$$\begin{cases}
\gamma_0^*(n) = 1\\
\gamma_i^*(n) = \dfrac{\gamma_{1,i}(n)}{\beta_i(n)} - \dfrac{\gamma_{1,i-1}(n)}{\beta_{i-1}(n)}
\end{cases} \tag{2.17}$$

なお, $q=1$ のときの予測子・修正子は, 次式のようになる.

$$\begin{aligned} p_{n+1}^{(d-k)} &= \sum_{i=0}^{k-1} \frac{h_{n+1}^{i}}{i!} y_n^{(d-k+i)} + \frac{h_{n+1}^{k}}{k!} f(x_n) \\ y_{n+1}^{(d-k)} &= p_{n+1}^{(d-k)} + \frac{h_{n+1}^{k}}{(k+1)!}(f(x_{n+1}) - f(x_n)) \end{aligned} \tag{2.18}$$

これは最初の 2 ステップの計算に用いる.
次数の制御は次のようにして行う.
局所離散化誤差を (2.16) 式から次式により推定する.

$$E = |h_{n+1}\left\{\gamma_q^*(n)\varphi_q(n+1) + \gamma_{q+1}^*(n)\varphi_{q+1}(n+1)\right\}| \tag{2.19}$$

要求精度は, 要求絶対精度と要求相対精度を絶対精度に換算し, 緩い方を採用するため, 次式で求める.

$$\begin{aligned} \varepsilon &= \max\left(\zeta_a, \zeta_r | y_{n+1} + h_{n+1}\frac{p'_{n+1}}{2}|\right) \\ &\quad (\zeta_a : \text{要求絶対精度},\ \ \zeta_r : \text{要求相対精度}) \end{aligned}$$

(2.16) 式の修正子の収束率 P_k を次のように定義する.

$$P_k = \left|\frac{\gamma_{k+1}^*(n)\varphi_{k+1}(n+1)}{\gamma_{k-1}^*(n)\varphi_{k-1}(n+1)}\right|$$

次数 q が大きすぎると伝播誤差が大きくなったり, 新しい誤差成分が加わり, P_k は大きくなる. また, 各変数を次のように定義する.

$$\begin{aligned} C_{\min} &= \min(p_q, p_{q-1}) \\ C_{\max} &= \max(p_q, p_{q-1}) \end{aligned}$$

$$R_c = \begin{cases} 10.0 \times C_{max} & (C_{max} \le 0.09) \\ 0.9 & (C_{min} < 0.09 < C_{max}) \\ 10.0 \times C_{min} & (0.09 \le C_{min} \le 0.105) \\ 1.05 & (C_{min} > 0.105) \end{cases}$$

各連立成分につき

(a) $\frac{E}{\varepsilon} > 0.01$ かつ $C_{max} < 0.025$, または, $d > 1$ かつ $C_{max} < 0.0625$:
q を 1 増加させる.
ただし, 最初の 9 回は条件を $\frac{E}{\varepsilon} > 0.01$ かつ $C_{max} < 0.09$ の場合に変更して適用する.

(b) $q > 1$ かつ, $C_{min} > 0.5$ または $\frac{E}{\varepsilon} < 0.001 \times P_q$: q を 1 減少させる.
ただし, 最初の 9 回の反復では減少させない.

きざみ幅の制御は次のようにして行う.
連立の数を e として

$$R_M = \max_j(10.0 \times \frac{E}{\varepsilon}, R_c) \quad (j = 1, \cdots, e)$$

(ここで, 10 進 1 ケタの余裕を考え 10.0 倍している)

$$\begin{aligned} \gamma &= \begin{cases} 1 + q(j_{max}) & (R_M \ge 1) \\ \max(q(j)) + \max(d(j)) & (R_M < 1) \end{cases} \\ h_{n+2} &= h_{n+1}(R_M)^{-\frac{1}{\gamma}} \end{aligned} \tag{2.20}$$

最初のきざみ幅は, 任意の h から出発し, $0.125 \leq R_M \leq 3$ になるまで (2.20) 式で h を調整する.
計算手順をまとめると以下のようになる.

(a) 最初は (2.18) 式を用いる.

(b) $\eta_i(n)$ を (2.7) 式により計算する.

(c) (2.13) 式の $\gamma_{k,i}(n)$ を (2.5)〜 (2.12) 式により計算する.

(d) $p_{n+1}^{(d-k)}$ $(k = d, d-1, \cdots, 1)$ を (2.14) 式により計算する.

(e) $\beta_i(n)$ $(i = 1, 2, \cdots, q)$ を次式により計算する.

$$\begin{aligned} \xi_i(n+1) &= h_{n+1} + \xi_{i-1}(n) \\ \beta_i(n) &= \beta_{i-1}(n)\frac{\xi_i(n+1)}{\xi_i(n)} \end{aligned}$$

また, $\gamma_i^*(n)$ を (2.17) 式により計算する.
さらに, $\gamma_{q+1}^*(n)$ を次式で近似する.

$$\gamma_{q+1}^*(n) = \frac{\{\gamma_q^*(n)\}^2}{\gamma_{q-1}^*(n)}$$

(f) $\varphi_0(n+1)$ を $f[x_{n+1}]$ として, $\varphi_{i+1}(n+1)$ を次式で計算する.

$$\varphi_{i+1}(n+1) = \varphi_i(n+1) - \beta_i(n)\varphi_i(n) \quad (i = 0, \cdots, q)$$

(g) 修正子 $y_{n+1}^{(d-k)}$ を (2.15), (2.16) 式により計算する.

(h) 局所離散化誤差を (2.19) 式より推定し次数 q を調節する.

(i) $\varphi_0(n+1)$ を計算し, $\varphi_i(n+1)$ $(i = 1, \cdots, q)$ を次式により修正する.

$$\varphi_i(n+1) = \varphi_i(n+1) + \left\{ f(x_{n+1}, y_{n+1}, \cdots, y_{n+1}^{(d-1)}) - f(x_{n+1}, p_{n+1}, \cdots, p_{n+1}^{(d-1)}) \right\}$$

(j) きざみ幅を (2.20) 式により調整し, (b) にもどる.

この方法は, 関数評価にかかるコストが大きい場合や要求精度がきびしい場合は最も効率がよい (参考文献 (3), (4), (9), (10) 参照).

(3) テーラー級数法 (陰的方程式への対応処理を含む)

微分方程式

$$f(x, y, y', \cdots, y^{(d)}) = 0 \tag{2.21}$$

$x + h$ 点での $y, y', \cdots, y^{(d-1)}$ の近似を求める方法として

$$\begin{aligned} y(x+h) &= y(x) + hy'(x) + \frac{h^2}{2!}y''(x) + \cdots \\ y'(x+h) &= y'(x) + hy''(x) + \frac{h^2}{2!}y'''(x) + \cdots \\ &\vdots \end{aligned}$$

としたテーラー展開が利用できる. なお, テーラー級数の項数を増し精度を上げるためには (2.21) 式を微分して $y^{(d+1)}$, さらに微分して $y^{(d+2)} \cdots$ と高階微分を含む式を作り, これらより非線形連立方程式を解いて $y^{(d+1)}, y^{(d+2)} \cdots$ を得, これをテーラー展開式に代入し, 求めることになる.

以上の手順をもう少し詳しく述べる.
入力される方程式群を次のようにする.

$$\begin{aligned} f_1(x,y,\cdots,y^{(d)}) &= 0 \\ f_1'(x,y,\cdots,y^{(d)}) &= f_2(x,y,\cdots,y^{(d)},y^{(d+1)}) = 0 \\ &\vdots \end{aligned} \tag{2.22}$$

なお, $f_1(x,\cdots)$ または $f_2(x,\cdots)\cdots$ が簡単に微分できない場合や, 微分できる場合でも入力する方程式の微分の手間を省くために, 自動関数微分を行う機能をもっている. たとえば, $f_1(x,y,\cdots,y^{(d+1)})$ の自動関数微分は

$$\begin{aligned} f_2(x,y,\cdots,y^{(d)},y^{(d+1)}) &= f_1'(x,y,\cdots,y^{(d+1)}) \\ &\simeq \frac{1}{2\delta}\{g_1(x+\delta)-g_1(x-\delta)\} \\ &\quad +\frac{y'}{2\delta}\{g_1(y+\delta)-g_1(y-\delta)\} \\ &\quad +\cdots \\ &\quad +\frac{y^{(d+2)}}{2\delta}\left\{g_1(y^{(d+1)}+\delta)-g_1(y^{(d+1)}-\delta)\right\} \\ &= 0 \end{aligned}$$

として中央差分近似計算をすることにより実現する. ここで, 以下のような省略記号を用いた.

$$\begin{aligned} g_1(x+\delta) &= f_1(x+\delta,y,\cdots,y^{(i)},\cdots,y^{(d+1)}) \\ g_1(y^{(i)}+\delta) &= f_1(x,y,\cdots,y^{(i)}+\delta,\cdots,y^{(d+1)}) \quad (i=0,\cdots,d+1) \end{aligned}$$

δ としては, $\sqrt[3]{\text{誤差判定のための単位}}$ 程度の小さな値を用いる. さらに高次の微分を考え,
$g_2(\cdots), g_3(\cdots), \cdots, g_i(\cdots)$ を同様にして決定することによって, $y^{(d+i)}$ を含む式

$$f_{i+1}(x,y,\cdots,y^{(d+i)}) = 0 \quad (i=1,2,\cdots)$$

を作成することができる.
次に, このようにして作られた方程式群より, $x, y(x), y'(x), \cdots, y^{(d-1)}(x)$ を初期値とし, $y^{(d)}(x), \cdots, y^{(d+i)}(x)$ および $y(x+h), y'(x+h), \cdots, y^{(d-1)}(x+h)$ を未知数として次の非線形連立方程式を作る.

$$\left\{\begin{array}{l} y(x+h)-\left\{y(x)+hy'(x)+\cdots+h^{d+i}\dfrac{y^{(d+i)}(x)}{(d+i)!}\right\}=0 \\ y'(x+h)-\left\{y'(x)+hy''(x)+\cdots+h^{d+i-1}\dfrac{y^{(d+i)}(x)}{(d+i-1)!}\right\}=0 \\ \vdots \\ y^{(d-1)}(x+h)-\left\{y^{(d-1)}(x)+hy^{(d)}(x)+\cdots+h^{i+1}\dfrac{y^{(d+i)}(x)}{(i+1)!}\right\}=0 \\ f_1(x,y(x),\cdots,y^{(d)}(x))=0 \\ \vdots \\ f_{i+1}(x,y(x),\cdots,y^{(d+i)}(x))=0 \end{array}\right. \tag{2.23}$$

この非線形方程式を解き, 得られた $y(x+h),\cdots,y^{(d-1)}(x+h)$ を次の初期値 $y(x),\cdots,y^{(d-1)}(x)$ にし, $x+h$ を次の x にする.
微分方程式が連立している場合は各方程式に対して (2.23) 式に対応する一群の方程式が作られ, これらすべて

を連立させて連立非線形方程式を解くことになる. したがって, たとえば次に示すように, 複数の方程式計算に $y_i^{(d)}\ \ (i=1,\cdots,n)$ を含む陰的な場合,

$$\begin{aligned} &f_1^1(x,y_1,\cdots,y_1^{(d)},\cdots,y_n,\cdots,y_n^{(d)})=0\\ &\vdots\\ &f_1^n(x,y_1,\cdots,y_1^{(d)},\cdots,y_n,\cdots,y_n^{(d)})=0 \end{aligned} \tag{2.24}$$

(ここで, n:連立数) または, 代数方程式と連立する場合でも, 連立するすべての方程式を対象とする非線形連立方程式を解くので解が得られる. さらに, 極端な場合として, 与えられた微分方程式が微分項を含まず, すべて非線形方程式のときや代数方程式のときは, テーラー展開による $y(x+h)$ を含んだ式が展開されないようにし, これら方程式のみからなる非線形連立方程式を作成して解を得る機能も有する.
常微分方程式を含んだ連立方程式のときは, h のきざみ幅で最終点の x まで積分を続ける. 最後には, $y_i^{(d)}(x+h)$ の解を得るために, 次の非線形方程式に得られた $y_i^{(d-i)}(x+h)$ の値を入れ, これを解く.

$$\begin{aligned} &f_1^1(x+h,y_1(x+h),\cdots,\underline{y_1^{(d)}(x+h),}\cdots,\underline{y_n^{(d)}(x+h)})=0\\ &\vdots\\ &f_1^n(x+h,y_1(x+h),\cdots,\underline{y_1^{(d)}(x+h),}\cdots,\underline{y_n^{(d)}(x+h)})=0 \end{aligned} \tag{2.25}$$

ここで, 下線を付加した項が未知数である. 非線形連立方程式の解法は次のようにする.
与えられた方程式が 1 式で連立方程式でない場合, $f(x)=0$ となる x を求める方法として次の反復法を考える.

$$x_{n+1}=x_n+2^{\frac{p-3r-1}{3}}S\sinh^{-1}\{f(x_n)\}\quad (S=\pm1) \tag{2.26}$$

S は $f(x_n)<0$ のとき (このとき $f(x)$ 単調増加すれば $x>x_n$)$S=-1$
$f(x_n)>0$ のとき (このとき $f(x)$ が単調減少すれば $x<x_n$)$S=1$ とする.
r,p の初期値は 0 とし, r は $f(x_n)$ の計算解の符号が変わるたびに 1 増して減速制御し, p は計算解の符号が変わらないとき 1 増して加速制御する.
次のような連立方程式を解く場合は,

$$\begin{cases} f_1(x_1,x_2,\cdots,x_n)=0\\ f_n(x_1,x_2,\cdots,x_n)=0 \end{cases}$$

まず f_1 式で x_1 を未知数とし $x_2\sim x_n$ を初期値として f_1 の (2.26) 式に対応する式を 1 回反復する. 次に f_2 で x_2 を未知数とし, x_1 を前回計算値, $x_3\sim x_n$ を初期値として f_2 の (2.26) 式に対応する式を 1 回反復する. こうして f_n まで計算すれば, また f_1 にもどり上記手順をくり返す. こうして真解に収束させる. ここで $S_i\ \ (i=1,\cdots,n)$ の初期値は, f_i の x_i に対する傾きが正なら $S_i=-1$, 負なら $S_i=1$ と決めておく. しかし, このように初期値を決めても局所的に傾き方向が逆転することもあり, 探査方向を誤ることもある. このために, もし反復が 9 回以上で $|f_i|>10000$ ならば探査方向ミスと判定して $S_i=-S_i$ とし, 初期値をもどし再度連立方程式を解きなおす. さらに, 前回探査方向を修正したにもかかわらず, 再度探査方向ミスと判定すれば, 計算不能として処理を打ち切る.
本ライブラリの関数では, この非線形連立方程式の最大反復回数を 100, 収束判定を

$$\varepsilon=\left\{\begin{array}{l}\text{単精度:}10^{-5}\\ \text{倍精度:}10^{-12}\end{array}\right\}$$

として $x_{n+1}<|\varepsilon x_n|+\varepsilon$ かつ $|f_i(\cdots)|<|10.0\times\varepsilon x_n|+\varepsilon$ のとき収束したものとみなしている.
この方法は, h を進めるたびに連立数の多い非線形方程式を解かなければならず, 効率は悪いが, 陰的な問題や代数方程式・非線形方程式と連立する常微分方程式の場合は他の方法で解くことができないので有効である. さらに, すべてが非線形方程式の場合もニュートン法のような偏微分計算用関数を作る必要がなく, 目的とする式の入力だけでかなり効率良く解が得られる (参考文献 (3), (5) 参照).

(4) **スティフ問題に対するギア法**

1 次から 5 次までのギア法を用いる. 次数と刻み幅の選択は自動制御される.

(a) 予測子・修正子の計算

微分方程式が次に示す形が与えられているとする. ここで, $\boldsymbol{y}$ は N 次元のベクトルとする.

$$\boldsymbol{y}' = \boldsymbol{f}(x, \boldsymbol{y})$$

いま, $x = x_{n-1}$ までの計算解 $\boldsymbol{y}_i \ \ (i = 0, \cdots, n-1)$ が求められていて, $x = x_n$ における計算解を求めるものとする. このとき, 次に示す条件を満たす q 次の補間多項式のベクトル $\boldsymbol{p}_{n-1}(x)$ を作ることができる.

$$\begin{aligned} \boldsymbol{p}_{n-1}(x_{n-i}) &= \boldsymbol{y}_{n-i} \ \ (i = 1, \cdots, q) \\ \boldsymbol{p}'_{n-1}(x_{n-1}) &= \boldsymbol{f}(x_{n-1}, \boldsymbol{y}_{n-1}) \end{aligned}$$

$x = x_n$ における計算解 $\boldsymbol{y}_n$ を使って q 次の補間多項式のベクトル $\boldsymbol{p}_n(x)$ を作ったとき, それが次に示す条件を満たすように $\boldsymbol{y}_n$ を決めようというのが基本的な考え方である.

$$\begin{aligned} \boldsymbol{p}_n(x_{n-i}) &= \boldsymbol{y}_{n-i} \ \ (i = 0, \cdots, q) \\ \boldsymbol{p}'_n(x_n) &= \boldsymbol{f}(x_n, \boldsymbol{y}_n) \end{aligned}$$

$\boldsymbol{p}_{n-1}(x)$ に関する情報は次に示す **Nordsieck** によって考案された行列の形で保持する.

$$Z_{n-1} = \left[\boldsymbol{y}_{n-1}, h\boldsymbol{y}'_{n-1}, h^2 \frac{\boldsymbol{y}''_{n-1}}{2!}, \cdots, h^q \frac{\boldsymbol{y}^{(q)}_{n-1}}{q!} \right]$$

ここで,

$$\begin{aligned} \boldsymbol{y}^{(q)}_{n-1} &= \boldsymbol{p}^{(q)}_{n-1}(x_{n-1}) \\ h &= x_n - x_{n-1} \end{aligned}$$

である.

Z_n の予測子 $Z_n(0)$ を次に示すように定義する.

$$\begin{aligned} Z_{n(0)} &= \left[\boldsymbol{y}_{n(0)}, h\boldsymbol{y}'_{n(0)}, h^2 \frac{\boldsymbol{y}''_{n(0)}}{2!}, \cdots, h^q \frac{\boldsymbol{y}^{(q)}_{n(0)}}{q!} \right] \\ \boldsymbol{y}^{(q)}_{n(0)} &= \boldsymbol{p}^{(q)}_{n-1}(x_n) \end{aligned}$$

このとき, $Z_{n(0)}$ は次に示す式によって計算できる.

$$Z_{n(0)} = Z_{n-1} A \tag{2.27}$$

ここで, A はパスカルの三角形に対応する行列であり, その i, j 成分 $a_{i,j}$ は次に示すように定義される.

$$a_{i,j} = \begin{cases} 0 & (i < j) \\ \dfrac{i!}{j!(i-j)!} & (i \geq j) \end{cases}$$

いま, s に関する多項式 $L_n(s)$ を次に示すように定義する.

$$\begin{aligned} L_n(s) &= \prod_{i=1}^{q} (1 + \frac{s}{d_i}) \\ d_i &= \frac{x_n - x_{n-i}}{h} \end{aligned}$$

そして, $L_n(s)$ の係数ベクトル $\boldsymbol{l}$ を次に示すように定義する.

$$\begin{aligned} \boldsymbol{l} &= [l_0, l_1, \cdots, l_q] \\ L_n(s) &= \sum_{i=0}^{q} l_i s^i \end{aligned}$$

このとき,

$$\begin{aligned} Z_n &= Z_{n(0)} + \boldsymbol{e}_n \boldsymbol{l} \\ \boldsymbol{e}_n &= \boldsymbol{y}_n - \boldsymbol{y}_{n(0)} \end{aligned} \tag{2.28}$$

となることが示される. これの第 1 列を書くと

$$h\boldsymbol{y}'_n = h\boldsymbol{y}'_{n(0)} + (\boldsymbol{y}_n - \boldsymbol{y}_{n(0)})l_1$$

であるから, $\boldsymbol{g}(\boldsymbol{y})$ を次に示すように定義するとき, $\boldsymbol{y}_n$ は $\boldsymbol{g}(\boldsymbol{y}) = 0$ の根として計算される.

$$\boldsymbol{g}(\boldsymbol{y}) = \boldsymbol{y} - \boldsymbol{y}_{n(0)} - (\frac{h}{l_1})(\boldsymbol{f}(x_n, \boldsymbol{y}) - \boldsymbol{y}_{n(0)'})$$

方程式 $\boldsymbol{g}(\boldsymbol{y}) = 0$ を解くために $\boldsymbol{y} = \boldsymbol{y}_{n(0)}$ を出発値としてニュートン法を用いる. すなわち, 次に示す漸化式を用いて計算する.

$$\begin{aligned} \boldsymbol{y}_{n(m+1)} &= \boldsymbol{y}_{n(m)} - P_m^{-1}\boldsymbol{g}(\boldsymbol{y}_{n(m)}) \\ P_m &= 1 - \frac{h}{l_1} J(x_n, \boldsymbol{y}_{n(m)}) \end{aligned}$$

ここで, $J(x, \boldsymbol{y})$ は $\boldsymbol{f}(x, \boldsymbol{y})$ のヤコビ行列であり, 次に示すように定義される.

$$J(x, \boldsymbol{y}) = \begin{bmatrix} \dfrac{\partial f_1(x, \boldsymbol{y})}{\partial y_1} & \cdots & \dfrac{\partial f_1(x, \boldsymbol{y})}{\partial y_N} \\ \vdots & & \\ \dfrac{\partial f_N(x, \boldsymbol{y})}{\partial y_1} & \cdots & \dfrac{\partial f_N(x, \boldsymbol{y})}{\partial y_N} \end{bmatrix}$$

プログラムでは計算時間を節約するため, 可能な限り前回に計算したヤコビ行列をそのまま使用するようになっている. また, 修正子の反復は最大 3 回まで行われる.

(b) 次数ときざみ幅の決定

次に誤差評価およびきざみ幅と次数の制御について記述する.

局所離散化絶対誤差を $E_n(q)$, 局所離散化相対誤差 $R_n(q)$ とするとき, 計算解の受け入れ判定をユーザが与えた 2 つのパラメータ, 要求局所相対精度 E_a, 要求局所絶対精度 E_r を用いて次に示すように行う.

$$\|E_n(q)\| \leq E_a \quad \text{または} \quad \|R_n(q)\| \leq E_r \tag{2.29}$$

ここで, $\|\ \|$ は最大値ノルムである. また, 相対誤差というのは現時点までの計算解の最大値に対する絶対誤差の比を意味する.

(2.29) の条件が満たされたなら, 次のステップで使用するきざみ幅と次数の選択を行う. これは次のようにして行う. 上記の誤差のほかに $q-1$ 次における誤差 $E_n(q-1)$, $R_n(q-1)$ および $q+1$ 次における誤差 $E_n(q+1)$, $R_n(q+1)$ を計算する. これを用いて許容される最大のきざみ幅を計算する. すなわち, 次に示す $\eta_i \quad (i = 1, \cdots, 6)$ のうち最大の値を h の増加率とし, そのときの誤差計算に使った次数を次のステップの次数とする.

$$\begin{aligned} \eta_1 &= \frac{\sqrt[q]{\frac{E_a}{\|E_n(q-1)\|}}}{1.3} \\ \eta_2 &= \frac{\sqrt[q]{\frac{E_r}{\|R_n(q-1)\|}}}{1.3} \\ \eta_3 &= \frac{\sqrt[(q+1)]{\frac{E_a}{\|E_n(q)\|}}}{1.2} \\ \eta_4 &= \frac{\sqrt[(q+1)]{\frac{E_r}{\|R_n(q)\|}}}{1.2} \end{aligned}$$

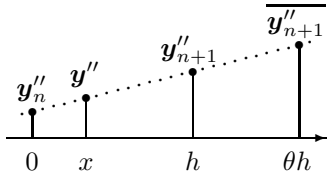
$$\eta_5 = \frac{\sqrt[(q+2)]{\frac{E_a}{\|E_n(q+1)\|}}}{1.4}$$
$$\eta_6 = \frac{\sqrt[(q+2)]{\frac{E_r}{\|R_n(q+1)\|}}}{1.4}$$

一方, 計算解が (2.29) の条件を満たさなかったときは刻み幅を小さくして現在のステップをやりなおす. 積分の出発過程はセルフスタートであり, 1 次の公式によって計算される.

(5) **ウィルソンの θ 法**

$x=0$ と $x=h$ で $\boldsymbol{y}''$ が直線的に変化するものとして

$$\boldsymbol{y}'' = \boldsymbol{y}''_n \frac{h-x}{h} + \boldsymbol{y}''_{n+1}$$



これを x で積分して

$$\begin{aligned}\boldsymbol{y}' &= \boldsymbol{y}'_n + \frac{\boldsymbol{y}'_n}{h}\left(hx - \frac{x^2}{2}\right) + \frac{\boldsymbol{y}''_{n+1}}{h}\frac{x^2}{2} \\ \boldsymbol{y} &= \boldsymbol{y}_n + \boldsymbol{y}'_n x + \frac{\boldsymbol{y}''_n}{h}\left(\frac{hx^2}{2} - \frac{x^3}{6}\right) + \frac{\boldsymbol{y}''_{n+1}}{h}\frac{x^3}{6}\end{aligned}$$

上式で $x=h$ とおくと, $\boldsymbol{y}'$, $\boldsymbol{y}$ は $\boldsymbol{y}'_{n+1}$, $\boldsymbol{y}_{n+1}$ になる. すなわち

$$\boldsymbol{y}'_{n+1} = \boldsymbol{y}'_n + \boldsymbol{y}''_n\frac{h}{2} + \boldsymbol{y}''_{n+1}\frac{h}{2} \tag{2.30}$$
$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + \boldsymbol{y}'_n h + \boldsymbol{y}''_n\frac{h^2}{3} + \boldsymbol{y}''_{n+1}\frac{h^2}{6} \tag{2.31}$$

(2.30), (2.31) 式をもとの方程式に代入すると

$$M\boldsymbol{y}''_{n+1} + C\left\{\boldsymbol{y}'_n + (\boldsymbol{y}''_n + \boldsymbol{y}''_{n+1})\frac{h}{2}\right\} + K\left\{\boldsymbol{y}_n + \boldsymbol{y}'h + (2\boldsymbol{y}''_n + \boldsymbol{y}''_{n+1})\frac{h^2}{6}\right\} = \boldsymbol{p}(h)$$

これを $\boldsymbol{y}''_{n+1}$ について解くと, 次の連立 1 次方程式ができる.

$$\begin{aligned}&\left\{M + \frac{h}{2}C + \frac{h^2}{6}K\right\}\boldsymbol{y}''_{n+1} \\ &= \left[\boldsymbol{p}(h) - C\left\{\boldsymbol{y}'_n + \boldsymbol{y}''_n\frac{h}{2}\right\} - K\left\{\boldsymbol{y}_n + \boldsymbol{y}'_n h + \boldsymbol{y}''_n\frac{h^2}{3}\right\}\right]\end{aligned} \tag{2.32}$$

しかし, ここまでは解が不安定になることが多いので, きざみ幅 h を θ 倍した θh 点について解いて $\overline{\boldsymbol{y}''_{n+1}}$ を求め, 次式により $\boldsymbol{y}''_{n+1}$ を求める.

$$\boldsymbol{y}''_{n+1} = \boldsymbol{y}''_n + \frac{\overline{\boldsymbol{y}''_{n+1}} - \boldsymbol{y}''_n}{\theta} \tag{2.33}$$

この場合, θ は 1.37 以上が良いことが知られているが, 大きすぎると打ち切り誤差が増加し, 精度が悪くなる. たとえば, $\theta=2$ でもこの誤差がかなり顕著に現れる. ウィルソンは, この θ の実用的な値として 1.4 を推奨している.

以上をまとめると, 次に示す手順になる.

(a) 次の連立 1 次方程式を解いて最初の初期値 $\boldsymbol{y}''_n$ を得る.

$$M\boldsymbol{y}''_n = \{\boldsymbol{p}(x) - C\boldsymbol{y}'_n - K\boldsymbol{y}\} \tag{2.34}$$

ただし, もし M 行列の対角成分に 0 を含むことにより連立 1 次方程式が解けない場合は, $\boldsymbol{y}''_n$ を 0 にし, きざみ幅 h を 8 分割し, 以下 (b)~(d) で示すウィルソン θ 法で h 進んだ点の $\boldsymbol{y}''_{n+1}$, $\boldsymbol{y}'_{n+1}$, $\boldsymbol{y}_{n+1}$ を求め (d) に移る.

(b) 次の連立 1 次方程式を解いて θh 進んだ点の $\overline{\boldsymbol{y}''_{n+1}}$ を求める.

$$\begin{aligned}\left\{M + \frac{\theta h}{2}C + \frac{(\theta h)^2}{6}K\right\}\overline{\boldsymbol{y}''_{n+1}} \quad &= \boldsymbol{p}(x) + (\boldsymbol{p}(x+h) - \boldsymbol{p}(x))\theta - C\left\{\boldsymbol{y}'_n + \boldsymbol{y}''_n\frac{\theta h}{2}\right\} \\ &\quad - K\left\{\boldsymbol{y}_n + \boldsymbol{y}'_n\theta h + \boldsymbol{y}''_n\frac{(\theta h)^2}{3}\right\}\end{aligned} \tag{2.35}$$

なお, $\left\{M + \frac{\theta h}{2}C + \frac{(\theta h)^2}{6}K\right\}$ は, きざみ幅 h が変化しない限り一定なので, 最初に 1 回 LU 分解しておき, あとは右辺項を各 x に対して作りなおしながら, 前進代入, 後退代入をしていけば, 各 x での $\overline{\boldsymbol{y}''_{n+1}}$ が得られる.

ここで, もし $\overline{\boldsymbol{y}''_{n+1}}$ と $\boldsymbol{y}''_n$ の差が 1 成分でも 1/(誤差判定のための単位) と比較して著しく大きい場合は, $\boldsymbol{y}''_n$ の初期値設定が不適当と考えられるので, $\boldsymbol{y}''_n$ を 0 にし, きざみ幅 h を 8 分割し, (b)~(d) で示すウィルソン θ 法で h 進んだ点の $\overline{\boldsymbol{y}''_{n+1}}$, $\boldsymbol{y}'_n$, $\boldsymbol{y}_n$ を求め (d) に移る.

(c) $\boldsymbol{y}''_{n+1}$ を (2.33) 式より求める.

(d) $\boldsymbol{y}_{n+1}$ を (2.31) 式より求める.

(e) $\boldsymbol{y}'_{n+1}$ を (2.30) 式より求める.

(f) $\boldsymbol{y}''_n = \boldsymbol{y}''_{n+1}$, $\boldsymbol{y}'_n = \boldsymbol{y}'_{n+1}$, $\boldsymbol{y}_n = \boldsymbol{y}_{n+1}$ として次のきざみにつき (b) から実行する.

この方法は, たとえば運動方程式の場合, M が質量行列, C が減衰行列, K が剛性行列, $\boldsymbol{p}(x)$ が時間 x での作用外力, x が時間, $\boldsymbol{y}''$ が加速度, $\boldsymbol{y}'$ が速度, $\boldsymbol{y}$ が位置に相当し, 地震の応答解析では $\boldsymbol{p}(x) = -M\boldsymbol{y}''_e$ ($\boldsymbol{y}''_e$: 地盤の加速度) と考えることができる (参考文献 (6), (7) 参照).

2.1.2.2 常微分方程式境界値問題

(1) **多点射撃法**

この方法は, 区間内に 2 個以上の射撃点 x を選び, その点での残差を 0 にする初期値を 1 つ手前の射撃点で探して解く方法である.

微分方程式を

$$\begin{aligned} y'_1 &= f_1(x, y_1, y_2, \cdots, y_n) \\ y'_2 &= f_2(x, y_1, y_2, \cdots, y_n) \\ &\vdots \\ y'_n &= f_n(x, y_1, y_2, \cdots, y_n) \end{aligned} \tag{2.36}$$

境界条件を

$$\begin{aligned} g_1(y_1(a), y_2(a), \cdots, y_n(a), y_1(b), y_2(b), \cdots, y_n(b)) &= 0 \\ g_2(y_1(a), y_2(a), \cdots, y_n(a), y_1(b), y_2(b), \cdots, y_n(b)) &= 0 \\ &\vdots \\ g_n(y_1(a), y_2(a), \cdots, y_n(a), y_1(b), y_2(b), \cdots, y_n(b)) &= 0 \end{aligned} \tag{2.37}$$

($x=a$：左側境界, $x=b$：右側境界) とする.
射撃点の個数を n_x, 射撃点 x_i $(i=1,2,\cdots,n_x-1)$ での厳密解 $y_j(x_i)$ $(j=1,2,\cdots,n)$ を近似するベクトルを

$$\boldsymbol{u}_i=(u_{(i)1},u_{(i)2},\cdots,u_{(i)n})$$

とする. $\hat{y}_j(x_i)=u_{(i)j}$ として (2.36) 式より, 次式を得る.

$$\hat{y}_j'=f_j(x,\hat{y}_1,\hat{y}_2,\cdots,\hat{y}_n) \tag{2.38}$$

(2.38) 式よりベクトル $\boldsymbol{u}_i$ を初期値として x_{i+1} まで積分して得られたベクトルの要素を $\hat{y}_j(x_{i+1})$ とする. このときの初期値問題計算にはルンゲ・クッタ・バーナー法 (詳細は 2.1.2 参照) を用いる.
こうすると残差は, 次のようになる.

$$\begin{aligned} r_{(i)j}&=\hat{y}_j(x_{i+1})-u_{(i+1)j}\\ r_{(n_x)j}&=g_j(u_{(1)1},u_{(1)2},\cdots,u_{(1)n},u_{(n_x)1},u_{(n_x)2},\cdots,u_{(n_x)n}) \end{aligned} \tag{2.39}$$

ここでは, 残差 (2.39) を 0 にするベクトル $\boldsymbol{u}_i$ を見つけることを基本としている.
この $n\times n_x$ の連立非線形方程式を解くのにニュートン法を使用する.
ベクトル $\boldsymbol{u}_i$ の修正ベクトルを $\Delta\boldsymbol{u}_i$ とする. この修正ベクトル $\Delta\boldsymbol{u}_i$ は連立 1 次方程式

$$\begin{bmatrix} A_1 & -I & 0 & & 0 & 0\\ 0 & A_2 & -I & & 0 & 0\\ 0 & 0 & A_3 & \cdots & 0 & 0\\ & & \vdots & & & \\ 0 & 0 & 0 & & A_{n_x-1} & -I\\ G_1 & 0 & 0 & & 0 & G_{n_x} \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{u}_1\\ \Delta\boldsymbol{u}_2\\ \Delta\boldsymbol{u}_3\\ \vdots\\ \Delta\boldsymbol{u}_{n_x-1}\\ \Delta\boldsymbol{u}_{n_x} \end{bmatrix} = \begin{bmatrix} -\boldsymbol{r}_1\\ -\boldsymbol{r}_2\\ -\boldsymbol{r}_3\\ \vdots\\ -\boldsymbol{r}_{n_x-1}\\ -\boldsymbol{r}_{n_x} \end{bmatrix} \tag{2.40}$$

を解くことによって得られる. ここで行列 A_i, G_1, G_{n_x} の成分は以下のとおりである.

$$\begin{aligned} (A_i)_{jk}&=\frac{\partial\hat{y}_j(x_{i+1})}{\partial u_{(i)k}}\\ (G_1)_{jk}&=\frac{\partial g_j}{\partial u_{(1)k}}\\ (G_{n_x})_{jk}&=\frac{\partial g_j}{\partial u_{(n_x)k}} \end{aligned} \tag{2.41}$$

この連立 1 次方程式 (2.40) をそのまま解くのは困難なので, 次のように計算する.

$A_{n_x}\leftarrow G_1$
$\Delta\boldsymbol{u}_{n_x}\leftarrow-\boldsymbol{r}_{n_x}$
for $i=1,\cdots,n_x-1$
  $A_{n_x}\leftarrow A_{n_x}A_i^{-1}$
  $\Delta\boldsymbol{u}_{n_x}\leftarrow\Delta\boldsymbol{u}_{n_x}+A_{n_x}\boldsymbol{r}_i$

$A_{n_x}\leftarrow A_{n_x}+G_{n_x}$
$\Delta\boldsymbol{u}_{n_x}\leftarrow A_{n_x}^{-1}\Delta\boldsymbol{u}_{n_x}$
for $i=n_x-1,\cdots,1$
  $\Delta\boldsymbol{u}_i\leftarrow A_i^{-1}(\Delta\boldsymbol{u}_{i+1}-\boldsymbol{r}_i)$

ここで A_i^{-1} $(i = 1, 2, \cdots, n_x - 1)$ は直接逆行列の計算をしないで次のようにして求める.
(2.38) 式を $u_{(i)k}$ に関して微分すると, 次のようになる.

$$\frac{d}{dx}\left(\frac{\partial \hat{y}_j}{\partial u_{(i)k}}\right) = \sum_{p=1}^{n} \frac{\partial f_j}{\partial \hat{y}_p} \frac{\partial \hat{y}_p}{\partial u_{(i)k}} \tag{2.42}$$

そこで,

$$(\hat{A}_i(x))_{jk} = \frac{\partial \hat{y}_j(x)}{\partial u_{(i)k}} \quad (\hat{A}_i(x_i) = I, \quad A_i(x_{i+1}) = A_i)$$

とすれば, (2.42) 式は次のように表せる.

$$\frac{d}{dx}\hat{A}_i = J\hat{A}_i \quad \left((J)_{jp} = \frac{\partial f_j(x, y_1, y_2, \cdots, y_n)}{\partial y_p}\right)$$

それより次式が導かれる.

$$\frac{d}{dx}\hat{A}_i^{-1} = -J\hat{A}_i^{-1} \quad \left(\hat{A}_i^{-1}(x_i) = I\right) \tag{2.43}$$

(2.43) 式を x_{i+1} まで積分することによって $A_i^{-1} = \hat{A}_i^{-1}(x_{i+1})$ を得る. すなわち,

$$\begin{aligned} A_i^{-1} &= \exp((x_i - x_{i+1})J) \\ &\simeq I + B + \frac{B^2}{2!} + \frac{B^3}{3!} + \frac{B^4}{4!} + \frac{B^5}{5!} \end{aligned}$$

(ただし, $B = (x_i - x_{i+1})J$).
以上より, ベクトル $\boldsymbol{u}_i$ $(i = 1, 2, \cdots, n_x)$ の計算手順は, 次のようになる.

(a) $\boldsymbol{u}_i$ の初期値として全成分を 0.1 とする.

(b) 射撃点を定める.
$i \geq b - a$ となる最小の整数 i を定める. このとき, 射撃点の個数 n_x を次式で与える.

$$n_x = \min(2 \times i + 4, 50)$$

射撃点は区間上でこの n_x によって等間隔に与える.

(c) A_i^{-1} $(i = 1, 2, \cdots, n_x - 1)$ を計算する.
行列 A_i の条件数 $\|A_i\|\|A_i^{-1}\|$ に対し,

$$\|A_i\|\|A_i^{-1}\| \geq 2.5$$

が満たされたとき, x_i と x_{i+1} の中点を新しい射撃点として前回の射撃点に加える. そして, そのときの A_i^{-1} を再び計算する.

(d) G_1, G_{n_x} を計算する.

(e) 残差 $\boldsymbol{r}_i$ $(i = 1, 2, \cdots, n_x)$ を計算する.

(f) 修正ベクトル $\Delta\boldsymbol{u}_i$ $(i = 1, 2, \cdots, n_x)$ を計算する.

(g) ベクトル $\boldsymbol{u}_i$ $(i = 1, 2, \cdots, n_x)$ を次の式で更新する.

$$\boldsymbol{u}_i \leftarrow \boldsymbol{u}_i + \Delta\boldsymbol{u}_i$$

(h) 収束判定

ε_r : 要求相対精度

ε_a : 要求絶対精度

(ε_r, ε_a は, 初期値問題を解く場合の要求局所精度に対し, 単精度 5 倍, 倍精度 10 倍とする) とする. すべての i $(i = 1, 2, \cdots, n_x)$, j $(j = 1, 2, \cdots, n)$ に対して

$$\max(|r_{(i)j}|, |\Delta u_{(i)j}|) < \max(\varepsilon_a, \varepsilon_r |u_{(i)j}|)$$

が満たされたときに収束したものとみなす. そうでなければ, (e) に戻って計算し直す.

さて, 非線形問題に対しては, 収束の安定のために次のように計算する.

与えられた問題の非線形になる計算部分に α を乗じておく. 最初にパラメータ α を 0 にして線形化する. 反復回数が 10 になったとき, または収束したときに 1 に戻し, そのときの解を初期値とし, もとの非線形問題を解く. 非線形問題を解くときは A_i^{-1} を反復のたびに計算しなおす (参考文献 (13) 参照).

(2) **選点法**

この方法は, 区間内の点 x を選び, その点で残差を 0 にする方法である. この際, 常微分方程式の近似解を表すのに B-スプラインを基底とするスプライン関数を用いている.

微分方程式を

$$a_1(x)y^{(m)} + a_2(x)y^{(m-1)} + \cdots + a_{m+1}(x)y + a_{m+2}(x) = 0 \tag{2.44}$$

境界条件は左側境界 $x = a$ において

$$y^{(d_i)}(a) = c_1, y^{(d_2)}(a) = c_2, \cdots, y^{(d_h)} = c_h \tag{2.45}$$

右側境界 $x = b$ において

$$y^{(d_{h+1})}(b) = c_{h+1}, y^{(d_{h+2})}(b) = c_{h+2}, \cdots, y^{(d_m)} = c_m \tag{2.46}$$

とする. ただし,

$$0 \le d_1 < d_2 < \cdots < d_h < m$$
$$0 \le d_{h+1} < d_{h+2} < \cdots < d_m < m$$

である. いま, 解を求める区間内に n_x 個の選点 x_i $(i = 1, 2, \cdots, n_x)$ をとり, $y(x_i) = u(x_i)$ をみたす B-スプライン関数 $u(x)$ を求め, これを近似解とすることを考える. $(k-1)$ 次の B-スプライン基底を $B_{i,k}(x)$ とすると,

$$u(x) = \sum_{i=1}^{n_x} e_i B_{i,k}(x) \tag{2.47}$$

と表すことができる. この e_i が求めるべき未知係数であり, 次の方程式から求める.

左側境界 $x = a$ の境界条件 (2.45) 式から,

$$\sum_{i=1}^{n_x} e_i B_{i,k}^{(d_p)}(a) = c_p \quad (p = 1, 2, \cdots, h) \tag{2.48}$$

次に (2.47) 式を (2.44) 式に代入して

$$\sum_{l=1}^{m+1} \sum_{i=1}^{n_x} e_i (a_l(x) B_{i,k}^{(m+1-l)}(x)) + a_{m+2}(x) = 0 \tag{2.49}$$

右側境界 $x = b$ の境界条件 (2.46) 式から,

$$\sum_{i=1}^{n_x} e_i B_{i,k}^{(d_p)}(b) = c_q \quad (q = h+1, \cdots, m) \tag{2.50}$$

以上の式は連立 1 次方程式:

$$A\boldsymbol{x} = \boldsymbol{b} \tag{2.51}$$

で表すことができる. ここで, 行列 A, 右辺ベクトル $\boldsymbol{b}$, 未知ベクトル $\boldsymbol{x}$ は以下のようになる.

$$A = \begin{bmatrix} B_{1,k}^{(d_1)}(a) & \cdots & B_{n_x,k}^{(d_1)}(a) \\ \vdots & & \vdots \\ B_{1,k}^{(d_h)}(a) & \cdots & B_{n_x,k}^{(d_h)}(a) \\ \sum_{l=1}^{m+1} a_l(x_{h+1}) B_{1,k}^{(m+1-l)}(x_{h+1}) & \cdots & \sum_{l=1}^{m+1} a_l(x_{h+1}) B_{n_x,k}^{(m+1-l)}(x_{h+1}) \\ \vdots & & \vdots \\ \sum_{l=1}^{m+1} a_l(x_{n_x-m+h}) B_{1,k}^{(m+1-l)}(x_{n_x-m+h}) & \cdots & \sum_{l=1}^{m+1} a_l(x_{n_x-m+h}) B_{n_x,k}^{(m+1-l)}(x_{n_x-m+h}) \\ B_{1,k}^{(d_{h+1})}(b) & \cdots & B_{n_x,k}^{(d_{h+1})}(b) \\ \vdots & & \vdots \\ B_{1,k}^{(d_m)}(b) & \cdots & B_{n_x,k}^{(d_m)}(b) \end{bmatrix}$$

$$\boldsymbol{x} = \begin{bmatrix} e_1 \\ \vdots \\ e_h \\ e_{h+1} \\ \vdots \\ e_{n_x-m+h} \\ e_{n_x-m+h+1} \\ \vdots \\ e_{n_x} \end{bmatrix} \quad \boldsymbol{b} = \begin{bmatrix} c_1 \\ \vdots \\ c_h \\ -a_{m+2}(x_{h+1}) \\ \vdots \\ -a_{m+2}(x_{n_x-m+h}) \\ c_{h+1} \\ \vdots \\ c_m \end{bmatrix}$$

以上より, 近似解 $u(x)$ の計算手順は, 次のようになる.

(a) 選点の設定
$i \geq b - a$ となる最小の整数 i を定める. このとき, 選点の個数 n_x は次式で与える.

$$n_x = \max(\min(5 \times i + 1, 101), m + 2)$$

選点は区間上でこの n_x によって等間隔に与える.

(b) スプライン次数の設定
スプライン次数の選点の個数に比例して設定する. 最初に $i \geq \dfrac{n_x}{10}$ となる最小の整数 i を定める. このとき, スプライン次数 j_s は

$$j_s = \max(i + 3, m)$$

とする.

(c) 節点の設定

$k = j_s + 1$ とすると, 必要な節点の個数は $n_x + k$ であり, この節点を次のように設定する.

$$q_1 = q_2 = \cdots = q_k = x_1$$
$$q_{i+k} = \frac{x_i + x_{i+k}}{2} \quad (i = 1, 2, \cdots, n_x - k)$$
$$q_{n_x+1} = q_{n_x+2} = \cdots = q_{n_x+k} = x_{n_x}$$

(d) 選点での B-スプラインの設定

選点 $x_i \quad (i = 1, 2, \cdots, n_x)$ での $(k-1)$ 次の B-スプライン $B_{j,k}(x_i) \quad (j = 1, 2, \cdots, n_x)$ は漸化式

$$B_{j,k}(x) = (\frac{x_i - q_i}{q_{j+k-1} - q_j})B_{j,k-1}(x_i) + (\frac{q_{j+k} - x_i}{q_{j+k} - q_{j+1}})B_{j+1,k-1}(x_i) \tag{2.52}$$

で表される. ここで, 出発値としての 0 次の B-スプラインを

$$B_{j,1}(x_i) \quad = \quad \begin{cases} 1 & (q_j \leq x_i < q_{j+1}) \\ 0 & (x_i < q_j, x_i \geq q_{j+1}) \end{cases}$$

と定める.

(e) 選点での B-スプラインの微分

(2.52) 式を選点 $x_i \quad (i = 1, 2, \cdots, n_x)$ について微分すると, 次の漸化式が得られる.

$$B'_{j,k}(x_i) = (k-1)(\frac{B_{j,k-1}(x_i)}{q_{j+k-1} - q_j} - \frac{B_{j+1,k-1}(x_i)}{q_{j+k} - q_{j+1}}) \tag{2.53}$$

同様に両辺を順次微分し, (2.53) 式によって 1 階微分を順次消去することによって, B-スプラインの高階微分を求める.

(f) 連立方程式 (2.51) の係数行列を求め, e_i を決定

(d), (e) で計算された

$$B_{j,k}(x_i), B'_{j,k}(x_i), \cdots, B^{(m)}_{j,k}(x_i) \quad (i = 1, 2, \cdots, n_x;\ j = 1, 2, \cdots, n_x)$$

を連立方程式 (2.51) の右辺の係数行列に代入する. 以上によって連立方程式 i(2.51) が作成され, これを解くことによって線形結合係数 $e_i \quad (i = 1, 2, \cdots, n_x)$ が求められる.

(g) 近似解の計算

(2.47) 式より, n_x 個の選点で厳密解を通る近似解を得る.

(h) 収束判定

近似解の曲線が厳密解の曲線から離れる割合をとらえる指標 $\gamma_j \quad (j = 1, 2, \cdots, n_x - 1)$ を次のように定める.

$$\gamma_j = \left| \frac{\Delta\delta_j}{\delta Y_j} \right|$$

ここで,

$|\delta Y_j|$: 選点 x_j, x_{j+1} の中点において計算した常微分方程式の各項のうち絶対値が最大となる値

$$|\delta Y_j| \quad = \quad \max_{l=1,2,\cdots,m+1} \left(|\sum_{i=1}^{n_x} e_i a_l (\frac{x_j + x_{j+1}}{2}) B^{(m+1-l)}_{i,k} (\frac{x_j + x_{j+1}}{2})|, |a_{m+2}(\frac{x_j + x_{j+1}}{2})| \right)$$

$|\Delta\delta_j|$: 中点における常微分方程式の残差の絶対値

$$|\Delta\delta_j| = |\sum_{l=1}^{m+1} \sum_{i=1}^{n_x} e_i \left(a_l (\frac{x_j + x_{j+1}}{2}) B^{(m+1-l)}_{i,k} (\frac{x_j + x_{j+1}}{2}) \right) + a_{m+2}(\frac{x_j + x_{j+1}}{2})|$$

である.

要求相対精度を ε_r, 要求絶対精度を ε_a とする.

$$\gamma_j \leq \varepsilon_r \text{または} |\Delta\delta_j| \leq \varepsilon_a \quad (j = 1, 2, \cdots, n_x - 1) \tag{2.54}$$

であれば, 収束したものとみなす. そうでなければ, 条件 (2.54) を満たさなかった中点を新しい選点として前回の選点に加え, (b) に戻って計算し直す (参考文献 (14) 参照).

(3) **係数決定法**

微分方程式を

$$y'' + a_1(x)y' + a_2(x)y + a_3(x) = 0 \tag{2.55}$$

とする. (2.55) の解を未定係数 c を用いて

$$y(x) = y_1(x) + cy_2(x) \tag{2.56}$$

とおき, 両辺を微分すると

$$y' = y_1'(x) + cy_2'(x) \tag{2.57}$$
$$y'' = y_1''(x) + cy_2''(x) \tag{2.58}$$

を得る. これらを (2.55) に代入して整理することによって次の (2.59), (2.60) を満たす解 y_1, y_2 が (2.55), (2.56) を満たすことがわかる.

$$y_1'' + a_1(x)y_1' + a_2(x)y_1 + a_3(x) = 0 \tag{2.59}$$
$$y_2'' + a_1(x)y_2' + a_2(x)y_2 = 0 \tag{2.60}$$

従って, ここでは (2.55) のかわりに (2.59), (2.60) を解くことを考える.
さて, 境界条件は左側境界 $x = a$, 右側境界 $x = b$ それぞれについて次のように 4 ケースある.

$$y(a) = ya_0 \quad , \quad y(b) = yb_0 \tag{2.61}$$
$$y(a) = ya_0 \quad , \quad y'(b) = yb_1 \tag{2.62}$$
$$y'(a) = ya_1 \quad , \quad y(b) = yb_0 \tag{2.63}$$
$$y'(a) = ya_1 \quad , \quad y'(b) = yb_1 \tag{2.64}$$

ここで, ya_0, ya_1, yb_0, yb_1 は定数.
未定係数 c を求める計算の手順を以下に示す.

(a) (2.59) の初期値問題計算

境界条件 (2.61), (2.62) のとき, (2.59) を初期値

$$y_2(a) = 0 \quad , \quad y_2'(a) = 1$$

で解くと, $y_2(b)$, $y_2'(b)$ が求まる.
境界条件 (2.63), (2.64) のとき, (2.59) を初期値

$$y_2(a) = 1 \quad , \quad y_2'(a) = 0$$

で解くと, $y_2(b)$, $y_2'(b)$ が求まる.

(b) (2.60) の初期値問題計算

境界条件 (2.61), (2.62) のとき, (2.56) 式より

$$y_1(a) = y(a) - cy_2(a)$$
$$= ya_0$$

となるので, (2.60) を初期値

$$y_1(a) = ya_0 \quad , \quad y_1'(a) = 0$$

で解くと, $y_1(b)$, $y_1'(b)$ が求まる.
境界条件 (2.63), (2.64) のとき, (2.57) 式より

$$\begin{aligned} y_1'(a) &= y'(a) - cy_2'(a) \\ &= ya_1 \end{aligned}$$

となるので, (2.60) を初期値

$$y_1(a) = 0 \quad , \quad y_1'(a) = ya_1$$

で解くと, $y_1(b)$, $y_1'(b)$ が求まる.

(c) 未定係数 c の計算
境界条件 (2.61), (2.63) のとき, (2.56) 式より

$$c = \frac{yb_0 - y_1(b)}{y_2(b)}$$

境界条件 (2.62), (2.64) のとき, (2.57) 式より

$$c = \frac{yb_1 - y_1'(b)}{y_2'(b)}$$

以上によって得られた c を用いて (2.56), (2.57), (2.58) により $y(x)$, $y'(x)$, $y''(x)$ を求める. なお, 初期値問題計算にはルンゲ・クッタ・バーナー法 (詳細は 2.1.2 参照) を用いている.
また, $y_2(b)$ または $y_2'(b)$ が 0 になるときは, 右側境界から左側境界に向けた初期値問題を作り, これから未定係数 c を求めるようにする. それでも分母が 0 になるときは, エラーとする (参考文献 (15) 参照).

2.1.2.3 積分方程式

(1) **第 2 種フレドホルム型積分方程式**
第 2 種フレドホルム型積分方程式は, 次式で表される.

$$y(t) - \int_a^b K(t,x)y(x)dx = f(t)$$

ただし

$$\begin{aligned} f(t) &: t \text{ の変域 } [a,b] \text{ で連続な既知関数} \\ K(t,x) &: \text{核 } (t \text{ および } x \text{ の変域 } [a,b] \text{ で連続な既知関数}) \\ y(t) &: \text{求めるべき未知関数} \end{aligned}$$

である. 以下, 核 K の t および x に関する導関数が連続な場合 (核 K が正則核) の解法を説明する. 方程式の左辺第 2 項を数値積分で近似すると,

$$\int_a^b K(t,x)y(x)dx \simeq \sum_{i=1}^{n} W_i K(t,x_i)y(x_i)$$

で表される. ただし, W_i, x_i, n は, 数値積分の公式によって定まる定数である. ここでは, ガウスの積分公式 (30 点公式) を用いているので,

i		W_i	X_i
1	30	0.00796819249616661	0.996893484074650
2	29	0.0184664683110910	0.983668123279747
3	28	0.0287847078833234	0.960021864968308
4	27	0.0387991925696270	0.926200047429274
5	26	0.0484026728305941	0.882560535792053
6	25	0.0574931562176191	0.829565762382768
7	24	0.0659742298821805	0.767777432104826
8	23	0.0737559747377052	0.697850494793316
9	22	0.0807558952294202	0.620526182989243
10	21	0.0868997872010830	0.536624148142020
11	20	0.0921225222377861	0.447033769538089
12	19	0.0963687371746443	0.352704725530878
13	18	0.0995934205867953	0.254636926167890
14	17	0.101762389748406	0.153869913608584
15	16	0.102852652893559	0.051471842555318

(ただし, $i = 1 \cdots 15$ に対する X_i には負号がつく.)

$$x_i = \frac{\|a-b\,|}{2} X_i + \frac{a+b}{2}$$

$$n = 30$$

である. この近似によって, 第 2 種フレドホルム型積分方程式は,

$$\begin{bmatrix} 1-W_1K(x_1,x_1) & -W_2K(x_1,x_2) & \cdots & -W_{30}K(x_1,x_{30}) \\ W_1K(x_2,x_1) & 1-W_2K(x_2,x_2) & \cdots & -W_{30}K(x_2,x_{30}) \\ \vdots & \vdots & \ddots & \vdots \\ W_1K(x_{30},x_1) & 1-W_2K(x_{30},x_2) & \cdots & 1-W_{30}K(x_{30},x_{30}) \end{bmatrix} \begin{bmatrix} y(x_1) \\ y(x_2) \\ \vdots \\ y(x_{30}) \end{bmatrix} = \begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_{30}) \end{bmatrix}$$

と表される. この 30 次の連立 1 次方程式をガウスの消去法によって解く. 得られた $y(x_1) \sim y(x_{30})$ の値を用いて, 任意の値 t における $y(t)$ を 3 次スプライン関数を用いた補間 (詳細は 6.1.2 参照) により求める.

(2) **第 1 種ボルテラ型積分方程式**

第 1 種ボルテラ型積分方程式は, 次式で表される.

$$f(t) = \int_a^t K(t,x)y(x)dx$$

ただし,

$f(t)$: t の変域 $[a,\infty)$ で有限, 連続な既知関数

$K(t,x)$: 核 (t および x の変域 $[a,\infty)$ で有限, 連続, あるいは $x=\alpha$で無限大となる既知関数)
$$\int_a^x K(x,y)dy < \infty$$

$y(t)$: 求めるべき未知関数

である. 以下, 核 K の t および x に関する導関数が連続な場合 (核 K が正則核) の解法を説明する. 積分区間 $[a, a+h]$ に対して, マクローリンの公式を適用すると,

$$f(a+h) = \int_a^{a+h} K(a+h,x)y(x)dx = K(a+h, a+\frac{h}{2}) \cdot y(a+\frac{h}{2}) \cdot h$$

となるから, $y(a+\frac{h}{2})$ は,

$$y(a+\frac{h}{2}) \simeq \frac{f(a+h)}{K(a+h, a+\frac{h}{2}) \cdot h}$$

で与えられる. 積分区間 $[a, a+2h], [a, a+3h], \cdots, [a, a+6h]$ に対してマクローリンの公式を適用すれば, 次の連立 1 次方程式が導かれる.

$$f(a+2h) \simeq 2h\{K(a+2h, a+\frac{h}{2})y(a+\frac{h}{2}) + K(a+2h, a+\frac{3}{2}h)y(a+\frac{3}{2}h)\}$$

$$\begin{aligned} f(a+3h) \simeq\ & 3h\{\frac{3}{8}K(a+3h, a+\frac{h}{2})y(a+\frac{h}{2}) + \frac{2}{8}K(a+3h, a+\frac{3}{2}h)y(a+\frac{3}{2}h) \\ & + \frac{3}{8}K(a+3h, a+\frac{5}{2}h)y(a+\frac{5}{2}h)\} \end{aligned}$$

$$\begin{aligned} f(a+4h) \simeq\ & 4h\{\frac{13}{48}K(a+4h, a+\frac{h}{2})y(a+\frac{h}{2}) + \frac{11}{48}K(a+4h, a+\frac{3}{2}h)y(a+\frac{3}{2}h) \\ & + \frac{11}{48}K(a+4h, a+\frac{5}{2}h)y(a+\frac{5}{2}h) + \frac{13}{48}K(a+5h, a+\frac{7}{2}h)y(a+\frac{7}{2}h)\} \end{aligned}$$

$$\begin{aligned} f(a+5h) \simeq\ & 5h\{\frac{275}{1152}K(a+5h, a+\frac{h}{2})y(a+\frac{h}{2}) + \frac{100}{1152}K(a+5h, a+\frac{3}{2}h)y(a+\frac{3}{2}h) \\ & + \frac{402}{1152}K(a+5h, a+\frac{5}{2}h)y(a+\frac{5}{2}h) + \frac{100}{1152}K(a+5h, a+\frac{7}{2}h)y(a+\frac{7}{2}h) \\ & + \frac{275}{1152}K(a+5h, a+\frac{9}{2}h)y(a+\frac{9}{2}h)\} \end{aligned}$$

$$\begin{aligned} f(a+6h) \simeq\ & 6h\{\frac{247}{1280}K(a+6h, a+\frac{h}{2})y(a+\frac{h}{2}) + \frac{139}{1280}K(a+6h, a+\frac{3}{2}h)y(a+\frac{3}{2}h) \\ & + \frac{254}{1280}K(a+6h, a+\frac{5}{2}h)y(a+\frac{5}{2}h) + \frac{254}{1280}K(a+6h, a+\frac{7}{2}h)y(a+\frac{7}{2}h) \\ & + \frac{139}{1280}K(a+6h, a+\frac{9}{2}h)y(a+\frac{9}{2}h) + \frac{247}{1280}K(a+6h, a+\frac{11}{2}h)y(a+\frac{11}{2}h)\} \end{aligned}$$

これを解いて, $y(a+\frac{3}{2}h), y(a+\frac{5}{2}h), y(a+\frac{7}{2}h), y(a+\frac{9}{2}h), y(a+\frac{11}{2}h)$ を求める. 次に, 積分区間 $[a+5h, a+7h], [a+5h, a+8h], \cdots, [a+5h, a+11h]$ に対して同様な連立 1 次方程式を導くことによって, $y(a+\frac{13}{2}h), y(a+\frac{15}{2}h), \cdots, y(a+\frac{21}{2}h)$ を求めることができる.

以下, 同様にすれば $y(a+\frac{2j-1}{2}h)(j=1, \cdots, n)$ の各点の値を求めることができる. 得られた y の値を用いて, 任意の値 x における $y(x)$ を 3 次スプライン関数を用いた補間 (詳細は 6.1.2 参照) により求める.

2.1.2.4 偏微分方程式

(1) **差分近似**

本ライブラリでは, 以下の偏微分方程式を差分法を用いて解く.

$$\nabla^2 + \lambda u = f \tag{2.65}$$

空間 2 次元の 2 変数関数 $u(x, y)$ と空間 3 次元の 3 変数関数 $u(x, y, z)$ は, その偏導関数の十分な微分可能性と連続性を仮定する. また, u_{xx}, u_{yy} はそれぞれ 2 変数関数 $u(x, y)$ の x と y に関する 2 階偏導関数である.

2 変数の関数 $u(x + h, y)$ と関数 $u(x - h, y)$ をそれぞれ x について Taylor 級数に展開すると,

$$u(x+h,y) \;=\; u(x,y) + u_x(x,y)h + u_{xx}(x,y)\frac{h^2}{2!} + \cdots$$

$$u(x-h,y) \;=\; u(x,y) - u_x(x,y)h + u_{xx}(x,y)\frac{h^2}{2!} - \cdots$$

となる. 同様に 2 変数の関数 $u(x, y + k)$ と関数 $u(x, y - k)$ もそれぞれ y について Taylor 級数に展開する.

$$u(x,y+k) \;=\; u(x,y) + u_y(x,y)h + u_{yy}(x,y)\frac{k^2}{2!} + \cdots$$

$$u(x,y-k) \;=\; u(x,y) - u_y(x,y)h + u_{yy}(x,y)\frac{k^2}{2!} - \cdots$$

上式より, $u_{xx}(x, y)$ と $u_{yy}(x, y)$ に対する中心差分近似が求められる.

$$u_{xx}(x,y) \cong \frac{1}{h^2}[u(x+h,y) - 2u(x,y) + u(x-h,y)]$$

$$u_{yy}(x,y) \cong \frac{1}{k^2}[u(x,y+k) - 2u(x,y) + u(x,y+k)]$$

この中心差分近似式を用いると, 式 (2.65) は

$$\frac{1}{h^2}[\; u(x+h,y) - 2u(x,y) + u(x-h,y) \;]+$$

$$\frac{1}{k^2}[\; u(x,y+k) - 2u(x,y) + u(x,y+k) \;] + \lambda\; u(x,y) = \; f(x,y) \tag{2.66}$$

となる. 3 次元の場合も同様に, $u(x+h, y, z), u(x-h, y, z), u(x, y+k, z), u(x, y-k, z), u(x, y, z-l), u(x, y, z+l)$ をそれぞれ x, y, z について Taylor 級数に展開し中心差分近似を求め, 式 (2.65) に代入すると式 (2.67) が導かれる.

$$\frac{1}{h^2}[\; u(x+h,y,z) - 2u(x,y,z) + u(x-h,y,z) \;]+$$

$$\frac{1}{k^2}[\; u(x,y+k,z) - 2u(x,y,z) + u(x,y+k,z) \;]+$$

$$\frac{1}{l^2}[\; u(x,y,z+l) - 2u(x,y,z) + u(x,y,z+l) \;] + \lambda\; u(x,y,z) = \; f(x,y,z) \tag{2.67}$$

式 (2.66), 式 (2.67) を離散化された領域内の各格子点で計算すると, 各格子点での u の値を変数とする連立 1 次方程式になる. その連立 1 次方程式を解くと u の値が求まる.

(2) **境界条件の差分近似**

本ライブラリにおける Dirichlet 問題と Neumann 問題の境界条件の扱いについて説明する. いずれの場合も与えられた領域の外側に仮想格子を設け, その仮想格子上に境界条件を付加する. Dirichlet 条件は仮想格子上に u の値を与え, Neumann 条件は u の外向き法線方向の微分係数 $\frac{\partial u}{\partial n}$ を与える.

本ライブラリ内では, 例えば離散化された矩形領域の第一番目の格子点 $(i, j) = (1, 1)$ において, 式 (2.66) で領域の下辺の境界条件は,

$$\begin{cases} \text{Dirichlet 条件の場合} & u(i, 0) = \text{s} \quad (\text{s : 境界条件 u の値}) \\ \text{Neumann 条件の場合} & u(i, 1) = u(i, 0) \end{cases}$$

として処理される.

2.1.3 参考文献

(1) Forsythe, Malcolm and Moler, 森正武訳, “計算機のための数値計算法”, 科学技術出版社, (1978)

(2) Verner, J. H. , “Explicit Runge–Kutta methods with estimate of the local truncation error”, SIAM J. Numer. Anal. Vol.15, No.4, pp.618-641, (1978).

(3) 三井斌友, “数値解析入門”, 朝倉書店, (1985)

(4) Krogh, F. T. , “A Variable Step Variable Order Multistep Method for the Numerical Solution of Ordinary Differential Equations”, Information Processing 68, North-Holand Pub. Co. , pp. 194-199, (1968)

(5) Kantaris, N. and Howden, P. F. , “The Universal Equation Solver”, SIGMA PRESS, (1983)

(6) 大地羊三, “耐震計算法入門”, 鹿島出版会, (1984)

(7) 戸川隼人, “有限要素法による振動解析”, サイエンス社, (1975)

(8) Gupta, G. K. , Sacks-Davis, R. & Tischer, P. E. , “A Review of Recent Development in Solving ODEs”, ACM Comp. Surveys, Vol.17, No.1, pp.5-47, (1985)

(9) Shampine, L. F. and Gordon, M. K. , “Computer Solution of Ordinary Differential Equations”, Freeman, 1975

(10) 清水留三郎, “常微分方程式の適応的解法”, 情報処理研修センター, (1981)

(11) Shampine, L. F. , Watts, H. A. and Davenport, S. M. “Solving Nonstiff Ordinary Differential Equations–the State of the Art”, SIAM Review, Vol.18, No.3, pp.376-411, (1976)

(12) Byrne, G. D. and Hindmarsh, A. C. , “A Polyalgorithm for the Numerical Solution of Ordinary Differential Equations”, ACM Trans. Math. Softw., Vol.1, pp.71–96, (1975).

(13) Hindmarsh, A. C. and Byrne, G. D. , “EPISODE : An Effective Package for the Integration of Systems of Ordinary Differential Equations”, UCID-30112, Rev.1, Lawrence Livermore Laboratory, (1977)

(14) 桜井明監修, 吉村和美, 高山文雄, “パソコンによるスプライン関数”, 東京電機大学出版局, (1988)

(15) 長田純一, 東田幸樹, 山下正, 山本芳人, “PL/I による数値計算入門”, 東京理科大学出版会, (1985)

(16) R. F. Churchhouse, ed. , “Handbook of Applicable Mathematics, vol. III”, John Wiley & Sons Inc. (1981)

(17) 磯田和男, 大野豊, “FORTRAN による数値計算ハンドブック”, オーム社

(18) 山内二郎, 宇野利雄, 一松信 共編 “電子計算機のための数値計算法 III”, オーム社

(19) スタンリー・ファーロー, “偏微分方程式”, 伊理正夫 他訳, pp. 308-314(1983).

(20) 高橋亮一・棚町芳弘共著, “差分法”, 培風館, pp. 68-75(1992).

2.2 常微分方程式初期値問題

2.2.1 ASL_dksncs, ASL_rksncs
連立高階常微分方程式 (速度優先)

(1) **機 能**

自動きざみ幅制御のもとで, 関数評価コストが少なく要求精度が厳しくない場合の常微分方程式初期値問題を解く.

(2) **使用法**

倍精度関数:

ierr = ASL_dksncs (f, & x, y, n, mx, m, xf, er, ea, & nst, isr, wk);

単精度関数:

ierr = ASL_rksncs (f, & x, y, n, mx, m, xf, er, ea, & nst, isr, 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	−	−	入 力	x, y の関数として微分方程式を定義する 関数 f(x, y, n) の関数名
2	x	{D∗ / R∗}	1	入 力	独立変数 x の初期値 x_0
				出 力	独立変数 x の最終到達点 x_e
3	y	{D∗ / R∗}	内容参照	入 力	$x = x_0$ での初期値 $y_i^{(j)}$ $(i = 1, \cdots, \mathrm{n};\ j = 0, \cdots, \mathrm{m}[i-1] - 1)$. $\mathrm{y}[(i-1) + n \times j] = y_i^{(j)}$. **大きさ:** $\mathrm{n} \times (\mathrm{mx} + 1)$
				出 力	$x = x_e$ での計算解 $y_i^{(j)}$ $(i = 1, \cdots, \mathrm{n};\ j = 0, \cdots, \mathrm{m}[i-1])$
4	n	I	1	入 力	方程式の連立数
5	mx	I	1	入 力	最大微分階数 $(\max(\mathrm{m}[i-1]))$
6	m	I*	n	入 力	連立する各方程式の左辺の微分階数 $\mathrm{m}[i-1]$
7	xf	{D / R}	1	入 力	求めたい解の最終点 x_f
8	er	{D / R}	1	入 力	要求局所相対精度 既定値: 倍精度: 10^{-12}, 単精度: 10^{-5}
9	ea	{D / R}	1	入 力	要求局所絶対精度 (既定値 : 表現できる絶対値最小値 $\times 2^{24}$)
10	nst	I*	1	入 力	ステップ回数初期値 (最初に呼ぶときは 0 を入力する)
				出 力	計算総ステップ回数 (続けて積分するときの入力値)
11	isr	I	1	入 力	0 : x_f まで積分を行い出力. エラーが発生しなければ $x_e = x_f$. 0 以外 : $x_0 \sim x_f$ の間, 自動きざみ幅進むごとに x_e と $y_i^{(j)}$ を出力する. 最後に $x_e = x_f$ として x_f 点で出力する.
12	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 wk [0] には, 最後に使われたきざみ幅が入る **大きさ:** $8 \times \mathrm{n} \times (\mathrm{mx} + 1) + 1$
13	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $\mathrm{n} \geq 1, \mathrm{nst} \geq 0$

(b) $\mathrm{m[i-1]} \geq 1, \mathrm{mx} \geq \max(\mathrm{m[i-1]})$ $(\mathrm{i} = 1, \cdots, \mathrm{n})$

(c) er$\geq e_r$. ここで e_r= 倍精度: 10^{-14}, 単精度: 10^{-5} (既定値にするため, 0.0 を入力する場合は除く)

(d) ea$\geq$ 表現できる絶対値最小値 $\times 2^{24}$ (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 (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	計算中きざみ幅が小さくなりすぎた.	その時点の x_e, $y_i^{(j)}$ の値を出力し, 処理を打ち切る.

(6) **注意事項**

(a) x, y の関数として微分方程式を定義する関数 f(x, y, n) の実際の名前は, 使用者側のプログラムで宣言し, 実際の関数を作成しておかなければならない. 連立高階常微分方程式

$$\left\{\begin{array}{rcl} y_1^{(m_1)} & = & f_1(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)}) \\ \vdots & \vdots & \vdots \\ y_i^{(m_i)} & = & f_i(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)}) \\ \vdots & \vdots & \vdots \\ y_n^{(m_n)} & = & f_n(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)}) \end{array}\right.$$

(ただし, 左辺が $y_i^{(m_i)}$ である場合対応する右辺の $y_i^{(j)}$ の微分階数 j は $j \leq m_i - 1$ を満たさなければならない.) を解く場合, この関数 f(x, y, n)(倍精度) の作り方は, 次に示すとおりである.

void FORTRAN f(double *x, double *y, int *n)
{
$\mathrm{y[(*n) * m[0]]} = f_1(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)});$
$\vdots$
$\mathrm{y[i-1+(*n) * m[i-1]]} = f_i(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)});$
$\vdots$
$\mathrm{y[(*n)-1+(*n) * m[(*n)-1]]} = f_n(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)});$
}

ただし, $x \leftrightarrow$ *x, $n \leftrightarrow$ *n, $y_i \leftrightarrow \mathrm{y[i-1]}$, $y_i^{(j)} \leftrightarrow \mathrm{y[i-1+(*n) * j]}$
と対応する.

例

$$\left\{\begin{array}{ll} 8(y_1'')^2 - 2 - y_2 y_1' = 0 & (y_1'' > 0) \cdots\cdots \text{①} \\ (y_2')^2 - 1 - (y_1')^2 = 0 & (y_2' > 0) \cdots\cdots \text{②} \end{array}\right.$$

①式より $y_1'' = \sqrt{\frac{1}{4} + \frac{1}{8} \cdot y_2 \cdot y_1'}$
②式より $y_2' = \sqrt{1 + (y_1')^2}$
この場合関数はつぎのようになる

void FORTRAN f(double ∗x, double ∗y, int ∗n)
{
y[4] = sqrt(0.25 + 0.125 ∗ y[1] ∗ y[2]);
y[3] = sqrt(1.0 + y[2] ∗ y[2]);
}

入力引数は n=2, mx=2, m [0] =2, m [1] =1 となる. また y [0] , y [2] , y [1] に初期値を与える.

(b) 初めて積分するときは nst=0 にすること.

(c) 続けて積分している間は x, y, nst の出力値をそのまま次の入力値とすること.

(d) ierr=4000 のときは x_e の点から 2.2.4 $\left\{\begin{array}{l}\text{ASL_dkssca}\\ \text{ASL_rkssca}\end{array}\right\}$を用いるか, 要求精度を緩くすれば続けて解くことができる.

(e) この関数は, ルンゲ・クッタ・バーナー法を利用している.

(7) **使用例**

(a) 問 題

$$\left\{\begin{array}{l} y_1'' = -\frac{y_1}{\gamma} \\ y_2'' = -\frac{y_2}{\gamma} \quad \gamma = (y_1^2 + y_2^2)^{\frac{3}{2}} \end{array}\right.$$

の連立 2 階常微分方程式を $x = 0.0$ における初期条件

$$y_1(0.0) = 1.0, y_1'(0.0) = 0.0, y_2(0.0) = 0.0, y_2'(0.0) = 1.0$$

のもとで解く.

(b) 入力データ

関数 f(x, y, n) の関数名: f, x=0.0 n=2,
y [0] =1.0, y [2] =0.0,
y [1] =0.0, y [3] =1.0,
mx=2, m [0] =2, m [1] =2, xf, er, ea, nst=0, isr=0

(c) 主プログラム

```
/*      C interface example for ASL_dksncs */

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
void    f(double *x,double *y,int *n)
#else
void    f(x,y,n)
double *x;
double *y;
int    *n;
#endif
{
    double r;

    r= pow(y[0]*y[0]+y[1]*y[1],1.5);
    y[2*(*n)]   = -y[0]/r;
    y[2*(*n)+1]= -y[1]/r;

}
#ifdef __cplusplus
}
#endif

int main()
{
    double x;
    double *y;
```

```
int n;
int mx;
int *m;
double xf;
double epr;
double epa;
int nst;
int isr;
double *wk;
int ierr;
int i,j,k,nwk;
FILE *fp;

n =2;
mx=2;
nst =0;
isr =0;
fp = fopen( "dksncs.dat", "r" );
if( fp == NULL )
{
    printf( "file open error\n" );
    return -1;
}

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


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

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

nwk=8*n*(mx+1)+1;
wk  = ( double * )malloc((size_t)( sizeof(double) * nwk ));
if( wk  == NULL )
{
    printf( "no enough memory for array wk \n" );
    return -1;
}

m[0]=2;
m[1]=2;
fscanf( fp, "%lf", &x );
printf( "\tx   =%8.3g\n", x   );
for( i=0 ; i<n ; i++ )
{
    for( j=0 ; j<mx ; j++ )
    {
        fscanf( fp, "%lf", &y[i+n*j] );
        printf( "\ty[%6d][%6d]=%8.3g\n",i,j,y[i+n*j] );
    }
}

printf( "\n\tn   =  %6d\n", n );
printf( "\tmx  =  %6d\n", mx);
printf( "\tm[0]=  %6d\n", m[0]);
printf( "\tm[1]=  %6d\n", m[1]);
fscanf( fp, "%lf", &epr );
fscanf( fp, "%lf", &epa );
printf( "\ter  =%8.3g\n", epr );
printf( "\tea  =%8.3g\n", epa );
printf( "\tnst =  %6d\n", nst );
printf( "\tisr =  %6d\n", isr );

fclose( fp );

for (k = 3;k <= 6;k += 3)
{
    xf = (double)k;
    if (k==6) printf( "\n     ** Input **\n\n" );
    printf( "\txf =%8.3g\n", xf   );

    ierr = ASL_dksncs(f, &x, y, n, mx, m, xf, epr, epa, &nst, isr, wk);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tx = %8.3g\n", x );

    printf( "\n\tSolution\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        for( j=0 ; j<(mx+1) ; j++ )
        {
        printf( "\t  y[%6d][%6d]=%8.3g\n",i,j,y[i+n*j] );
        }
```

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

        printf( "\n\tStep Number of Calculation\n\n" );
        printf( "\t  nst  = %6d\n", nst  );
    }

    free( m  );
    free( y  );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dksncs ***

 ** Input **

x   =       0
y[     0][     0]=       1
y[     0][     1]=       0
y[     1][     0]=       0
y[     1][     1]=       1

n   =       2
mx  =       2
m[0]=       2
m[1]=       2
er  =       0
ea  =   1e-10
nst =       0
isr =       0
xf =       3

 ** Output **

ierr =      0

x =        3

Solution

  y[     0][     0]=   -0.99
  y[     0][     1]=  -0.141
  y[     0][     2]=    0.99

  y[     1][     0]=   0.141
  y[     1][     1]=   -0.99
  y[     1][     2]=  -0.141


Step Number of Calculation

  nst  =     56

 ** Input **

xf =       6

 ** Output **

ierr =      0

x =        6

Solution

  y[     0][     0]=    0.96
  y[     0][     1]=   0.279
  y[     0][     2]=   -0.96

  y[     1][     0]=  -0.279
  y[     1][     1]=    0.96
  y[     1][     2]=   0.279


Step Number of Calculation

  nst  =    112
```

2.2.2 ASL_dksnca, ASL_rksnca
連立高階常微分方程式 (精度優先)

(1) **機 能**

自動きざみ幅, 自動次数制御のもとで, 関数評価コストが大きい場合や要求精度が厳しい場合の常微分方程式を解く.

(2) **使用法**

倍精度関数:

ierr = ASL_dksnca (f, & x, y, n, mx, m, xf, er, ea, & nst, isr, iwk, wk);

単精度関数:

ierr = ASL_rksnca (f, & x, y, n, mx, m, xf, er, ea, & nst, isr, 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	f	−	−	入 力	x, y の関数として微分方程式を定義する 関数 f(x, y, n) の関数名
2	x	{D∗ / R∗}	1	入 力	独立変数 x の初期値 x_0
				出 力	独立変数 x の最終到達点 x_e
3	y	{D∗ / R∗}	内容参照	入 力	$x = x_0$ での初期値 $y_i^{(j)}$ $(i = 1, \cdots, \mathrm{n};\ j = 0, \cdots, \mathrm{m}[i-1] - 1)$. $\mathrm{y}[(i-1) + n \times j] = y_i^{(j)}$. **大きさ:** $\mathrm{n} \times (\mathrm{mx} + 1)$
				出 力	$x = x_e$ での計算解 $y_i^{(j)}$ $(i = 1, \cdots, \mathrm{n};\ j = 0, \cdots, \mathrm{m}[i-1])$
4	n	I	1	入 力	方程式の連立数
5	mx	I	1	入 力	最大微分階数 $(\max(\mathrm{m}[i-1]))$
6	m	I*	n	入 力	連立する各方程式の左辺の微分階数 $\mathrm{m}[i-1]$
7	xf	{D / R}	1	入 力	求めたい解の最終点 x_f
8	er	{D / R}	1	入 力	要求局所相対精度 既定値: 倍精度: 10^{-14}, 単精度: 10^{-5}
9	ea	{D / R}	1	入 力	要求局所絶対精度 (既定値 : 表現できる絶対値最小値 $\times 2^{24}$)
10	nst	I*	1	入 力	ステップ回数初期値 (最初に呼ばれるときは 0 入力)
				出 力	計算総ステップ回数 (続けて積分するときの入力値)
11	isr	I	1	入 力	0 : x_f まで積分を行い出力. エラーが発生しなければ $x_e = x_f$. 0 以外 : $x_0 \sim x_f$ の間, 自動きざみ幅進むごとに x_e と $y_i^{(j)}$ を出力する. 最後に $x_e = x_f$ として x_f 点で出力する.
12	iwk	I*	$2 \times \mathrm{n} + 3$	ワーク	作業領域 iwk $[i-1]$ には, i 方程式に使われた次数 q が入る
13	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 wk [0] には, 最後に使われたきざみ幅が入る **大きさ:** $\mathrm{mx} \times (\mathrm{n} + 19) + 20 \times \mathrm{n} + 40$
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) n≥1, nst≥0

(b) $\mathrm{m[i-1]} \geq 1, \mathrm{mx} \geq \max(\mathrm{m[i-1]})$ $(\mathrm{i} = 1, \cdots, \mathrm{n})$

(c) er≥ e_r. ここで e_r= 倍精度: 10^{-14}, 単精度: 10^{-5}
(既定値にするため, 0.0 を入力する場合は除く)

(d) ea≥ 表現できる絶対値最小値 $\times 2^{24}$ (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 (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	計算中きざみ幅が小さくなりすぎた.	その時点の x_e, $y_i^{(j)}$ の値を出力し, 処理を打ち切る.

(6) **注意事項**

(a) x, y の関数として微分方程式を定義する関数 f(x, y, n) の実際の名前は, 使用者側のプログラムで宣言し, 実際の関数を作成しておかなければならない. 連立高階常微分方程式

$$\left\{\begin{array}{rcl} y_1^{(m_1)} & = & f_1(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)}) \\ \vdots & \vdots & \vdots \\ y_i^{(m_i)} & = & f_i(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)}) \\ \vdots & \vdots & \vdots \\ y_n^{(m_n)} & = & f_n(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)}) \end{array}\right.$$

(ただし, 左辺が $y_i^{(m_i)}$ である場合対応する右辺の $y_i^{(j)}$ の微分階数 j は $j \leq m_i - 1$ を満たさなければならない.) を解く場合, この関数 f(x, y, n)(倍精度) の作り方は, 次に示すとおりである.

void FORTRAN f(double ∗x, double ∗y, int ∗n)
{
$\mathrm{y[(*n)*m[0]]} = f_1(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)})$;
⋮
$\mathrm{y[i-1+(*n)*m[i-1]]} = f_i(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)})$;
⋮
$\mathrm{y[(*n)-1+(*n)*m[(*n)-1]]} = f_n(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)})$;
}

ただし, $x \leftrightarrow$ ∗x, $n \leftrightarrow$ ∗n, $y_i \leftrightarrow \mathrm{y[i-1]}$, $y_i^{(j)} \leftrightarrow \mathrm{y[i-1+(*n)*j]}$
と対応する.

例

$$\left\{\begin{array}{ll} 8(y_1'')^2 - 2 - y_2 y_1' = 0 & (y_1'' > 0) \cdots\cdots \text{①} \\ (y_2')^2 - 1 - (y_1')^2 = 0 & (y_2' > 0) \cdots\cdots \text{②} \end{array}\right.$$

①式より $y_1'' = \sqrt{\frac{1}{4} + \frac{1}{8} \cdot y_2 \cdot y_1'}$
②式より $y_2' = \sqrt{1 + (y_1')^2}$
この場合関数はつぎのようになる

void FORTRAN f(double ∗x, double ∗y, int ∗n)
{
y[2 ∗ (∗n)] = sqrt(0.25 + 0.125 ∗ y[1] ∗ y[(∗n)]);
y[1 + (∗n)] = sqrt(1.0 + y[(∗n)] ∗ y[(∗n)]);
}

入力引数は n=2, mx=2, m [0] =2, m [1] =1 となる. また y [0] , y [2] , y [1] に初期値を与える.

(b) 初めて積分するときは nst=0 にすること.

(c) 続けて積分している間は, x, y, nst の出力値をそのまま次の入力値とすること. また作業領域 iwk, wk は, 決して書きかえないこと.

(d) 単精度は ierr=4000 が出やすいので, なるべく倍精度を利用すること.
なお, ierr=4000 のときは, x_e の点から 2.2.4 $\left\{\begin{matrix}\text{ASL_dkssca}\\ \text{ASL_rkssca}\end{matrix}\right\}$を用いるか要求精度を緩くすれば, 続けて解くことができる.

(e) この関数は, 商差法を利用した線形多段階法を利用する.

(7) **使用例**

(a) 問 題

$$\begin{cases} y_1'' = -\frac{y_1}{\gamma} \\ y_2'' = -\frac{y_2}{\gamma} \quad \gamma = (y_1^2 + y_2^2)^{\frac{3}{2}} \end{cases}$$

の連立 2 階常微分方程式を $x = 0.0$ における初期条件

$$y_1(0.0) = 1.0, y_1'(0.0) = 0.0, y_2(0.0) = 0.0, y_2'(0.0) = 1.0$$

のもとで解く.

(b) 入力データ
関数 f(x, y, n) の関数名: f, x=0.0, n=2, y [0] =1.0, y [2] =0.0, y [1] =0.0, y [3] =1.0 mx=2, m [0] =2, m [1] =2, xf, er, ea, nst=0, isr=0

(c) 主プログラム

```
/*      C interface example for ASL_dksnca */

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
void    f(double *x,double *y,int *n)
#else
void    f(x,y,n)
double *x;
double *y;
int    *n;
#endif
{
    double r;

    r= pow(y[0]*y[0]+y[1]*y[1],1.5);
    y[2*(*n)]  = -y[0]/r;
    y[2*(*n)+1]= -y[1]/r;

}
#ifdef __cplusplus
}
#endif

int main()
{
    double x;
    double *y;
```

```
int n;
int mx;
int *m;
double xf;
double epsr;
double epsa;
int istep;
int isr;
int *iwk;
double *wk;
int ierr;
int i,j,k,nwk;
FILE *fp;

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

printf( "    *** ASL_dksnca ***\n" );
printf( "\n     ** Input **\n\n" );
n=2;
mx=2;
istep=0;
isr=0;

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

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


iwk  = ( int * )malloc((size_t)( sizeof(int) * (2*n+3) ));
if( iwk  == NULL )
{
    printf( "no enough memory for array iwk \n" );
    return -1;
}
nwk=mx*(n+19)+20*n+40;
wk  = ( double * )malloc((size_t)( sizeof(double) * nwk ));
if( wk  == NULL )
{
    printf( "no enough memory for array wk \n" );
    return -1;
}

m[0]=2;
m[1]=2;
fscanf( fp, "%lf", &x );
printf( "\tx   =%8.3g\n", x   );
for( i=0 ; i<n ; i++ )
{
    for( j=0 ; j<mx ; j++ )
    {
        fscanf( fp, "%lf", &y[i+n*j] );
        printf( "\ty[%6d][%6d]=%8.3g\n",i,j,y[i+n*j] );
    }
}

printf( "\n\tn   =  %6d\n", n );
printf( "\tmx  =  %6d\n", mx);
printf( "\tm[0]=  %6d\n", m[0]);
printf( "\tm[1]=  %6d\n", m[1]);
fscanf( fp, "%lf", &epsr );
fscanf( fp, "%lf", &epsa );
printf( "\ter  =%8.3g\n", epsr );
printf( "\tea  =%8.3g\n", epsa );
printf( "\tnst =  %6d\n", istep );
printf( "\tisr =  %6d\n", isr );

fclose( fp );

for (k = 3;k <= 6;k += 3)
{
    xf = (double)k;
    if (k==6) printf( "\n     ** Input **\n\n" );
    printf( "\txf =%8.3g\n", xf   );

    ierr = ASL_dksnca(f, &x, y, n, mx, m, xf, epsr, epsa, &istep, isr, iwk, wk);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tx = %8.3g\n", x );

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

```
            for( i=0 ; i<n ; i++ )
            {
                for( j=0 ; j<(mx+1) ; j++ )
                {
                printf( "\t  y[%6d][%6d]=%8.3g\n",i,j,y[i+n*j] );
                }
                printf( "\n" );
            }

            printf( "\tStep Number of Calculation\n\n" );
            printf( "\t  nst  = %6d\n", istep  );
        }

        free( y  );
        free( m  );
        free( iwk );
        free(  wk );

        return 0;
    }
```

(d) 出力結果

```
        *** ASL_dksnca ***

         ** Input **

        x   =       0
        y[     0][     0]=       1
        y[     0][     1]=       0
        y[     1][     0]=       0
        y[     1][     1]=       1

        n   =       2
        mx  =       2
        m[0]=       2
        m[1]=       2
        er  =       0
        ea  =   1e-10
        nst =       0
        isr =       0
        xf =       3

         ** Output **

        ierr =      0

        x =        3

        Solution

          y[     0][     0]=   -0.99
          y[     0][     1]=  -0.141
          y[     0][     2]=    0.99

          y[     1][     0]=   0.141
          y[     1][     1]=   -0.99
          y[     1][     2]=  -0.141

        Step Number of Calculation

          nst  =     35

         ** Input **

        xf =       6

         ** Output **

        ierr =      0

        x =        6

        Solution

          y[     0][     0]=    0.96
          y[     0][     1]=   0.279
          y[     0][     2]=   -0.96

          y[     1][     0]=  -0.279
          y[     1][     1]=    0.96
          y[     1][     2]=   0.279

        Step Number of Calculation

          nst  =     48
```

2.2.3 ASL_dkinct, ASL_rkinct
陰的連立常微分方程式

(1) **機 能**

陰的常微分方程式または代数方程式や非線形方程式と常微分方程式が連立している場合の初期値問題を解く. 一般の常微分方程式や非線形連立方程式も解くことができる.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dkinct  (f, & x, y, n, m, k, xf, idv, & nst, isd, iwk, wk);
```

単精度関数:

```
ierr = ASL_rkinct  (f, & x, y, n, m, k, xf, idv, & nst, isd, 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	f	−	−	入 力	x, y の関数として微分方程式を定義する 関数 f(x, y, n, ji, fn) の関数名
2	x	{D∗ / R∗}	1	入 力	独立変数 x の初期値 x_0
				出 力	独立変数 x の最終到達点 x_e
3	y	{D∗ / R∗}	内容参照	入 力	$x = x_0$ での初期値 $y_i^{(j)}$ $(i = 1, \cdots, \mathrm{n};\ j = 0, \cdots, \mathrm{m}[i-1]-1)$. $\mathrm{y}[(i-1)+\mathrm{n}\times j] = y_i^{(j)}$. **大きさ:** $\mathrm{n}\times(\mathrm{mx}+1)$ ただし, $\mathrm{mx} = \max(\mathrm{m[i-1]}+\mathrm{k[i-1]}+\mathrm{isd[i-1]}-1)$
				出 力	$x = x_e$ での計算解 $y_i^{(j)}$ $(i = 1, \cdots, \mathrm{n};\ j = 0, \cdots, \mathrm{m}[i-1])$
4	n	I	1	入 力	方程式の連立数
5	m	I*	n	入 力	連立する各方程式の微分階数 $\mathrm{m}[i-1]$
6	k	I*	n	入 力	連立する各方程式およびその微分式の部分集合数 $\mathrm{k}[i-1]$
7	xf	{D / R}	1	入 力	解を求めたい所の x_f (エラーがなければ $x_e = x_f$)
8	idv	I	1	入 力	$x_0 \sim x_f$ の区間の積分分割数 (積分きざみ幅は $(\lvert x_f - x_0\rvert/\mathrm{idv}$ になる)
9	nst	I*	1	入 力	ステップ回数初期値 (最初に使用するときは 0 入力)
				出 力	計算総ステップ回数 (続けて積分するときの入力値)
10	isd	I*	n	入 力	自動関数微分スイッチ $\mathrm{isd}[i-1] = 0$: i 番目の方程式群の自動微分は行わない. ($\mathrm{k}[i-1] > 1$ であり, 微分式が関数 f で定義されている.) $\mathrm{isd}[i-1] = 1$: i 番目の方程式群をさらに自動微分し, 精度を上げる. (i 番目の式の微分式作成が困難なときは, 微分式を作らず, $\mathrm{k}[i-1] = 1$, $\mathrm{isd}[i-1] = 1$ とするとよい.)
11	iwk	I*	$12\times\mathrm{n}+1$	ワーク	作業領域
12	wk	{D∗ / R∗}	$24\times\mathrm{n}$	ワーク	作業領域
13	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) n≥1, nst≥0, idv≥1

(b) $0 \leq$ m[i − 1] ≤ 4, k[i − 1] ≥ 1 (i $= 1, \cdots,$ n)

(c) isd[i − 1]=0 or 1, m[i − 1] + k[i − 1] + isd[i − 1] ≤ 6 (i $= 1, \cdots,$ 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), (b) または (c) を満足しなかった.	処理を打ち切る.
4000	非線形連立方程式が解けなかった.	

(6) **注意事項**

(a) x, y の関数として微分方程式を定義する関数 f(x, y, n, ji, fn) の実際の名前は, 使用者側のプログラムで宣言し, 実際の関数を作成しておかなければならない.

陰的連立常微分方程式

$$\begin{cases} f_1(x, \cdots, y_1^{(m_1)}, \cdots) = 0 \\ f_2(x, \cdots, y_2^{(m_2)}, \cdots) = 0 \\ \vdots \\ f_i(x, \cdots, y_i^{(m_i)}, \cdots) = 0 \\ \vdots \\ f_n(x, \cdots, y_n^{(m_n)}, \cdots) = 0 \end{cases}$$

(ここで, $y_i^{(m_i)}$ は y_i の最大微分階数の項で m_i は対応する微分階数) を解く場合, この関数 f(x, y, n, ji, fn)(倍精度) の作り方は, 次に示すとおりである.

```
void FORTRAN f(double *x, double *y, int *n, int *ji, double *fn)
{
 if (*ji==1)
   *fn=f_1(x, ..., y_1^(m_1), ...);
 else if (*ji==2)
   *fn=f_1'(x, ..., y_1^(m_1+1), ...) ;
 else if (*ji==3)
   *fn=f_1''(x, ..., y_1^(m_1+2), ...) ;
 ⋮
 else if (*ji==k_1)
   *fn=f_1^(k_1-1)(x, ..., y_1^(m_1+k_1-1), ...);
 else if (*ji==(k_1 + 1))
   *fn=f_2(x, ..., y_2^(m_2), ...);
 ⋮
 else if (*ji==(k_1 + k_2))
   *fn=f_2^(k_2-1)(x, ..., y_2^(m_2+k_2-1), ...) ;
 ⋮
```

else if (∗ji==($\sum_{i=1}^{n} k_i$))
∗fn=$f_n^{(k_n-1)}(x, \cdots, y_n^{(m_n+k_n-1)}, \cdots)$;
}

ただし, $x \leftrightarrow$ ∗x, $n \leftrightarrow$ ∗n, $y_i \leftrightarrow$ y[i − 1], $y_i^{(j)} \leftrightarrow$ y[i − 1 + (∗n) ∗ j]
と対応する.
なお, 第 i 方程式群の数 $k_i \leftrightarrow$ k[i − 1] は制限条件 (c) を満たすように十分注意して決定し, これに対応して配列 y の大きさも決定すること.

i. i 番目の微分式の作成が困難なときは, 微分式を作らず, k[i − 1] = 1, isd[i − 1] = 1 として自動微分を利用するかまたは微分式を差分式で代用する. $f(y'', y', y, x) = 0$ が与えられた場合, 差分式は

$$\begin{aligned} f'(y'', y', y, x) = \quad & y''' \frac{\partial f(y'', y', y, x)}{\partial y''} + y'' \frac{\partial f(y'', y', y, x)}{\partial y'} \\ & + y' \frac{\partial f(y'', y', y, x)}{\partial y} + \frac{\partial f(y'', y', y, x)}{\partial x} \\ \simeq \quad & y''' \frac{f(y''+h, y', y, x) - f(y''-h, y', y, x)}{2h} \\ & + y'' \frac{f(y'', y'+h, y, x) - f(y'', y'-h, y, x)}{2h} \\ & + y' \frac{f(y'', y', y+h, x) - f(y'', y', y-h, x)}{2h} \\ & + \frac{f(y'', y', y, x+h) - f(y'', y', y, x-h)}{2h} \end{aligned}$$

となる. ここで, $h = \sqrt[3]{\text{誤差判定のための単位}}$ とする.
例えば

$$f(y_1'', y_1', y_1, x) = y_1^2 + x y_1' + \log(\cos(y_1'' x^2)) = 0$$

のとき $g(y_1'', x) = \log(\cos(y_1'' x^2))$ の部分の微分式作成が困難であれば, この部分を差分式で代用し,

$$\begin{aligned} f'(y_1''', y_1'', y_1', y_1, x) = \quad & 2y_1 y_1' + y_1' + x y_1'' \\ & + y_1''' \frac{\log(\cos((y_1''+h)x^2)) - \log(\cos((y_1''-h)x^2))}{2h} \\ & + \frac{\log(\cos(y_1''(x+h)^2)) - \log(\cos(y_1''(x-h)^2))}{2h} \end{aligned}$$

とする. なお, $g(y_1'', x)$ は y_1', y_1 の関数でないことに注意のこと.

ii. 連立する方程式は, y_i の最大微分階数の項 $y_i^{(m_i)} \leftrightarrow$ y[i − 1 + (∗n) ∗ m[i − 1]] を含む方程式が第 i 方程式群の最初の方程式となるように並べる. また複数の式が y_i の最大微分階数の項 $y_i^{(m_i)}$ をもつ場合は, もっとも支配的であるものを含む方程式を第 i 方程式群の最初の方程式となるように選んだ方が良い. なお, 第 i 方程式群の最初の方程式は $y_i^{(m_i)}$ の項を含まなければならない.
例

$$\begin{cases} 3y_1 + 3y_2 + y_3 - 1 = 0 & \cdots\cdots\cdots ① \\ 2y_1' + y_2 + 2y_3 - 6 = 0 & \cdots\cdots\cdots ② \\ y_1 + 2y_2 + 3y_3 - 5 = 0 & \cdots\cdots\cdots ③ \end{cases}$$

のとき②を第 1, ①を第 2, ③を第 3 方程式群になるようならびかえる.

iii. この関数で連立非線形方程式を解くこともできる. このときの関数 f(x, y, n, ji, fn)(倍精度) の作り方は, 次に示すとおりである.

```
void FORTRAN f(double *x, double *y, int *n, int *ji, double *fn)
{
 if (*ji==1)
   *fn=f1(x, y1, ···);
 else if (*ji==2)
   *fn=f2(x, y2, ···);
```

⋮
else if (∗ji==(∗n))
∗fn=$f_n(x, y_n, \cdots)$;
}

ただし, $x \leftrightarrow$ ∗x, $n \leftrightarrow$ ∗n, $y_i \leftrightarrow$ y[i − 1], と対応する.
なお引数は,
n=連立数
m[i − 1] = isd[i − 1] = 0 (i = 1, · · · , n) k[i − 1] = 1 (i = 1, · · · , n) x = xf = 0
y[i − 1 + (∗n) ∗ j] (i = 1, · · · , n, j = 0) は入力不要 (任意)
idv = 1, nst = 0
とする.
計算は $y_i = 0$ ($i = 1, \cdots$, n) を初期値として行われる.
また, 非線形方程式を解くときは, n=1 とすればよい.

関数の作り方の例 (倍精度)

i. 一般の場合

$$\begin{cases} 8(y_1'')^2 - 2 - y_2 y_1' = 0 \\ (y_2')^2 - 1 - (y_1')^2 = 0 \end{cases}$$

void FORTRAN f(double ∗x, double ∗y, int ∗n, int ∗ji, double ∗fn)
{
if (∗ji==1)
∗ fn = 8.0 ∗ y[(∗n) ∗ 2] ∗ y[(∗n) ∗ 2] − 2.0 − y[1] ∗ y[(∗n)];
· · · · · · · · · $(8(y_1'')^2 - 2 - y_2 y_1' = 0)$
else if (∗ji==2)
∗ fn = 16.0 ∗ y[(∗n) ∗ 2] ∗ y[(∗n) ∗ 3] − y[1] ∗ y[(∗n) ∗ 2] − y[1 + (∗n)] ∗ y[(∗n)];
· · · · · · · · · $(16 y_1'' y_1''' - y_2 y_1'' - y_2' y_1' = 0)$
else if (∗ji==3)
∗ fn = y[1 + (∗n)] ∗ y[1 + (∗n)] − 1.0 − y[(∗n)] ∗ y[(∗n)];
· · · · · · · · · $((y_2')^2 - 1 - (y_1')^2 = 0)$
else if (∗ji==4)
∗ fn = 2.0 ∗ y[1 + (∗n)] ∗ y[1 + (∗n) ∗ 2] − 2.0 ∗ y[(∗n)] ∗ y[(∗n) ∗ 2]
· · · · · · · · · $(2 y_2' y_2'' - 2 y_1' y_1'' = 0)$
}

n=2,
m [0] =2, m [1] =1,
k [0] =2, k [1] =2 として
y [0] , y [2] , y [1] に初期値を与える.

ii. 陰的で代数方程式と連立する場合

$$\begin{cases} y_1^{(3)} - y_2'' - 2y_3' - x^2 = 0 \\ y_2'' - y_3' - \dfrac{x^2}{2} = 0 \\ y_3' - \dfrac{x^2}{2} = 0 \\ y_4 - 2y_1 + y_2 - 2y_3 - 1 = 0 \end{cases}$$

void FORTRAN f(double ∗x, double ∗y, int ∗n, int ∗ji, double ∗fn)
{

if (∗ji==1)
∗fn = y[(∗n) ∗ 3)] − y[1 + (∗n) ∗ 2] − 2.0 ∗ y[2 + (∗n)] − (∗x) ∗ (∗x);
$\cdots\cdots\cdots (y_1^{(3)} - y_2'' - 2y_3' - x^2 = 0)$
else if (∗ji==2)
∗fn = y[(∗n) ∗ 4)] − y[1 + (∗n) ∗ 3] − 2.0 ∗ y[2 + (∗n) ∗ 2] − 2.0 ∗ (∗x);
$\cdots\cdots\cdots (y_1^{(4)} - y_2^{(3)} - 2y_3'' - 2x = 0)$
else if (∗ji==3)
∗fn = y[(∗n) ∗ 5)] − y[1 + (∗n) ∗ 4] − 2.0 ∗ y[2 + (∗n) ∗ 3] − 2.0;
$\cdots\cdots\cdots (y_1^{(5)} - y_2^{(4)} - 2y_3^{(3)} - 2 = 0)$
else if (∗ji==4)
∗fn = y[1 + (∗n) ∗ 2)] − y[2 + (∗n)] − (∗x) ∗ (∗x)/2.0;
$\cdots\cdots\cdots (y_2'' - y_3' - \frac{x^2}{2} = 0)$
else if (∗ji==5)
∗fn = y[1 + (∗n) ∗ 3)] − y[2 + (∗n) ∗ 2] − (∗x);
$\cdots\cdots\cdots (y_2^{(3)} - y_3'' - x = 0)$
else if (∗ji==6)
∗fn = y[1 + (∗n) ∗ 4)] − y[2 + (∗n) ∗ 3] − 1.0;
$\cdots\cdots\cdots (y_2^{(4)} - y_3^{(3)} - 1 = 0)$
else if (∗ji==7)
∗fn = y[2 + (∗n)] − (∗x) ∗ (∗x)/2.0;
$\cdots\cdots\cdots (y_3' - \frac{x^2}{2} = 0))$
else if (∗ji==8)
∗fn = y[2 + (∗n) ∗ 2] − (∗x);
$\cdots\cdots\cdots (y_3'' - x = 0)$
else if (∗ji==9)
∗fn = y[2 + (∗n) ∗ 3] − 1.0;
$\cdots\cdots\cdots (y_3^{(3)} - 1 = 0)$
else if (∗ji==10)
∗fn = y[3] − 2.0 ∗ y[0] + y[1] − 2.0 ∗ y[2] − 1.0;
$\cdots\cdots\cdots (y_4 - 2y_1 + y_2 - 2y_3 - 1 = 0)$
}

n=4,
m [0] =3, m [1] =2, m [2] =1, m [3] =0,
k [0] =3, k [1] =3, k [2] =3, k [3] =1,
isd[i − 1] = 0 (i = 1, ⋯ 4) とし, y [0] , y [4] , y [8] , y [1] , y [5] , y [2] に初期値を与える.

iii. 非線形連立方程式の場合

$$\begin{cases} (y_1 - 5)^2 + (y_2 - 2)^2 = 4 \\ (y_1 - 2)^2 + (y_2 - 2)^2 = 4 \end{cases}$$

void FORTRAN f(double ∗x, double ∗y, int ∗n, int ∗ji, double ∗fn)
{
if (∗ji==1)
∗fn = (y[0] − 5.0) ∗ (y[0] − 5.0) + (y[1] − 2.0) ∗ (y[1] − 2.0) − 4.0;
else if (∗ji==2)
∗fn = (y[0] − 2.0) ∗ (y[0] − 2.0) + (y[1] − 2.0) ∗ (y[1] − 2.0) − 4.0;

}

n=2,
m [0] =0, m [1] =0,
k [0] =1, k [1] =1,
isd [0] =0, isd [1] =0, idv=1, x=0, xf=0 とし, y [0] と y [1] は入力不要.
この方程式の解は 2 つの円の交点であり 2 組存在するがそのうち原点 $(y_1, y_2) = (0, 0, 0, 0)$ に近い方の解が返される.

(b) isd[i − 1] = 1 とすれば第 i 方程式群に対し, さらに自動微分が行われるので, 一般に精度が向上するが, やや処理時間が増加する. なお k[i − 1] = 1 のときは, 連立非線形方程式を解く場合を除き必ず isd[i − 1] = 1 とすること.

(c) 方程式群をできるだけ多く作り, 集合数 k[i − 1] を大きくするか, 積分分割数, idv を大きくすると一般に精度が上がるが処理時間が増大する.

(d) 初めて積分するときは nst=0 にすること.

(e) 続けて積分したい場合は, x, y, nst の出力値をそのまま次の入力値とすること. また作業領域 iwk, wk は決して書きかえないこと.

(f) 単精度関数では ierr=4000 が出やすいので, なるべく倍精度関数を利用すること. なお, ierr= 4000 のときは, idv を大きくすることで解ける場合がある.

(7) 使用例

(a) 問 題

$$\begin{cases} y_1^{(3)} - y_2'' - 2y_3' - x^2 = 0 \\ y_2'' - y_3' - \dfrac{x^2}{2} = 0 \\ y_3' - \dfrac{x^2}{2} = 0 \\ y_4 - 2y_1 + y_2 - 2y_3 - 1 = 0 \end{cases}$$

の連立 3 階常微分–代数連立方程式を x=1.0 における初期条件
$y_1 = 0.05$, $y_1' = 0.25$, $y_1'' = 1.0$
$y_2 = 0.08333$, $y_2' = 0.333$
$y_3 = 0.1666$ のもとで解く.

(b) 入力データ
関数 f(x, y, n, ji, fn) の関数名: f, x=0.0, n=4,
y [0] =0.05, y [4] =0.25, y [8] =1.0, y [1] =0.08333, y [5] =0.333, y [2] =0.1666
m [0] =3, m [1] =2, m [2] =1, m [3] =0,
k [0] =2, k [1] =2, k [2] =2, k [3] =1,
xf=1.5, idv=10, nst=0 (最初のみ),
isd [0] =1, isd [1] =1, isd [2] =1, isd [3] =0

(c) 主プログラム

```
/*      C interface example for ASL_dkinct */

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
void    f(double *x,double *y,int *n,int *ji,double *fn)
```

```
#else
void    f(x,y,n,ji,fn)
double *x;
double *y;
int    *n;
int    *ji;
double *fn;
#endif
{
    if( *ji == 1 )
    {
        *fn=y[3*(*n)]-y[2*(*n)+1]-2.0*y[(*n)+2]-((*x)*(*x));
    }
    else if( *ji == 2 )
    {
        *fn=y[4*(*n)]-y[3*(*n)+1]-2.0*y[2*(*n)+2]-((*x)+(*x));
    }
    else if( *ji == 3 )
    {
        *fn=y[2*(*n)+1]-y[(*n)+2]-0.5*((*x)*(*x));
    }
    else if( *ji == 4 )
    {
        *fn=y[3*(*n)+1]-y[2*(*n)+2]-(*x);
    }
    else if( *ji == 5 )
    {
        *fn=y[(*n)+2]-0.5*((*x)*(*x));
    }
    else if( *ji == 6 )
    {
        *fn=y[2*(*n)+2]-(*x);
    }
    else if( *ji == 7 )
    {
        *fn=y[3]-2.0*y[0]+y[1]-2.0*y[2]-1.0;
    }
    else
    {
        printf("error from fkinct ji=%6d\n",*ji);
    }
}
#ifdef __cplusplus
}
#endif

int main()
{
    double x;
    double *y;
    int n;
    int *m;
    int *k;
    double xf;
    int idv;
    int nst;
    int *isd;
    int *iwk;
    double *wk;
    int ierr;
    int i,j,mx;
    FILE *fp;

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

    printf( "    *** ASL_dkinct ***\n" );
    printf( "\n     ** Input **\n\n" );
    n=4;
    nst=0;

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

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

    isd  = ( int * )malloc((size_t)( sizeof(int) * n ));
    if( isd  == NULL )
    {
        printf( "no enough memory for array  isd\n" );
        return -1;
    }
    m[0]=3;
    m[1]=2;
```

```
m[2]=1;
m[3]=0;
k[0]=2;
k[1]=2;
k[2]=2;
k[3]=1;
isd[0]=1;
isd[1]=1;
isd[2]=1;
isd[3]=0;
mx=0;
for( i=0 ; i<n; i++  )
{
    mx=( (mx>m[i]+k[i]+isd[i]-1) ? mx : m[i]+k[i]+isd[i]-1 );
}

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


iwk  = ( int * )malloc((size_t)( sizeof(int) * (12*n+1) ));
if( iwk  == NULL )
{
    printf( "no enough memory for array iwk \n" );
    return -1;
}
wk  = ( double * )malloc((size_t)( sizeof(double) * (24*n) ));
if( wk  == NULL )
{
    printf( "no enough memory for array wk \n" );
    return -1;
}

fscanf( fp, "%lf", &x );
printf( "\tx   =%8.3g\n", x   );
for( i=0 ; i<n; i++  )
{
    for( j=0 ; j<mx ; j++ )
    {
        y[i+n*j]=0.0;
    }
}
for( i=0 ; i<n;  )
{
    for( j=0 ; j<m[i] ; j++ )
    {
        fscanf( fp, "%lf", &y[i+n*j] );
        printf( "\ty[%6d][%6d]=%8.3g\n",i,j,y[i+n*j] );
    }
    i++;
}
fscanf( fp, "%lf", &xf);
fscanf( fp, "%d", &idv);

printf( "\n" );
printf( "\tn     =  %6d\n", n );
printf( "\tm[0]  =  %6d\n", m[0]);
printf( "\tm[1]  =  %6d\n", m[1]);
printf( "\tm[2]  =  %6d\n", m[2]);
printf( "\tm[3]  =  %6d\n", m[3]);
printf( "\tk[0]  =  %6d\n", k[0]);
printf( "\tk[1]  =  %6d\n", k[1]);
printf( "\tk[2]  =  %6d\n", k[2]);
printf( "\tk[3]  =  %6d\n", k[3]);
printf( "\txf    =%8.3g\n", xf );
printf( "\tidv   =  %6d\n", idv );
printf( "\tnst   =  %6d\n", nst );
printf( "\tisd[0]=  %6d\n", isd[0]);
printf( "\tisd[1]=  %6d\n", isd[1]);
printf( "\tisd[2]=  %6d\n", isd[2]);
printf( "\tisd[3]=  %6d\n", isd[3]);
printf( "\n" );

fclose( fp );

ierr = ASL_dkinct(f, &x, y, n, m, k, xf, idv, &nst, isd, iwk, wk);

printf( "\n     ** Output **\n\n" );
printf( "\tierr = %6d\n", ierr );
printf( "\n\tx  = %8.3g\n", x );

printf( "\n\tSolution\n\n" );
for( i=0 ; i<n ; i++ )
{
    for( j=0 ; j<m[i]+1 ; j++ )
    {
        printf( "\ty[%6d][%6d]=%8.3g\n",i,j,y[i+n*j] );
    }
    printf( "\n" );
}
```

```
    printf( "\n\tStep Number of Calculation\n\n" );
    printf( "\t  nst  = %6d\n", nst  );
    printf( "\n" );

    free( m  );
    free( k  );
    free( isd );
    free( y  );
    free( wk );
    free( iwk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dkinct ***

 ** Input **

x   =       1
y[     0][     0]=    0.05
y[     0][     1]=    0.25
y[     0][     2]=       1
y[     1][     0]=  0.0833
y[     1][     1]=   0.333
y[     2][     0]=   0.167

n     =       4
m[0]  =       3
m[1]  =       2
m[2]  =       1
m[3]  =       0
k[0]  =       2
k[1]  =       2
k[2]  =       2
k[3]  =       1
xf    =     1.5
idv   =      10
nst   =       0
isd[0]=       1
isd[1]=       1
isd[2]=       1
isd[3]=       0


 ** Output **

ierr =      0

x  =      1.5

Solution

y[     0][     0]=    0.38
y[     0][     1]=    1.27
y[     0][     2]=    3.37
y[     0][     3]=    6.75

y[     1][     0]=   0.422
y[     1][     1]=    1.12
y[     1][     2]=    2.25

y[     2][     0]=   0.562
y[     2][     1]=    1.13

y[     3][     0]=    2.46


Step Number of Calculation

  nst  =     10
```

2.2.4 ASL_dkssca, ASL_rkssca
スティフ問題の連立高階常微分方程式

(1) **機 能**

自動刻み幅, 自動次数制御のもとで, スティフな常微分方程式の初期値問題を解く.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dkssca  (f, & x, y, n, mx, m, xf, er, ea, & nst, isr, iwk, wk);
```

単精度関数:

```
ierr = ASL_rkssca  (f, & x, y, n, mx, m, xf, er, ea, & nst, isr, 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	f	−	−	入 力	x, y の関数として微分方程式を定義する 関数 f(x, y, n) の関数名
2	x	{D∗ / R∗}	1	入 力	独立変数 x の初期値 x_0
				出 力	独立変数 x の最終到達点 x_e
3	y	{D∗ / R∗}	内容参照	入 力	$x = x_0$ での初期値 $y_i^{(j)}$ $(i = 1, \cdots, \mathrm{n};\ j = 0, \cdots, \mathrm{m}[i-1] - 1)$. $\mathrm{y}[(i-1) + n \times j] = y_i^{(j)}$. **大きさ:** $\mathrm{n}\times(\mathrm{mx}+1)$
				出 力	$x = x_e$ での計算解 $y_i^{(j)}$ $(i = 1, \cdots, \mathrm{n};\ j = 0, \cdots, \mathrm{m}[i-1])$
4	n	I	1	入 力	方程式の連立数
5	mx	I	1	入 力	最大微分階数 $(\max(\mathrm{m[i-1]}))$
6	m	I*	n	入 力	連立する各方程式の左辺の微分階数 $\mathrm{m[i-1]}$
7	xf	{D / R}	1	入 力	求めたい解の最終点 x_f
8	er	{D / R}	1	入 力	要求局所相対精度 既定値: 倍精度: 10^{-14}, 単精度: 10^{-5}
9	ea	{D / R}	1	入 力	要求局所絶対精度 (既定値 : 表現できる絶対値最小値 $\times 2^{24}$)
10	nst	I*	1	入 力	ステップ回数初期値 (最初に呼ばれたときは 0 入力)
				出 力	計算総ステップ回数 (続けて積分するときの入力値)
11	isr	I	1	入 力	利用者プログラムに戻るタイミングを指定するパラメータ (注意事項 (d) 参照)
12	iwk	I*	内容参照	ワーク	作業領域 **大きさ**: $\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{m[i-1]} + 6$
13	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 wk [0] には最後に使われたきざみ幅が入る **大きさ**: $28 + (\mathrm{k}+9) \times \mathrm{k} + 2 \times \mathrm{n} \times (\mathrm{mx}+1)$ ここで, $\mathrm{k} = \sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{m[i-1]}$
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) $\mathrm{n} > 0, \mathrm{nst} \geq 0$

(b) $\mathrm{mx} \geq \mathrm{m[i-1]} \geq 1$ $(i = 1, \cdots, \mathrm{n})$

(c) isr={0, 1, 2}

(d) er≥ e_r. ここで e_r= 倍精度: 10^{-14}, 単精度: 10^{-5} (既定値にするため, 0.0 を入力する場合は除く)

(e) ea≥ 表現できる絶対値最小値 $\times 2^{24}$ (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 (d) または (e) 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), (b) または (c) を満足しなかった.	処理を打ち切る.
4000	計算中きざみ幅が小さくなりすぎた.	その時点の x_e, $y_i^{(j)}$ の値を出力し, 処理を打ち切る.
4100	修正子を求める方程式が解けない.	その時点の x_e, $y_i^{(j)}$ の値を出力し, 処理を打ち切る.

(6) **注意事項**

(a) x, y の関数として微分方程式を定義する関数 f(x, y, n) の実際の名前は, 使用者側のプログラムで宣言し, 実際の関数を作成しておかなければならない. 連立高階常微分方程式

$$\left\{\begin{array}{rcl} y_1^{(m_1)} & = & f_1(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)}) \\ \vdots & \vdots & \vdots \\ y_i^{(m_i)} & = & f_i(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)}) \\ \vdots & \vdots & \vdots \\ y_n^{(m_n)} & = & f_n(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)}) \end{array}\right.$$

(ただし, 左辺が $y_i^{(m_i)}$ である場合対応する右辺の $y_i^{(j)}$ の微分階数 j は $j \leq m_i - 1$ を満たさなければならない.) を解く場合, この関数 f(x, y, n)(倍精度) の作り方は, 次に示すとおりである.

void FORTRAN f(double ∗x, double ∗y, int ∗n)
{
$\mathrm{y[(*n) * m[0]]} = f_1(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)})$;
⋮
$\mathrm{y[i - 1 + (*n) * m[i - 1]]} = f_i(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)})$;
⋮
$\mathrm{y[(*n) - 1 + (*n) * m[(*n) - 1]]} = f_n(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)})$;
}

ただし, $x \leftrightarrow$ ∗x, $n \leftrightarrow$ ∗n, $y_i \leftrightarrow \mathrm{y[i-1]}$, $y_i^{(j)} \leftrightarrow \mathrm{y[i - 1 + (*n) * j]}$
と対応する.

例

$$\left\{\begin{array}{l} 8(y_1'')^2 - 2 - y_2 y_1' = 0 \quad (y_1'' > 0) \cdots\cdots \text{①} \\ (y_2')^2 - 1 - (y_1')^2 = 0 \quad (y_{2'} > 0) \cdots\cdots \text{②} \end{array}\right.$$

①式より $y_1'' = \sqrt{\frac{1}{4} + \frac{1}{8} \cdot y_2 \cdot y_1'}$
②式より $y_2' = \sqrt{1 + (y_1')^2}$
この場合関数はつぎのようになる.

```
void FORTRAN f(double *x, double *y, int *n)
{
    y[2 * (*n)] = sqrt(0.25 + 0.125 * y[1] * y[(*n)]);
    y[1 + (*n)] = sqrt(1.0 + y[(*n)] * y[(*n)]);
}
```

入力引数は n=2, mx=2, m [0] =2, m [1] =1 となる. また y [0] , y [2] , y [1] に初期値を与える.

(b) er と ea は与えられた値が既定値よりも小さいときは既定値に設定して計算する.

(c) ler を局所相対誤差を要素とするベクトル, lea を局所絶対誤差を要素とするベクトルとするとき $\| \text{ler} \| \leq \text{er}$ または $\| \text{lea} \| \leq \text{ea}$ のどちらかが満たされれば解を受け入れる. $\| \ \|$ は最大値ノルムである. 相対誤差は過去の最大の値に対する誤差として計算される.

(d) isr の値は以下のように指定する.
isr=0　xf を越えて積分した後, xf に内挿する.
isr=1　内挿を行わず, 正確に xf に行き着くまで計算する.
isr=2　xf の方向に 1 つの積分区間だけを計算する.
xf を越えた点で微係数が定義されないとき, または xf を越えた近くに不連続点をもつときは isr=1 を指定する.

(e) 続けて積分している間は, x, y の出力値はそのまま次の入力値とすること. また作業領域 iwk, wk は, 決して書き換えないこと.

(f) ierr=4000 のときは, 要求精度を緩くすれば続けて解くことができる.

(7) 使用例

(a) 問 題

$$\begin{cases} y_1'' = -\frac{y_1}{\gamma} \\ y_2'' = -\frac{y_2}{\gamma} \quad \gamma = (y_1^2 + y_2^2)^{\frac{3}{2}} \end{cases}$$

の連立 2 階常微分方程式を $x = 0.0$ における初期条件

$$y_1(0.0) = 1.0, y_1'(0.0) = 0.0, y_2(0.0) = 0.0, y_2'(0.0) = 1.0$$

のもとで解く.

(b) 入力データ
関数 f(x, y, n) の関数名: f, x=0.0, n=2, y [0] =1.0, y [2] =0.0, y [1] =0.0, y [3] =1.0, mx=2, m [0] =2, m [1] =2, xf, er, ea, nst=0, isr=0

(c) 主プログラム

```
/*      C interface example for ASL_dkssca */

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
void    f(double *x,double *y,int *n)
#else
void    f(x,y,n)
double *x;
```

```
double *y;
int    *n;
#endif
{
    double r;

    r= pow(y[0]*y[0]+y[1]*y[1],1.5);
    y[2*(*n)]  = -y[0]/r;
    y[2*(*n)+1]= -y[1]/r;

}
#ifdef __cplusplus
}
#endif

int main()
{
    double x;
    double *y;
    int n;
    int mx;
    int *m;
    double xf;
    double er;
    double ea;
    int nst;
    int isr;
    int *iwk;
    double *wk;
    int ierr;
    int i,j,k,summ,nwk;
    FILE *fp;

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

    printf( "    *** ASL_dkssca ***\n" );
    printf( "\n     ** Input **\n\n" );
    n=2;
    mx=2;
    nst=0;

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

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

    m[0]=2;
    m[1]=2;
    summ=0;
    for( i=0 ; i<n ; i++ )
        summ+=m[i];

    iwk  = ( int * )malloc((size_t)( sizeof(int) * (summ+6) ));
    if( iwk  == NULL )
    {
        printf( "no enough memory for array iwk \n" );
        return -1;
    }
    nwk=28+(summ+9)*summ+2*n*(mx+1);
    wk  = ( double * )malloc((size_t)( sizeof(double) * nwk ));
    if( wk  == NULL )
    {
        printf( "no enough memory for array wk \n" );
        return -1;
    }

    fscanf( fp, "%d", &isr );
    fscanf( fp, "%lf", &x );
    printf( "\tx =%8.3g\n", x   );
    for( i=0 ; i<n ; i++ )
    {
        for( j=0 ; j<mx ; j++ )
        {
            fscanf( fp, "%lf", &y[i+n*j] );
            printf( "\ty[%6d][%6d]=%8.3g\n",i,j,y[i+n*j] );
        }
    }

    printf( "\n\tn   =  %6d\n", n );
    printf( "\tmx  =  %6d\n", mx);
    printf( "\tm[0]=  %6d\n", m[0]);
    printf( "\tm[1]=  %6d\n", m[1]);
    fscanf( fp, "%lf", &er );
```

```
    fscanf( fp, "%lf", &ea );
    printf( "\ter  =%8.3g\n", er );
    printf( "\tea  =%8.3g\n", ea );
    printf( "\tnst =  %6d\n", nst );
    printf( "\tisr =  %6d\n", isr );

    fclose( fp );

    for (k = 3;k <= 6;k += 3)
    {
        xf = (double)k;
        if (k==6) printf( "\n     ** Input **\n\n" );
        printf( "\txf  =%8.3g\n", xf   );

        ierr = ASL_dkssca(f, &x, y, n, mx, m, xf, er, ea, &nst, isr, iwk, wk);

        printf( "\n     ** Output **\n\n" );
        printf( "\tierr = %6d\n", ierr );
        printf( "\n\tx = %8.3g\n", x );

        printf( "\n\tSolution\n\n" );
        for( i=0 ; i<n ; i++ )
        {
            for( j=0 ; j<(mx+1) ; j++ )
            {
            printf( "\t  y[%6d][%6d]=%8.3g\n",i,j,y[i+n*j] );
            }
            printf( "\n" );
        }

        printf( "\n\tStep Number of Calculation\n\n" );
        printf( "\t  nst  = %6d\n", nst  );
        printf( "\n" );
    }

    free( m  );
    free( y  );
    free( wk );
    free( iwk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dkssca ***

 ** Input **

x =       0
y[     0][     0]=       1
y[     0][     1]=       0
y[     1][     0]=       0
y[     1][     1]=       1

n   =       2
mx  =       2
m[0]=       2
m[1]=       2
er  =       0
ea  =       0
nst =       0
isr =       0
xf  =       3

 ** Output **

ierr =      0

x =        3

Solution

  y[     0][     0]=   -0.99
  y[     0][     1]=  -0.141
  y[     0][     2]=    0.99

  y[     1][     0]=   0.141
  y[     1][     1]=   -0.99
  y[     1][     2]=  -0.141


Step Number of Calculation

  nst  =    551


 ** Input **

xf  =       6

 ** Output **

ierr =      0

x =        6
```

```
Solution

  y[     0][     0]=    0.96
  y[     0][     1]=   0.279
  y[     0][     2]=   -0.96

  y[     1][     0]=  -0.279
  y[     1][     1]=    0.96
  y[     1][     2]=   0.279

Step Number of Calculation

  nst  =   1053
```

2.2.5 ASL_dkfncs, ASL_rkfncs
連立 1 階常微分方程式

(1) **機 能**

自動きざみ幅制御のもとで, 要求局所精度を満足する連立 1 階の常微分方程式初期値問題を解く.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dkfncs  (f, & x, y, n, xf, er, ea, & nst, wk);
```

単精度関数:

```
ierr = ASL_rkfncs  (f, & x, y, n, xf, er, ea, & nst, 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	f	−	−	入 力	x, y の関数として微分方程式を定義する 関数 f(x, y, n) の関数名
2	x	{D∗ / R∗}	1	入 力	独立変数 x の初期値 x_0
				出 力	独立変数 x の最終到達点 x_e
3	y	{D∗ / R∗}	2×n	入 力	$x = x_0$ での初期値 $y_i (i = 1, \cdots, \mathrm{n})$ y[$i-1$] $= y_i$.
				出 力	$x = x_e$ での計算解 $y_i^{(j)} (i = 1, \cdots, \mathrm{n}; j = 0, 1)$
4	n	I	1	入 力	方程式の連立数
5	xf	{D / R}	1	入 力	求めたい解の最終点 x_f
6	er	{D / R}	1	入 力	要求局所相対精度 既定値: 倍精度: 10^{-12}, 単精度: 10^{-5}
7	ea	{D / R}	1	入 力	要求局所絶対精度 (既定値 : 表現できる絶対値最小値 $\times 2^{24}$)
8	nst	I*	1	入 力	ステップ回数初期値 (最初に呼ばれるときは 0 入力)
				出 力	計算総ステップ回数 (続けて積分するときの入力値)
9	wk	{D∗ / R∗}	16 × n + 1	ワーク	作業領域 wk [0] には最後に使われたきざみ幅が入る
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 ≥ 1, nst ≥ 0

(b) er$\geq e_r$. ここで e_r= 倍精度: 10^{-14}, 単精度: 10^{-5} (既定値にするため, 0.0 を入力する場合は除く)

(c) ea$\geq$ 表現できる絶対値最小値 $\times 2^{24}$ (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 (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	計算中きざみ幅が小さくなりすぎた.	その時点の x_e, $y_i^{(j)}$ の値を出力し, 処理を打ち切る.

(6) **注意事項**

(a) x, y の関数として微分方程式を定義する関数 f(x, y, n) の実際の名前は, 使用者側のプログラムで宣言し, 実際の関数を作成しておかなければならない. 連立常微分方程式

$$\left\{\begin{array}{lcl} y_1' &=& f_1(x, y_1, \cdots, y_i, \cdots, y_n) \\ \vdots & \vdots & \vdots \\ y_i' &=& f_i(x, y_1, \cdots, y_i, \cdots, y_n) \\ \vdots & \vdots & \vdots \\ y_n' &=& f_n(x, y_1, \cdots, y_i, \cdots, y_n) \end{array}\right.$$

を解く場合, この関数 f(x, y, n)(倍精度) の作り方は, 次に示すとおりである.

```
void FORTRAN f(double *x, double *y, int *n)
{
  y[(*n)] = f1(x, y1, ···, yi, ···, yn);
  ⋮
  y[i − 1 + (*n)] = fi(x, y1, ···, yi, ···, yn);
  ⋮
  y[2 * (*n) − 1] = fn(x, y1, ···, yi, ···, yn);
}
```

ただし, $x \leftrightarrow$ *x, $n \leftrightarrow$ *n, $y_i \leftrightarrow$ y[i − 1], と対応する.

例

$$\left\{\begin{array}{ll} y_1' = y_2 & \cdots\cdots ① \\ y_2' = \dfrac{4y_1}{x^2} + \dfrac{2y_2}{x} & \cdots\cdots ② \end{array}\right.$$

この場合, 連立数 n=2 であり, 作成すべき関数は次のようになる.

```
void FORTRAN f(double *x, double *y, int *n)
{
   y[(*n)] = y[1];
   y[1 + (*n)] = 4.0 * y[0]/((*x) * (*x)) + 2.0 * y[1]/(*x);
}
```

(b) 初めて積分するときは nst=0 にすること.

(c) 続けて積分している間は, x, y, nst の出力値をそのまま次の入力値とし, xf を次々と与えながら解くとよい.

(d) ierr=4000 のときは, x_e の点から 2.2.4 $\left\{\begin{array}{l}\text{ASL\_dkssca}\\ \text{ASL\_rkssca}\end{array}\right\}$ を用いるか, 要求精度を緩くすれば続けて解くことができる.

(e) この関数は, ルンゲ・クッタ・バーナー法を利用している.

(7) 使用例

(a) 問 題

$$\begin{cases} y_1' = y_3 \\ y_2' = y_4 \\ y_3' = -\frac{y_1}{\gamma} \\ y_4' = -\frac{y_2}{\gamma} \quad \gamma = (y_1^2 + y_2^2)^{\frac{3}{2}} \end{cases}$$

の連立 1 階常微分方程式を $x = 0.0$ における初期条件

$$y_1 = 1.0, y_2 = 0.0, y_3 = 0.0, y_4 = 1.0$$

のもとで $x = 3.0$ と $x = 6.0$ について解く.

(b) 入力データ

関数 f(x, y, n) の関数名: f, x=0.0, n=2, y [0] =1.0, y [1] =0.0, y [2] =0.0, y [3] =1.0 xf, er, ea, nst=0(最初のみ)

(c) 主プログラム

```
/*      C interface example for ASL_dkfncs */

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
void    f(double *x,double *y,int *n)
#else
void    f(x,y,n)
double *x;
double *y;
int    *n;
#endif
{
    double r;

    r= pow(y[0]*y[0]+y[1]*y[1],1.5);
    y[(*n)]  =  y[2];
    y[(*n)+1]=  y[3];
    y[(*n)+2]= -y[0]/r;
    y[(*n)+3]= -y[1]/r;

}
#ifdef __cplusplus
}
#endif

int main()
{
    double x;
    double *y;
    int n;
    double xf;
    double epr;
    double epa;
    int nst;
    double *wk;
    int ierr;
    int i,j,k;
    FILE *fp;

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

    printf( "    *** ASL_dkfncs ***\n" );
    printf( "\n     ** Input **\n\n" );
    n=4;
    nst=0;

    y  = ( double * )malloc((size_t)( sizeof(double) * (2*n) ));
    if( y  == NULL )
```

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

    wk  = ( double * )malloc((size_t)( sizeof(double) * (16*n+1) ));
    if( wk  == NULL )
    {
        printf( "no enough memory for array wk \n" );
        return -1;
    }

    fscanf( fp, "%lf", &x );
    printf( "\tx =%8.3g\n", x   );
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &y[i] );
        printf( "\ty[%6d][     0]=%8.3g\n",i,y[i] );
    }

    printf( "\n\tn   =  %6d\n", n );
    fscanf( fp, "%lf", &epr );
    fscanf( fp, "%lf", &epa );
    printf( "\ter  =%8.3g\n", epr );
    printf( "\tea  =%8.3g\n", epa );
    printf( "\tnst =  %6d\n", nst );

    fclose( fp );

    for (k = 3;k <= 6;k += 3)
    {
        xf = (double)k;
        if (k==6) printf( "\n     ** Input **\n\n" );
        printf( "\txf  =%8.3g\n", xf   );

        ierr = ASL_dkfncs(f, &x, y, n, xf, epr, epa, &nst, wk);

        printf( "\n     ** Output **\n\n" );
        printf( "\tierr = %6d\n", ierr );
        printf( "\n\tx = %8.3g\n", x );

        printf( "\n\tSolution\n\n" );
        for( i=0 ; i<n ; i++ )
        {
            for( j=0 ; j<2 ; j++ )
            {
                printf( "\t  y[%6d][%6d]=%8.3g\n",i,j,y[i+n*j] );
            }
            printf( "\n" );
        }

        printf( "\n\tStep Number of Calculation\n\n" );
        printf( "\t  nst  = %6d\n", nst  );
    }

    free( y  );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dkfncs ***

 ** Input **

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

n   =       4
er  =       0
ea  =   1e-10
nst =       0
xf  =       3

 ** Output **

ierr =      0

x =        3

Solution

  y[     0][     0]=   -0.99
  y[     0][     1]=  -0.141

  y[     1][     0]=   0.141
  y[     1][     1]=   -0.99

  y[     2][     0]=  -0.141
  y[     2][     1]=    0.99
```

```
  y[     3][     0]=   -0.99
  y[     3][     1]=  -0.141

Step Number of Calculation

  nst  =     56

 ** Input **

xf  =       6

 ** Output **

ierr =      0

x =        6

Solution

  y[     0][     0]=    0.96
  y[     0][     1]=   0.279

  y[     1][     0]=  -0.279
  y[     1][     1]=    0.96

  y[     2][     0]=   0.279
  y[     2][     1]=   -0.96

  y[     3][     0]=    0.96
  y[     3][     1]=   0.279

Step Number of Calculation

  nst  =    112
```

2.2.6 ASL_dkhncs, ASL_rkhncs
高階常微分方程式

(1) **機 能**

自動きざみ幅制御のもとで, 要求局所精度を満足する単独の高階常微分方程式初期値問題を解く.

(2) **使用法**

倍精度関数:

ierr = ASL_dkhncs (f, & x, y, m, xf, er, ea, & nst, wk);

単精度関数:

ierr = ASL_rkhncs (f, & x, y, m, xf, er, ea, & nst, 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	f	−	−	入 力	x, y の関数として微分方程式を定義する 関数 f(x, y) の関数名
2	x	{D∗ / R∗}	1	入 力	独立変数 x の初期値 x_0
				出 力	独立変数 x の最終到達点 x_e
3	y	{D∗ / R∗}	m + 1	入 力	$x = x_0$ での初期値 $y^{(j)}(j = 0, \cdots, \mathrm{m}-1)$ $\mathrm{y}[j] = y^{(j)}$.
				出 力	$x = x_e$ での計算解 $y^{(j)}(j = 0, \cdots, \mathrm{m})$
4	m	I	1	入 力	方程式左辺の微分階数 (最大微分階数)
5	xf	{D / R}	1	入 力	求めたい解の最終点 x_f
6	er	{D / R}	1	入 力	要求局所相対精度 既定値: 倍精度: 10^{-12}, 単精度: 10^{-5}
7	ea	{D / R}	1	入 力	要求局所絶対精度 (既定値 : 表現できる絶対値最小値 $\times 2^{24}$)
8	nst	I*	1	入 力	ステップ回数初期値 (最初に使用するときは 0 入力)
				出 力	計算総ステップ回数 (続けて積分するときの入力値)
9	wk	{D∗ / R∗}	8 × m + 9	ワーク	作業領域 wk [0] には最後に使われたきざみ幅が入る
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) m ≥ 1, nst ≥ 0

(b) er≥ e_r. ここで e_r= 倍精度: 10^{-14}, 単精度: 10^{-5} (既定値にするため, 0.0 を入力する場合は除く)

(c) ea≥ 表現できる絶対値最小値 $\times 2^{24}$ (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 (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	計算中きざみ幅が小さくなりすぎた.	その時点の x_e, $y_i^{(j)}$ の値を出力し, 処理を打ち切る.

(6) **注意事項**

(a) x, y の関数として微分方程式を定義する関数 f(x, y) の実際の名前は, 使用者側のプログラムで宣言し, 実際の関数を作成しておかなければならない. 高階常微分方程式

$$y^{(m)} = f(x, y, \cdots, y^{(j)}, \cdots, y^{(m-1)})$$

(ただし, 左辺が $y^{(m)}$ である場合右辺の $y^{(j)}$ の微分階数 j は $j \leq m-1$ を満たさなければならない. また, m は定数である.) を解く場合, この関数 f(x, y)(倍精度) の作り方は, 次に示すとおりである.

```
void FORTRAN f(double *x, double *y)
{
  y[m] = f(x, y, ···, y^(j), ···, y^(m-1));
}
```

ただし, $x \leftrightarrow$ *x, $y \leftrightarrow$ y[0], $y^{(j)} \leftrightarrow$ y[j]
と対応する.

例

$$y'' = \frac{4y}{x^2} + \frac{2y'}{x}$$

この場合関数はつぎのようになる

```
void FORTRAN f(double *x, double *y)
{
   y[2] = 4.0 * y[0]/((*x) * (*x)) + 2.0 * y[1]/(*x);
}
```

入力引数は m=2 となる.

(b) 初めて積分するときは nst=0 にすること.

(c) 続けて積分している間は, x, y, nst の出力値をそのまま次の入力値とし, xf を次々と与えながら解くとよい.

(d) ierr=4000 のときは, x_e の点から 2.2.4 $\left\{\begin{matrix}\text{ASL\_dkssca}\\ \text{ASL\_rkssca}\end{matrix}\right\}$ を用いるか, 要求精度を緩くすれば続けて解くことができる.

(e) この関数は, ルンゲ・クッタ・バーナー法を利用している.

(7) **使用例**

(a) 問 題

$$y'' = \frac{4y}{x^2} + \frac{2y'}{x}$$

を x=1.0 における初期条件
y=5.0, $y' = 3.0$ として x=5.0 と 10.0 に対して解く.

(b) 入力データ

関数 f(x, y) の関数名: f, x=1.0, y [0] =5.0, y [1] =3.0, m=2, xf, er, ea, nst=0(最初のみ)

(c) 主プログラム

```
/*      C interface example for ASL_dkhncs */

#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[2]= 4.0*y[0]/((*x)*(*x))+2.0*y[1]/(*x);
}
#ifdef __cplusplus
}
#endif

int main()
{
    double x;
    double *y;
    int m;
    double xf;
    double epr;
    double epa;
    int nst;
    double *wk;
    int ierr;
    int i,k;
    FILE *fp;

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

    printf( "    *** ASL_dkhncs ***\n" );
    printf( "\n     ** Input **\n\n" );
    m=2;
    nst=0;

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

    wk  = ( double * )malloc((size_t)( sizeof(double) * (8*m+9) ));
    if( wk  == NULL )
    {
        printf( "no enough memory for array wk \n" );
        return -1;
    }

    fscanf( fp, "%lf", &x );
    printf( "\tx   =%8.3g\n", x   );
    for( i=0 ; i<m ; i++ )
    {
        fscanf( fp, "%lf", &y[i] );
        printf( "\ty[%6d]=%8.3g\n",i,y[i] );
    }

    printf( "\n\tm   =  %6d\n", m );
    fscanf( fp, "%lf", &epr );
    fscanf( fp, "%lf", &epa );
    printf( "\ter  =%8.3g\n", epr );
    printf( "\tea  =%8.3g\n", epa );
    printf( "\tnst =  %6d\n", nst );

    fclose( fp );
    for (k = 5;k <= 10;k += 5)
    {
        xf = (double)k;
        if (k==10) printf( "\n     ** Input **\n\n" );
        printf( "\txf  =%8.3g\n", xf   );

        ierr = ASL_dkhncs(f, &x, y, m, xf, epr, epa, &nst, wk);

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

```
            printf( "\n\tx    = %8.3g\n", x );

            printf( "\n\tSolution\n\n" );
            for( i=0 ; i<m+1 ; i++ )
            {
                printf( "\t  y[%6d]=%8.3g\n",i,y[i] );
            }

            printf( "\n\tStep Number of Calculation\n\n" );
            printf( "\t  nst  = %6d\n", nst  );
        }

        free( y  );
        free( wk );

        return 0;
    }
```

(d) 出力結果

```
*** ASL_dkhncs ***

 ** Input **

x   =       1
y[     0]=       5
y[     1]=       3

m   =       2
er  =       0
ea  =   1e-10
nst =       0
xf  =       5

 ** Output **

ierr =      0

x    =        5

Solution

  y[     0]=   1e+03
  y[     1]=     800
  y[     2]=     480

Step Number of Calculation

  nst  =    119

 ** Input **

xf  =      10

 ** Output **

ierr =      0

x    =       10

Solution

  y[     0]= 1.6e+04
  y[     1]= 6.4e+03
  y[     2]=1.92e+03

Step Number of Calculation

  nst  =    176
```

2.2.7 ASL_dkmncn, ASL_rkmncn
$M\mathbf{y}'' + C\mathbf{y}' + K\mathbf{y} = \mathbf{p}(x)$ 型常微分方程式

(1) **機 能**

運動方程式で知られる行列形式連立 2 階常微分方程式

$$M\boldsymbol{y}'' + C\boldsymbol{y}' + K\boldsymbol{y} = \boldsymbol{p}(x)$$

の初期値問題を解く. ここで, M は質量行列, C は減衰行列, K は剛性行列とそれぞれ呼ばれる.

(2) **使用法**

倍精度関数:

ierr = ASL_dkmncn (m, c, k, n, po, p, y, h, sta, & isw, iwk, wk);

単精度関数:

ierr = ASL_rkmncn (m, c, k, n, po, p, y, h, sta, & 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	m	{D∗ / R∗}	n×n	入 力	質量行列 M
2	c	{D∗ / R∗}	n×n	入 力	減衰行列 C
3	k	{D∗ / R∗}	n×n	入 力	剛性行列 K
4	n	I	1	入 力	連立数 n
5	po	{D∗ / R∗}	n	入 力	$x = x_0$ における外力 $\boldsymbol{p}(x_0)$ (最初に呼ばれるとき値の入力必要)
				出 力	$\boldsymbol{p}(x_0 + h)$ の値で更新 (続けて積分するときの入力値)
6	p	{D∗ / R∗}	n	入 力	$x = x_0 + h$ における外力 $\boldsymbol{p}(x_0 + h)$ の値
7	y	{D∗ / R∗}	n×3	入 力	$x = x_0$ での初期値 $y_i^{(j)}(i = 1, \cdots, \mathrm{n};\ j = 0, 1)$ (最初に使用するとき値を入力) 続けて積分するときは, 前回出力された y[(i − 1) + n ∗ j] をそのまま入力する.
				出 力	$x = x_0 + h$ での計算解 $y_i^{(j)}(i = 1, \cdots, \mathrm{n}; j = 0, 1, 2)$ (続けて積分するときの入力値)
8	h	{D / R}	1	入 力	x のきざみ幅 h
9	sta	{D / R}	1	入 力	解を安定に求めるための定数 θ (既定値 1.4)
10	isw	I*	1	入 力	きざみ幅変更スイッチ 最初は 0 を入力 h, sta を変更せず続けて積分するときは, 前回の出力値を入力 h, sta を変更するときは 2 を入力.
				出 力	h, sta を変更せず続けて積分するときの次の入力値
11	iwk	I*	n	ワーク	作業領域
12	wk	{D∗ / R∗}	n × n + n	ワーク	作業領域
13	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) n ≥ 1

(b) sta ≥ 1.0 (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 (b) 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	連立 1 次方程式が解けなかった. 最初の積分であるのに isw $\neq 0$ であった.	

(6) **注意事項**

(a) この関数は十分小さなきざみ幅 h に対して, $x = x_0$ での条件を与えて $x = x_0 + h$ での解を計算する. したがって, 始点 x_0 と終点 x_f の差 $x_f - x_0$ が大きい場合には $x_f = x_0 + \sum_{i=1}^{v} h_i$ となるようにきざみ幅 $h_i (i = 1, \cdots, v)$ を設定し, v 回この関数を呼び出して計算する必要がある. また, 外力 $\boldsymbol{p}(x)$ が x に依存して変化する場合には, 各点での外力値 $\boldsymbol{p}(x_0 + \sum_{i=1}^{j} h_i)$ $(j = 1, \cdots, v)$ を用意し, p に逐次入力する必要がある.

(b) 初めて積分するときは, isw=0 にし, 外力値 $\boldsymbol{p}(x_0)$ と $\boldsymbol{p}(x_0 + h)$ を po と p にそれぞれ入力する. ここで, h はきざみ幅.

(c) 続けて積分している間は, po, y の出力値をそのまま次の入力値とすること. また isw は, きざみ幅 h または定数 sta を変更しないときは前回の出力値をそのまま次の入力値とし, 変更するときは isw=2 とすること. さらに, 作業領域 wk は決して書きかえないこと.

(d) sta の値は 1.37 以上で安定した解が得られるが, 2.0 以上だと誤差がかなり大きくなる. ウィルソンによると, 1.4 程度が良いとされている.

(e) この関数は, ウィルソンの θ 法を利用している.

(7) **使用例**

(a) 問 題

行列形式連立 2 階常微分方程式

$$M\boldsymbol{y}'' + C\boldsymbol{y}' + K\boldsymbol{y} = \boldsymbol{p}(x)$$

を解く. ここで質量行列 M, 減衰行列 C, 剛性行列 K は以下のように与える.

$$M = \begin{bmatrix} 2000.0 & -1000.0 \\ 2000.0 & -2000.0 \end{bmatrix}$$

$$C = \begin{bmatrix} 300.0 & -600.0 \\ 300.0 & -1200.0 \end{bmatrix}$$

$$K = \begin{bmatrix} 2000.0 & -1500.0 \\ 2000.0 & -3000.0 \end{bmatrix}$$

また, 外力 $\boldsymbol{p}(x)$ を

$$\boldsymbol{p}(x) = \begin{bmatrix} 100.0 \\ 100.0 \end{bmatrix}$$

(一定) とし, $x = x_0$ における初期条件は

$$\boldsymbol{y}|_{x=x_0} = \begin{bmatrix} 100.0 \\ 100.0 \end{bmatrix},\ \boldsymbol{y}'|_{x=x_0} = \begin{bmatrix} 100.0 \\ 100.0 \end{bmatrix}$$

(b) 入力データ

質量行列 M, 減衰行列 C, 剛性行列 K, n=2

外力 $\boldsymbol{p}(x)$(変数 po と p に設定) $x = x_0$ での初期値 $\boldsymbol{y}|_{x=x_0}$ と $\boldsymbol{y}'|_{x=x_0}$ (配列 y に設定), きざみ幅 h=0.01(変数 h に設定, 各ステップ毎に変更しない), sta=0.0, isw=0 (最初のみ) として, $x = x_0 + 50h$, と $x = x_0 + 100h$ での値を出力する. ただし $x_0 = 0.0$ とする.

(c) 主プログラム

```
/*      C interface example for ASL_dkmncn */

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

int main()
{
    double *m;
    double *c;
    double *k;
    int n;
    double *po;
    double *p;
    double *y;
    double h;
    double sta;
    int isw;
    int *iwk;
    double *wk;
    int ierr;
    double htmp;
    int i,j,l;
    FILE *fp;

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

    printf( "    *** ASL_dkmncn ***\n" );
    printf( "\n     ** Input **\n\n" );
    n=2;
    isw=0;

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

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

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

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

    p  = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( p  == NULL )
    {
        printf( "no enough memory for array p\n" );
```

```
    return -1;
}

y  = ( double * )malloc((size_t)( sizeof(double) * (3*n) ));
if( y  == NULL )
{
    printf( "no enough memory for array y \n" );
    return -1;
}

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

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

printf( "\tm[i][j]\n" );
printf( "\t               j=0      j=1\n" );
for( i=0 ; i<n ; i++ )
{
    printf( "\t i=%6d",i );
    for( j=0 ; j<n ; j++ )
    {
        fscanf( fp, "%lf", &m[i+n*j] );
        printf( " %8.3g",m[i+n*j] );
    }
    printf( "\n" );
}

printf( "\n\tc[i][j]\n" );
printf( "\t               j=0      j=1\n" );
for( i=0 ; i<n ; i++ )
{
    printf( "\t i=%6d",i );
    for( j=0 ; j<n ; j++ )
    {
        fscanf( fp, "%lf", &c[i+n*j] );
        printf( " %8.3g",c[i+n*j] );
    }
    printf( "\n" );
}

printf( "\n\tk[i][j]\n" );
printf( "\t               j=0      j=1\n" );
for( i=0 ; i<n ; i++ )
{
    printf( "\t i=%6d",i );
    for( j=0 ; j<n ; j++ )
    {
        fscanf( fp, "%lf", &k[i+n*j] );
        printf( " %8.3g",k[i+n*j] );
    }
    printf( "\n" );
}

printf( "\n\tn =%6d\n\n", n );
for( i=0 ; i<n ; i++ )
{
    fscanf( fp, "%lf", &po[i] );
    printf( "\tpo[%6d]=%8.3g\n",i,po[i] );
}

for( i=0 ; i<n ; i++ )
{
    fscanf( fp, "%lf", &p[i] );
    printf( "\tp[%6d] =%8.3g\n",i,p[i] );
}

printf( "\n\ty[i][j]\n" );
printf( "\t               j=0      j=1\n" );
for( i=0 ; i<n ; i++ )
{
    printf( "\t i=%6d",i );
    for( j=0 ; j<2 ; j++ )
    {
        fscanf( fp, "%lf", &y[i+n*j] );
        printf( " %8.3g",y[i+n*j] );
    }
    printf( "\n" );
}

fscanf( fp, "%lf", &h );
fscanf( fp, "%lf", &sta );
htmp=h;
printf( "\n\th   =%8.3g\n", h );
```

```
    printf( "\tsta =%8.3g\n", sta );
    printf( "\tisw =  %6d\n", isw );

    fclose( fp );
    printf( "\n     ** Output **\n\n" );
    for ( l = 1; l <= 100; l++ )
    {

        ierr = ASL_dkmncn(m, c, k, n, po, p, y, h, sta, &isw, iwk, wk);

        if(l==50 || l==100)
        {
            printf( "\n\tierr = %6d\n", ierr );
            printf( "\n\tx    =%8.3g\n", htmp   );

            printf( "\n\ty[i][j]\n" );
            printf( "\t               j=0      j=1      j=2\n" );
            for( i=0 ; i<n ; i++ )
            {
                printf( "\t i=%6d",i );
                for( j=0 ; j<3 ; j++ )
                {
                    printf( " %8.3g",y[i+n*j] );
                }
                printf( "\n" );
            }
        }
        htmp += h;
    }

    free(  m  );
    free(  c  );
    free(  k  );
    free(  po );
    free(  p  );
    free(  y  );
    free(  iwk );
    free(  wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dkmncn ***

 ** Input **

m[i][j]
                j=0      j=1
 i=     0    2e+03   -1e+03
 i=     1    2e+03   -2e+03

c[i][j]
                j=0      j=1
 i=     0      300     -600
 i=     1      300 -1.2e+03

k[i][j]
                j=0      j=1
 i=     0    2e+03 -1.5e+03
 i=     1    2e+03   -3e+03

n =     2

po[     0]=     100
po[     1]=     100
p[     0] =     100
p[     1] =     100

y[i][j]
                j=0      j=1
 i=     0      100      100
 i=     1      100      100

h   =    0.01
sta =       0
isw =       0

 ** Output **


ierr =      0

x    =     0.5

y[i][j]
                j=0      j=1      j=2
 i=     0      134       35     -139
 i=     1      124    -1.65     -185

ierr =      0

x    =       1

y[i][j]
                j=0      j=1      j=2
```

```
i=     0      134    -33.6     -129
i=     1      103    -76.4     -109
```

2.3 常微分方程式境界値問題

2.3.1 ASL_dosnnv, ASL_rosnnv
連立高階常微分方程式 (数値境界)

(1) **機 能**

境界条件を入力値として与えるものとして連立高階常微分方程式境界値問題を多点射撃法で解く.

(2) **使用法**

倍精度関数:

ierr = ASL_dosnnv (f, xa, xb, in, ib, ic, bn, m, n, x, k, er, ea, & nx, & nev, y, isw, iwk, wk);

単精度関数:

ierr = ASL_rosnnv (f, xa, xb, in, ib, ic, bn, m, n, x, k, er, ea, & nx, & nev, 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	f	−	−	入 力	x, y の関数として微分方程式を定義する 関数 f(x, y, n, alf) の関数名
2	xa	{D / R}	1	入 力	左側境界の x 座標
3	xb	{D / R}	1	入 力	右側境界の x 座標
4	in	I*	内容参照	入 力	境界条件の設定位置 (xa 側：0, xb 側：0 以外) **大きさ**: $\sum_{i=1}^{\mathrm{n}} \mathrm{m}[i-1]$
5	ib	I*	内容参照	入 力	境界条件の値を与える変数 $y_i^{(j)}$ の要素番号 i **大きさ**: $\sum_{i=1}^{\mathrm{n}} \mathrm{m}[i-1]$
6	ic	I*	内容参照	入 力	境界条件の値を与える変数 $y_i^{(j)}$ の微分階数 j **大きさ**: $\sum_{i=1}^{\mathrm{n}} \mathrm{m}[i-1]$
7	bn	{D∗ / R∗}	内容参照	入 力	変数 $y_i^{(j)}$ に与える境界条件の値 **大きさ**: $\sum_{i=1}^{\mathrm{n}} \mathrm{m}[i-1]$
8	m	I*	n	入 力	微分方程式に現れる変数 y_i の最大微分階数
9	n	I	1	入 力	方程式の連立数
10	x	{D∗ / R∗}	k	入 力	解を計算する点の x 座標 x_i

項番	引数と戻り値	型	大きさ	入出力	内　　容
11	k	I	1	入 力	計算点の数
12	er	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	1	入 力	要求局所相対精度 既定値: 倍精度: 10^{-12}, 単精度: 10^{-5}
13	ea	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	1	入 力	要求局所絶対精度 (既定値：誤差判定のための単位)
14	nx	I*	1	入 力	射撃点の最大個数
				出 力	射撃点の実際の個数
15	nev	I*	1	入 力	最大反復回数 (既定値:100)
				出 力	実際の反復回数
16	y	$\left\{\begin{matrix}\mathrm{D}*\\ \mathrm{R}*\end{matrix}\right\}$	内容参照	出 力	解 $y_j^{(v)}(x_i)$ $\mathrm{y}[(\mathrm{i}-1)+\mathrm{k}\times(\mathrm{j}-1)+\mathrm{k}\times\mathrm{n}\times\mathrm{v}]=y_j^{(v)}(x_i)$ **大きさ**: $\mathrm{k}\times\mathrm{n}\times(\max(\mathrm{m}[\mathrm{i}-1])+1)$
17	isw	I	1	入 力	パラメータ化処理スイッチ 0:パラメータ化しない 0 以外:パラメータ化する
18	iwk	I*	内容参照	ワーク	作業領域 **大きさ**: $\sum_{i=1}^{\mathrm{n}}\mathrm{m}[i-1]$
19	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**: $(\mathrm{n}\times\mathrm{km}+1)^2\times(\mathrm{nx}+1)+\mathrm{n}\times\mathrm{km}\times\max(3\times\mathrm{n}\times\mathrm{km},\mathrm{km}+15)$ ここで, $\mathrm{km}=\max(\mathrm{m}[\mathrm{i}-1])$
20	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $\mathrm{xa}<\mathrm{xb}$

(b) er$\geq e_r$. ここで $e_r=$ 倍精度: 10^{-14}, 単精度: 10^{-5} (既定値にするため, 0.0 を入力する場合は除く)

(c) ea$\geq$ 表現できる絶対値最小値 $\times 2^{24}$ (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)

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

(f) $\mathrm{m}[\mathrm{i}-1]\geq 1$ $(\mathrm{i}=1,2,\cdots,\mathrm{n})$

(g) $1\leq\mathrm{ib}[\mathrm{i}-1]\leq\mathrm{n}$ かつ $0\leq\mathrm{ic}[\mathrm{i}-1]<\mathrm{m}[\mathrm{ib}[\mathrm{i}-1]]$ $(\mathrm{i}=1,2,\cdots,\sum_{\mathrm{j}=1}^{\mathrm{n}}\mathrm{m}[\mathrm{j}-1])$

(h) $\mathrm{k}\geq 1$

(i) $\mathrm{xa}\leq\mathrm{x}[\mathrm{i}-1]\leq\mathrm{xb}$ $(\mathrm{i}=1,2,\cdots,\mathrm{k})$

(j) $\mathrm{nx}\geq\min(5\mathrm{i}+1,51)$
ここで i は $\mathrm{xb}-\mathrm{xa}\leq\mathrm{i}$ となる最小の整数

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	制限条件 (a) を満足しなかった. (ただし, xa $\neq$ xb)	xa と xb を互いに置き換えたものとして処理する.
1500	制限条件 (b), (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	xa=xb	処理を打ち切る.
3010	制限条件 (e) を満足しなかった.	
3020	制限条件 (f) を満足しなかった.	
3030	制限条件 (g) を満足しなかった.	
3040	制限条件 (h) を満足しなかった.	
3050	制限条件 (i) を満足しなかった.	
3060	制限条件 (j) を満足しなかった.	
4000	激しい振動 (大きな導関数) などにより射撃点が追加できなかった.	
4500	初期値問題計算中きざみ幅が小さくなりすぎた (特異点の存在など).	
5000	射撃点の最大個数に到達した.	
5500	最大反復回数に到達した (解が存在しない場合や不定の場合を含む).	その時点での解を返して処理を打ち切る.

(6) **注意事項**

(a) 連立高階常微分方程式の非線形問題

$$\left\{\begin{array}{lcl} y_1^{(m_1)} & = & f_1(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)}) \\ \vdots & \vdots & \vdots \\ y_i^{(m_i)} & = & f_i(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)}) \\ \vdots & \vdots & \vdots \\ y_n^{(m_n)} & = & f_n(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)}) \end{array}\right.$$

(ただし, 左辺が $y_i^{(m_i)}$ である場合対応する右辺の $y_i^{(j)}$ の微分階数 j は $j \leq m_i - 1$ を満たさなければならない.) において, 各方程式の右辺の各項に変数 $y_i^{(j)}$ の非線形項 (たとえば $y_1 \cdot y_1'$, $\frac{y_2}{y_3'}$, y_4^2 などの 2 つ以上の乗除算や, $\sin(y_1)$, $\mathrm{abs}(y_2')$ などの関数項) がある場合は, この部分に alf を乗じることによってこの非線形問題をパラメータ化する.

例

$$\left\{\begin{array}{l} y_1'' = y_2 \\ y_2' = y_1' - y_1 \cdot y_2 \end{array}\right.$$

この非線形問題は次のようにパラメータ化し, isw=1 として解く.

$$\left\{\begin{array}{l} y_1'' = y_2 \\ y_2' = y_1' - \underline{\mathrm{alf} \cdot y_1 \cdot y_2} \end{array}\right.$$

なお, 線形問題はパラメータ化する必要がなく isw=0 として解く.

(b) x, y の関数として微分方程式を定義する関数 f(x, y, n, alf) の実際の名前は, 使用者側のプログラムで宣言し, 実際の関数を作成しておかなければならない. 注意事項 (a) に述べた連立高階常微分方程式に対応する関数 f(x, y, n, alf)(倍精度) の作り方は, 次に示すとおりである.

void FORTRAN f(double ∗x, double ∗y, int ∗n, double ∗alf)
{
$\mathrm{y}[(*\mathrm{n}) * \mathrm{m}[0]] = f_1(*\mathrm{alf}, x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots);$
$\vdots$
$\mathrm{y}[\mathrm{i}-1+(*\mathrm{n}) * \mathrm{m}[\mathrm{i}-1]] = f_i(*\mathrm{alf}, x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots);$
$\vdots$
$\mathrm{y}[(*\mathrm{n})-1+(*\mathrm{n}) * \mathrm{m}[(*\mathrm{n})-1]] = f_n(*\mathrm{alf}, x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots);$
}

ただし, $x \leftrightarrow$ ∗x, $n \leftrightarrow$ ∗n, $y_i \leftrightarrow \mathrm{y}[\mathrm{i}-1]$, $y_i^{(j)} \leftrightarrow \mathrm{y}[\mathrm{i}-1+(*\mathrm{n}) * \mathrm{j}]$
と対応する. なお, $f_i(*\mathrm{alf}, x, \cdots)$ は $f_i(x, \cdots)$ をパラメータ化した式である.

例

$$\begin{cases} y_1'' = y_2 \\ y_2' = y_1' - \underline{\mathrm{alf} \cdot y_1 \cdot y_2} \end{cases}$$

パラメータ化した式が上のような場合関数はつぎのようになる

void FORTRAN f(double ∗x, double ∗y, int ∗n, double ∗alf)
{
y[4] = y[1];
y[3] = y[2] − (∗alf) ∗ (y[0] ∗ y[1]);
}

入力引数は n=2, m [0] =2, m [1] =1, isw=1 となる.

(c) 連立高階常微分方程式の境界条件が

$$\text{左側境界} \quad \begin{cases} y_{a_1}^{(b_1)} = c_1 \\ y_{a_2}^{(b_2)} = c_2 \\ \vdots \\ y_{a_p}^{(b_p)} = c_p \end{cases} \qquad \text{右側境界} \quad \begin{cases} y_{a_{p+1}}^{(b_{p+1})} = c_{p+1} \\ y_{a_{p+2}}^{(b_{p+2})} = c_{p+2} \\ \vdots \\ y_{a_q}^{(b_q)} = c_q \end{cases} \qquad q = \sum_{i=1}^{n} m_i$$

で与えられているとき, 配列 in, ib, ic, bn には次のような値を設定する.
$\mathrm{in}[i-1] = 0 \ (i = 1, 2, \cdots, p)$
$\mathrm{in}[i-1] = 1 \ (i = p+1, p+2, \cdots, q)$
$\mathrm{ib}[i-1] = a_i (i = 1, 2, \cdots, q)$
$\mathrm{ic}[i-1] = b_i (i = 1, 2, \cdots, q)$
$\mathrm{bn}[i-1] = c_i (i = 1, 2, \cdots, q)$

例 境界条件が

$$\text{左側境界} \quad \begin{cases} y_1' = 0.0 \\ y_2 = 1.0 \end{cases} \qquad \text{右側境界} \quad y_1' = 2.0$$

で与えられているとき,
in [0] =0, ib [0] =1, ic [0] =1, bn [0] =0.0
in [1] =0, ib [1] =2, ic [1] =0, bn [1] =1.0
in [2] =1, ib [2] =1, ic [2] =1, bn [2] =2.0

(7) 使用例

(a) 問 題

$$\begin{cases} y_1'' = y_2 \\ y_2' = y_1' - y_1 \cdot y_2 \end{cases}$$

の連立常微分方程式を次の境界条件で解く.

$$y_1'|_{x=0.0} = 0.0, y_1|_{x=0.0} = 0.486, y_1'|_{x=3.0} = 2.0$$

(b) 入力データ

関数 f(x, y, n, alf) の関数名: f, n=2, xa=0.0, xb=3.0,

in [0] =0, ib [0] =1, ic [0] =1, bn [0] =0.0

in [1] =0, ib [1] =1, ic [1] =0, bn [1] =0.486

in [2] =1, ib [2] =1, ic [2] =1, bn [2] =2.0

m [0] =2, m [1] =1, x, k=3, er, ea, nx, nev=0, isw=1

(c) 主プログラム

```
/*      C interface example for ASL_dosnnv */

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
void    f(double *x,double *y,int *n,double *alf)
#else
void    f(x,y,n,alf)
double *x;
double *y;
double *alf;
int    *n;
#endif
{
    y[2*(*n)]= y[1];
    y[(*n)+1]= y[(*n)]-(*alf)*(y[0]*y[1]);
}
#ifdef __cplusplus
}
#endif

int main()
{
    double ax;
    double bx;
    int *in;
    int *ib;
    int *ic;
    double *bn;
    int *m;
    int n;
    double *x;
    int k;
    double epr;
    double epa;
    int nx;
    int nev;
    double *y;
    int isw;
    int *iwk;
    double *wk;
    int ierr;
    int i,j,v,maxm,summ,nwk;
    FILE *fp;

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

    printf( "    *** ASL_dosnnv ***\n" );
    printf( "\n     ** Input **\n\n" );
    n=2;
    k=3;
    isw=0;
    m  = ( int * )malloc((size_t)( sizeof(int) * n ));
    if( m  == NULL )
    {
```

```
    printf( "no enough memory for array m\n" );
    return -1;
}
printf( "\n\tOrder of Each Differential Equations\n\n" );
maxm=summ=0;
for( i=0 ; i<n ; i++ )
{
    fscanf( fp, "%d", &m[i] );
    printf( "\t  m[%6d] = %6d\n",i,m[i]);
    maxm=( (maxm>m[i]) ? maxm : m[i] );
    summ+=m[i];
}

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

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

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

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

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

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

fscanf( fp, "%lf", &ax );
fscanf( fp, "%lf", &bx );
printf( "\txa = %8.3g\n", ax );
printf( "\txb = %8.3g\n", bx );
printf( "\n\tBoundary Condition\n\n" );
for( i=0 ; i<summ ; i++ )
{
    fscanf( fp, "%d", &in[i] );
}
for( i=0 ; i<summ ; i++ )
{
    fscanf( fp, "%d", &ib[i] );
}
for( i=0 ; i<summ ; i++ )
{
    fscanf( fp, "%d", &ic[i] );
}
for( i=0 ; i<summ ; i++ )
{
    fscanf( fp, "%lf", &bn[i] );
}
printf( "\t  Position   Index of y   Order of y    Value\n" );
for( i=0 ; i<summ ; i++ )
{
    printf( "\t %6d     %6d       %6d      %8.3g\n",in[i],ib[i],ic[i],bn[i]);
}
printf( "\n\tSimulations Number of differential equations = %6d\n",n);
printf( "\n\tPoints Where Approximate Values are Computed\n\n" );
for( i=0 ; i<k ; i++ )
{
    fscanf( fp, "%lf", &x[i] );
    printf( "\t  x[%6d] = %8.3g\n",i,x[i]);
}
printf( "\n\tNumber of Points Where Approximate Values are Computed = %6d\n",k );

fscanf( fp, "%lf", &epr );
```

```
    fscanf( fp, "%lf", &epa );
    fscanf( fp, "%d", &nx );
    fscanf( fp, "%d", &nev );
    fscanf( fp, "%d", &isw );

    nwk=(n*maxm+1)*(n*maxm+1)*(nx+1)+n*maxm*( (3*n*maxm>maxm+15) ? 3*n*maxm : maxm+15 );
    wk  = ( double * )malloc((size_t)( sizeof(double) * nwk ));
    if( wk  == NULL )
    {
        printf( "no enough memory for array wk \n" );
        return -1;
    }

    printf( "\tRequired Local Relative Precision = %8.3g\n", epr );
    printf( "\tRequired Local Absolute Precision = %8.3g\n", epa);
    printf( "\tMaximum Number of Shooting Points =   %6d\n", nx );
    printf( "\tMaximum Iterative Number = %6d\n", nev );
    printf( "\tisw = %6d\n", isw );

    fclose( fp );

    ierr = ASL_dosnnv(f, ax, bx, in, ib, ic, bn, m, n, x, k, epr, epa, &nx, &nev, y, isw, iwk, wk);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tPractical Number of Shooting Points = %6d\n", nx );
    printf( "\tPractical Iterative Number = %6d\n", nev );
    printf( "\n\tApproximate Values\n\n" );
    for( i=0 ; i<k ; i++ )
    {
        for( j=0 ; j<n ; j++ )
        {
        for( v=0; v<m[j]+1 ; v++ )
        {
        printf( "\t  y[%6d][%6d][%6d] = %8.3g\n",i,j,v,y[i+k*j+k*n*v] );
        }
        }
    }

    free( in );
    free( ib );
    free( ic );
    free( bn );
    free( m );
    free( x );
    free( y );
    free( iwk );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dosnnv ***

 ** Input **


Order of Each Differential Equations

  m[     0] =      2
  m[     1] =      1
xa =        0
xb =        3

Boundary Condition

  Position   Index of y   Order of y    Value
      0          1            1            1
      0          1            0        0.486
      1          1            1            2

Simulations Number of differential equations =      2

Points Where Approximate Values are Computed

  x[     0] =        0
  x[     1] =      1.5
  x[     2] =        3

Number of Points Where Approximate Values are Computed =      3
Required Local Relative Precision =        0
Required Local Absolute Precision =        0
Maximum Number of Shooting Points =       50
Maximum Iterative Number =      0
isw =      1

 ** Output **

ierr =      0

Practical Number of Shooting Points =     19
Practical Iterative Number =     19

Approximate Values
```

```
y[     0][     0][     0] =    0.486
y[     0][     0][     1] =        1
y[     0][     0][     2] =   -0.528
y[     0][     1][     0] =   -0.528
y[     0][     1][     1] =     1.26
y[     1][     0][     0] =     1.94
y[     1][     0][     1] =     1.19
y[     1][     0][     2] =    0.513
y[     1][     1][     0] =    0.513
y[     1][     1][     1] =    0.199
y[     2][     0][     0] =     4.34
y[     2][     0][     1] =        2
y[     2][     0][     2] =    0.486
y[     2][     1][     0] =    0.486
y[     2][     1][     1] =   -0.111
```

2.3.2 ASL_dosnnf, ASL_rosnnf
連立高階常微分方程式 (関数境界)

(1) **機 能**

境界条件を関数として与えるものとして連立高階常微分方程式境界値問題を多点射撃法で解く.

(2) **使用法**

倍精度関数:

ierr = ASL_dosnnf (f, fb, xa, xb, m, n, x, k, er, ea, & nx, & nev, y, isw, iwk, wk);

単精度関数:

ierr = ASL_rosnnf (f, fb, xa, xb, m, n, x, k, er, ea, & nx, & nev, 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	f	−	−	入 力	x, y の関数として微分方程式を定義する 関数 f(x, y, n, alf) の関数名
2	fb	−	−	入 力	境界条件を定義する関数 fb(ya, yb, n, g) の関数名
3	xa	{D / R}	1	入 力	左側境界の x 座標
4	xb	{D / R}	1	入 力	右側境界の x 座標
5	m	I*	n	入 力	連立する各方程式の左辺の微分階数
6	n	I	1	入 力	方程式の連立数
7	x	{D∗ / R∗}	k	入 力	解を計算する点の x 座標 x_i
8	k	I	1	入 力	計算点の数
9	er	{D / R}	1	入 力	要求局所相対精度 既定値: 倍精度: 10^{-12}, 単精度: 10^{-5}
10	ea	{D / R}	1	入 力	要求局所絶対精度 (既定値：誤差判定のための単位)
11	nx	I*	1	入 力	射撃点の最大個数
				出 力	射撃点の実際の個数
12	nev	I*	1	入 力	最大反復回数 (既定値:100)
				出 力	実際の反復回数
13	y	{D∗ / R∗}	内容参照	出 力	解 $y_j^{(v)}(x_i)$ $\mathrm{y}[(\mathrm{i}-1)+\mathrm{k}\times(\mathrm{j}-1)+\mathrm{k}\times\mathrm{n}\times\mathrm{v}] = y_j^{(v)}(x_i)$ **大きさ**: $\mathrm{k}\times\mathrm{n}\times(\max(\mathrm{m}[\mathrm{i}-1])+1)$
14	isw	I	1	入 力	パラメータ化処理スイッチ 0:パラメータ化しない 0 以外:パラメータ化する
15	iwk	I*	内容参照	ワーク	作業領域 **大きさ**: $\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{m}[\mathrm{i}-1]$
16	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 **大きさ**:$(\mathrm{n}\times\mathrm{km}+1)^2\times(\mathrm{nx}+1)+\mathrm{n}\times\mathrm{km}\times\max(3\times\mathrm{n}\times\mathrm{km}, \mathrm{km}+15)$ ここで, $\mathrm{km} = \max(\mathrm{m}[\mathrm{i}-1])$
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) xa $<$ xb

(b) er$\geq e_r$. ここで e_r= 倍精度: 10^{-14}, 単精度: 10^{-5} (既定値にするため, 0.0 を入力する場合は除く)

(c) ea$\geq$ 表現できる絶対値最小値 $\times 2^{24}$ (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)

(e) n ≥ 1

(f) m[i $-$ 1] ≥ 1 (i $= 1, 2, \cdots,$ n)

(g) k ≥ 1

(h) xa $\leq$ x[i $-$ 1] $\leq$ xb (i $= 1, 2, \cdots,$ k)

(i) nx $\geq \min(5\text{i} + 1, 51)$
ここで i は xb $-$ xa $\leq$ i となる最小の整数

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	制限条件 (a) を満足しなかった. (ただし, xa $\neq$ xb)	xa と xb を互いに置き換えたものとして処理する.
1500	制限条件 (b), (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	xa $=$ xb	処理を打ち切る.
3010	制限条件 (e) を満足しなかった.	
3020	制限条件 (f) を満足しなかった.	
3030	制限条件 (g) を満足しなかった.	
3040	制限条件 (h) を満足しなかった.	
3050	制限条件 (i) を満足しなかった.	
4000	激しい振動 (大きな導関数) などにより射撃点が追加できなかった.	
4500	初期値問題計算中きざみ幅が小さくなりすぎた (特異点の存在など).	
5000	射撃点の最大個数に到達した.	
5500	最大反復回数に到達した (解が存在しない場合や不定の場合を含む).	その時点での解を返して処理を打ち切る.

(6) **注意事項**

(a) 連立高階常微分方程式の非線形問題

$$\left\{\begin{array}{lcl} y_1^{(m_1)} & = & f_1(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)}) \\ \vdots & \vdots & \vdots \\ y_i^{(m_i)} & = & f_i(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)}) \\ \vdots & \vdots & \vdots \\ y_n^{(m_n)} & = & f_n(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)}) \end{array}\right.$$

(ただし, 左辺が $y_i^{(m_i)}$ である場合対応する右辺の $y_i^{(j)}$ の微分階数 j は $j \leq m_i - 1$ を満たさなければならない.) において, 各方程式の右辺の各項に変数 $y_i^{(j)}$ の非線形項 (たとえば $y_1 \cdot y_1'$, $\frac{y_2}{y_3}$, y_4^2 などの 2 つ以上の乗除算や, $\sin(y_1)$, $\mathrm{abs}(y_2')$ などの関数項) がある場合は, この部分に alf を乗じることによってこの非線形問題をパラメータ化する.

例

$$\left\{\begin{array}{l} y_1'' = y_2 \\ y_2' = y_1' - y_1 \cdot y_2 \end{array}\right.$$

この非線形問題は次のようにパラメータ化し, isw=1 として解く.

$$\left\{\begin{array}{l} y_1'' = y_2 \\ y_2' = y_1' - \underline{\mathrm{alf} \cdot y_1 \cdot y_2} \end{array}\right.$$

なお, 線形問題はパラメータ化する必要がなく isw=0 として解く.

(b) x, y の関数として微分方程式を定義する関数 f(x, y, n, alf) の実際の名前は, 使用者側のプログラムで宣言し, 実際の関数を作成しておかなければならない. 注意事項 (a) に述べた連立高階常微分方程式に対応する関数 f(x, y, n, alf)(倍精度) の作り方は, 次に示すとおりである.

void FORTRAN f(double ∗x, double ∗y, int ∗n, double ∗alf)
{
$\mathrm{y}[(*\mathrm{n}) * \mathrm{m}[0]] = f_1(*\mathrm{alf}, x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots);$
$\vdots$
$\mathrm{y}[\mathrm{i} - 1 + (*\mathrm{n}) * \mathrm{m}[\mathrm{i} - 1]] = f_i(*\mathrm{alf}, x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots);$
$\vdots$
$\mathrm{y}[(*\mathrm{n}) - 1 + (*\mathrm{n}) * \mathrm{m}[(*\mathrm{n}) - 1]] = f_n(*\mathrm{alf}, x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots);$
}

ただし, $x \leftrightarrow *\mathrm{x}$, $n \leftrightarrow *\mathrm{n}$, $y_i \leftrightarrow \mathrm{y[i-1]}$, $y_i^{(j)} \leftrightarrow \mathrm{y[i-1+(*n)*j]}$
と対応する. なお, $f_i(*\mathrm{alf}, x, \cdots)$ は $f_i(x, \cdots)$ をパラメータ化した式である.

例

$$\left\{\begin{array}{l} y_1'' = y_2 \\ y_2' = y_1' - \underline{\mathrm{alf} \cdot y_1 \cdot y_2} \end{array}\right.$$

パラメータ化した式が上のような場合, 関数はつぎのようになる

```
void FORTRAN f(double *x, double *y, int *n, double *alf)
{
   y[4] = y[1];
   y[3] - (*alf) * (y[0] * y[1]);
}
```

入力引数は n=2, m [0] =2, m [1] =1, isw=1 となる.

(c) 境界条件を定義する関数 fb(ya, yb, n, g) の実際の名前は, 使用者側のプログラムで宣言し, 実際の関数を作成しておかなければならない. いま, q 個の境界条件が左側境界 $x=a$ での $y_i^{(j)}$ の値 $y_i^{(j)}(a)$ と右側境界 $x=b$ での $y_i^{(j)}$ の値 $y_i^{(j)}(b)$ の関数として

$$\begin{cases} g_1(y_1(a),\cdots,y_i^{(j)}(a),\cdots,y_n^{(m_n-1)}(a),y_1(b),\cdots,y_i^{(j)}(b),\cdots,y_n^{(m_n-1)}(b))=0 \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ g_v(y_1(a),\cdots,y_i^{(j)}(a),\cdots,y_n^{(m_n-1)}(a),y_1(b),\cdots,y_i^{(j)}(b),\cdots,y_n^{(m_n-1)}(b))=0 \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ g_q(y_1(a),\cdots,y_i^{(j)}(a),\cdots,y_n^{(m_n-1)}(a),y_1(b),\cdots,y_i^{(j)}(b),\cdots,y_n^{(m_n-1)}(b))=0 \end{cases}$$

と与えられているとすると関数 fb(ya, yb, n, g)(倍精度) の作り方は, 次に示すとおりである.

```
void FORTRAN fb(double *x, double *y, int *n, double *g)
{
  g[0] = g_1(y_1(a),...,y_i^(j)(a),...,y_n^(m_n-1)(a),y_1(b),...,y_i^(j)(b),...,y_n^(m_n-1)(b));
  g[1] = g_2(y_1(a),...,y_i^(j)(a),...,y_n^(m_n-1)(a),y_1(b),...,y_i^(j)(b),...,y_n^(m_n-1)(b));
  ⋮
  g[q-1] = g_q(y_1(a),...,y_i^(j)(a),...,y_n^(m_n-1)(a),y_1(b),...,y_i^(j)(b),...,y_n^(m_n-1)(b));
}
```

ただし,
$n \leftrightarrow$ *n, $y_i(a) \leftrightarrow$ ya[i − 1], $y_i^{(j)}(a) \leftrightarrow$ ya[i − 1 + (*n) * j]
$y_i(b) \leftrightarrow$ yb[i − 1], $y_i^{(j)}(b) \leftrightarrow$ yb[i − 1 + (*n) * j]
と対応する.

例 境界条件が

$$\begin{cases} y_1(a)-y_2(b)=0.0 \\ y_1'(a)=1.0 \\ y_1'(b)=2.0 \end{cases}$$

で与えられているとき,

```
void FORTRAN fb(double *ya, double *yb, int *n, double *g)
{
  g[0] = ya[0] - yb[1];
  g[1] = ya[2] - 1.0;
  g[2] = yb[2] - 2.0;
}
```

(7) **使用例**

(a) 問 題

$$\begin{cases} y_1''=y_2 \\ y_2'=y_1'-y_1\cdot y_2 \end{cases}$$

の連立常微分方程式を次の境界条件で解く.

$$\begin{cases} y_1(0.0)-y_2(3.0)=0.0 \\ y_1'(0.0)=1.0 \\ y_1'(3.0)=2.0 \end{cases}$$

(b) 入力データ

関数 f(x, y, n, alf) の関数名: f1,

関数 fb(ya, yb, n, g) の関数名 f2,

n=2, xa=0.0, xb=3.0, m [0] =2, m [1] =1, x, k=3, er, ea, nx, nev=0, isw=1

(c) 主プログラム

```
/*      C interface example for ASL_dosnnf */

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
void    f1(double *x,double *y,int *n,double *alf)
#else
void    f1(x,y,n,alf)
double *x;
double *y;
double *alf;
int    *n;
#endif
{
    y[2*(*n)]= y[1];
    y[(*n)+1]= y[(*n)]-(*alf)*(y[0]*y[1]);
}
#ifdef __cplusplus
}
#endif

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
void    f2(double *ya,double *yb,int *n,double *g)
#else
void    f2(ya,yb,n,g)
double *ya;
double *yb;
double *g;
int    *n;
#endif
{
    g[0]= ya[0]-yb[1];
    g[1]= ya[(*n)]-1.0;
    g[2]= yb[(*n)]-2.0;
}
#ifdef __cplusplus
}
#endif

int main()
{
    double ax;
    double bx;
    int *m;
    int n;
    double *x;
    int k;
    double epr;
    double epa;
    int nx;
    int nev;
    double *y;
    int isw;
    int *iwk;
    double *wk;
    int ierr;
    int i,j,v,maxm,summ,nwk;
    FILE *fp;

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

    printf( "    *** ASL_dosnnf ***\n" );
    printf( "\n     ** Input **\n\n" );
    n=2;
    k=3;

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

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


    fscanf( fp, "%lf", &ax );
    fscanf( fp, "%lf", &bx );
    printf( "\txa  = %8.3g\n", ax );
    printf( "\txb  = %8.3g\n", bx );
    printf( "\n\tBoundary Conditions\n\n" );
    printf( "\t  ya1-yb2 = 0.0 \n" );
    printf( "\t  ya1'    = 1.0    (ya=y(x=0.0),yb=y(x=3.0))\n" );
    printf( "\t  yb1'    = 2.0 \n" );
    printf( "\n\tOrder of Each Differential Equations\n\n" );
    maxm=summ=0;
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%d", &m[i] );
        printf( "\t  m[%6d]=%6d\n",i,m[i]);
        maxm=( (maxm>m[i]) ? maxm : m[i] );
        summ+=m[i];
    }
    y  = ( double * )malloc((size_t)( sizeof(double) * (k*n*(maxm+1)) ));
    if( y  == NULL )
    {
        printf( "no enough memory for array y \n" );
        return -1;
    }
    iwk  = ( int * )malloc((size_t)( sizeof(int) * summ ));
    if( iwk  == NULL )
    {
        printf( "no enough memory for array iwk\n" );
        return -1;
    }

    printf( "\n\tSimulations Number of differential equations = %6d\n",n);
    printf( "\n\tPoints Where Approximate Values are Computed\n\n" );
    for( i=0 ; i<k ; i++ )
    {
        fscanf( fp, "%lf", &x[i] );
        printf( "\t  x[%6d] = %8.3g\n",i,x[i]);
    }
    printf( "\n\tNumber of Points Where Approximate Values are Computed = %6d\n",k );

    fscanf( fp, "%lf", &epr );
    fscanf( fp, "%lf", &epa );
    fscanf( fp, "%d", &nx );
    fscanf( fp, "%d", &nev );
    fscanf( fp, "%d", &isw );

    printf( "\n\tRequired Local Relative Precision  = %8.3g\n", epr );
    printf( "\tRequired Local Absolute Precision  = %8.3g\n", epa);
    printf( "\tMaximum Number of Shooting Points  =   %6d\n", nx );
    printf( "\tMaximum Iterative Number  = %6d\n", nev );
    printf( "\tisw = %6d\n", isw );

    nwk=(n*maxm+1)*(n*maxm+1)*(nx+1)+n*maxm*( (3*n*maxm>maxm+15) ? 3*n*maxm : maxm+15 );
    wk  = ( double * )malloc((size_t)( sizeof(double) * nwk ));
    if( wk  == NULL )
    {
        printf( "no enough memory for array wk \n" );
        return -1;
    }
    fclose( fp );

    ierr = ASL_dosnnf(f1, f2, ax, bx, m, n, x, k, epr, epa, &nx, &nev, y, isw, iwk, wk);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tPractical Number of Shooting Points = %6d\n", nx );
    printf( "\n\tPractical Iterative Number = %6d\n", nev );
    printf( "\n\tApproximate Values\n\n" );
    for( i=0 ; i<3 ; i++ )
    {
        for( j=0 ; j<n ; j++ )
        {
        for( v=0; v<m[j]+1 ; v++ )
        {
        printf( "\t  y[%6d][%6d][%6d] = %8.3g\n",i,j,v,y[i+k*j+k*n*v] );
        }
        }
    }

    free( m );
    free( x );
    free( y );
    free( iwk );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dosnnf ***

 ** Input **

xa  =        0
xb  =        3

Boundary Conditions

  ya1-yb2 = 0.0
  ya1'    = 1.0    (ya=y(x=0.0),yb=y(x=3.0))
  yb1'    = 2.0

Order of Each Differential Equations

  m[     0]=     2
  m[     1]=     1

Simulations Number of differential equations =      2

Points Where Approximate Values are Computed

  x[     0] =        0
  x[     1] =      1.5
  x[     2] =        3

Number of Points Where Approximate Values are Computed =      3

Required Local Relative Precision  =        0
Required Local Absolute Precision  =        0
Maximum Number of Shooting Points  =       50
Maximum Iterative Number  =      0
isw =      1

 ** Output **

ierr =      0

Practical Number of Shooting Points =     19

Practical Iterative Number =     19

Approximate Values

  y[     0][     0][     0] =    0.486
  y[     0][     0][     1] =        1
  y[     0][     0][     2] =   -0.528
  y[     0][     1][     0] =   -0.528
  y[     0][     1][     1] =     1.26
  y[     1][     0][     0] =     1.94
  y[     1][     0][     1] =     1.19
  y[     1][     0][     2] =    0.513
  y[     1][     1][     0] =    0.513
  y[     1][     1][     1] =    0.199
  y[     2][     0][     0] =     4.34
  y[     2][     0][     1] =        2
  y[     2][     0][     2] =    0.486
  y[     2][     1][     0] =    0.486
  y[     2][     1][     1] =   -0.111
```

2.3.3 ASL_dofnnv, ASL_rofnnv
連立 1 階常微分方程式 (数値境界)

(1) **機 能**

境界条件を入力値として与えるものとして連立 1 階常微分方程式境界値問題を多点射撃法で解く.

(2) **使用法**

倍精度関数:

ierr = ASL_dofnnv (f, xa, xb, in, ib, bn, n, x, k, er, ea, & nx, & nev, y, isw, iwk, wk);

単精度関数:

ierr = ASL_rofnnv (f, xa, xb, in, ib, bn, n, x, k, er, ea, & nx, & nev, 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	f	−	−	入 力	x, y の関数として微分方程式を定義する 関数 f(x, y, n, alf) の関数名
2	xa	{D / R}	1	入 力	左側境界の x 座標
3	xb	{D / R}	1	入 力	右側境界の x 座標
4	in	I*	n	入 力	境界条件の設定位置 (xa 側：0, xb 側：0 以外)
5	ib	I*	n	入 力	境界条件の値を与える変数 y_i の要素番号 i
6	bn	{D*/ R*}	n	入 力	変数 y_i に与える境界条件の値
7	n	I	1	入 力	方程式の連立数
8	x	{D*/ R*}	k	入 力	解を計算する点の x 座標 x_i
9	k	I	1	入 力	計算点の数
10	er	{D / R}	1	入 力	要求局所相対精度 既定値: 倍精度: 10^{-12}, 単精度: 10^{-5}
11	ea	{D / R}	1	入 力	要求局所絶対精度 (既定値：誤差判定のための単位)
12	nx	I*	1	入 力	射撃点の最大個数
				出 力	射撃点の実際の個数
13	nev	I*	1	入 力	最大反復回数 (既定値:100)
				出 力	実際の反復回数
14	y	{D*/ R*}	内容参照	出 力	解 $y_j^{(v)}(x_i)$ $\mathrm{y}[(\mathrm{i}-1)+\mathrm{k}\times(\mathrm{j}-1)+\mathrm{k}\times\mathrm{n}\times\mathrm{v}] = y_j^{(v)}(x_i)$ **大きさ**: k×n×2
15	isw	I	1	入 力	パラメータ化処理スイッチ 0:パラメータ化しない 0 以外:パラメータ化する
16	iwk	I*	n	ワーク	作業領域
17	wk	{D*/ R*}	内容参照	ワーク	作業領域 **大きさ**: $(\mathrm{n}+1)^2\times(\mathrm{nx}+1)+\mathrm{n}\times\max(2\times\mathrm{n}, 17)$
18	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) xa $<$ xb

(b) er$\geq e_r$. ここで e_r= 倍精度: 10^{-14}, 単精度: 10^{-5} (既定値にするため, 0.0 を入力する場合は除く)

(c) ea$\geq$ 表現できる絶対値最小値 $\times 2^{24}$ (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)

(e) n ≥ 1

(f) $1 \leq$ ib[i $-$ 1] $\leq$ n (i $= 1, 2, \cdots, \sum_{\mathrm{j}=1}^{\mathrm{n}}$ m[j $-$ 1])

(g) k ≥ 1

(h) xa $\leq$ x[i $-$ 1] $\leq$ xb (i $= 1, 2, \cdots,$ k)

(i) nx $\geq \min(5\mathrm{i} + 1, 51)$
ここで i は xb $-$ xa $\leq$ i となる最小の整数

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	制限条件 (a) を満足しなかった. (ただし, xa $\neq$ xb)	xa と xb を互いに置き換えたものとして処理する.
1500	制限条件 (b), (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	xa=xb	処理を打ち切る.
3010	制限条件 (e) を満足しなかった.	
3020	制限条件 (f) を満足しなかった.	
3030	制限条件 (g) を満足しなかった.	
3040	制限条件 (h) を満足しなかった.	
3050	制限条件 (i) を満足しなかった.	
4000	激しい振動 (大きな導関数) などにより射撃点が追加できなかった.	
4500	初期値問題計算中きざみ幅が小さくなりすぎた (特異点の存在など).	
5000	射撃点の最大個数に到達した.	
5500	最大反復回数に到達した (解が存在しない場合や不定の場合を含む).	その時点での解を返して処理を打ち切る.

(6) **注意事項**

(a) 連立 1 階常微分方程式の非線形問題

$$\left\{\begin{array}{rcl} y_1' &=& f_1(x, y_1, y_2, \cdots, y_n) \\ y_2' &=& f_2(x, y_1, y_2, \cdots, y_n) \\ \vdots & \vdots & \vdots \\ y_n' &=& f_n(x, y_1, y_2, \cdots, y_n) \end{array}\right.$$

において, 各方程式の右辺の各項に変数 y_i の非線形項 (たとえば $y_1 \cdot y_2$, $\frac{y_2}{y_3}$, y_4^2 などの 2 つ以上の乗除算や $\sin(y_1)$, $\mathrm{abs}(y_2)$ などの関数項) がある場合は, この部分に alf を乗じることによってこの非線形問題をパラメータ化する.

例

$$\left\{\begin{array}{l} y_1' = y_2 \\ y_2' = y_1 - y_1 \cdot y_2 \end{array}\right.$$

この非線形問題は次のようにパラメータ化し, isw=1 として解く.

$$\left\{\begin{array}{l} y_1' = y_2 \\ y_2' = y_1 - \underline{\mathrm{alf} \cdot y_1 \cdot y_2} \end{array}\right.$$

なお, 線形問題はパラメータ化する必要がなく isw=0 として解く.

(b) x, y の関数として微分方程式を定義する関数 f(x, y, n, alf) の実際の名前は, 使用者側のプログラムで宣言し, 実際の関数を作成しておかなければならない. 注意事項 (a) に述べた連立 1 階常微分方程式に対応する関数 f(x, y, n, alf)(倍精度) の作り方は, 次に示すとおりである.

void FORTRAN f(double ∗x, double ∗y, int ∗n, double ∗alf)
{
y[(∗n)] = $f_1(*\mathrm{alf}, x, y_1, \cdots, y_i, \cdots, y_n)$;
⋮
y[i − 1 + (∗n)] = $f_i(*\mathrm{alf}, x, y_1, \cdots, y_i, \cdots, y_n)$;
⋮
y[2 ∗ (∗n) − 1] = $f_n(*\mathrm{alf}, x, y_1, \cdots, y_i, \cdots, y_n)$;
}

ただし, $x \leftrightarrow$ ∗x, $n \leftrightarrow$ ∗n, $y_i \leftrightarrow$ y[i − 1], と対応する. なお, $f_i(*\mathrm{alf}, x, \cdots)$ は $f_i(x, \cdots)$ をパラメータ化した式である.

例

$$\left\{\begin{array}{l} y_1' = y_2 \\ y_2' = y_1 - \underline{\mathrm{alf} \cdot y_1 \cdot y_2} \end{array}\right.$$

パラメータ化した式が上のような場合関数はつぎのようになる

void FORTRAN f(double ∗x, double ∗y, int ∗n, double ∗alf)
{
y[2] = y[1];
y[3] = y[0] − (∗alf) ∗ (y[0] ∗ y[1]);
}

入力引数は n=2, isw=1 となる.

(c) 連立 1 階常微分方程式の境界条件が

$$\text{左側境界}\quad \begin{cases} y_{a_1} = c_1 \\ y_{a_2} = c_2 \\ \quad\vdots \\ y_{a_p} = c_p \end{cases} \qquad \text{右側境界}\quad \begin{cases} y_{a_{p+1}} = c_{p+1} \\ y_{a_{p+2}} = c_{p+2} \\ \quad\vdots \\ y_{a_n} = c_n \end{cases}$$

で与えられているとき, 配列 in, ib, bn には次のような値を設定する.

$\mathrm{in}[i-1] = 0\ (i = 1, 2, \cdots, p)$

$\mathrm{in}[i-1] = 1\ (i = p+1, p+2, \cdots, n)$

$\mathrm{ib}[i-1] = a_i (i = 1, 2, \cdots, n)$

$\mathrm{bn}[i-1] = c_i (i = 1, 2, \cdots, n)$

例 境界条件が

$$\text{左側境界}\quad y_1 = 1.0 \quad \text{右側境界}\quad y_2 = 2.0$$

で与えられているとき,

in [0] =0, ib [0] =1, bn [0] =1.0

in [1] =1, ib [1] =2, bn [1] =2.0

(7) **使用例**

(a) 問 題

$$\begin{cases} y_1' = y_2 \\ y_2' = -y_1' - y_1 \cdot y_2 \end{cases}$$

の連立常微分方程式を次の境界条件で解く.

$$y_2|_{x=0.0} = 1.0, y_2|_{x=1.0} = 2.0$$

(b) 入力データ

関数 f(x, y, n, alf) の関数名: f, n=2, xa=0.0, xb=1.0,

in [0] =0, ib [0] =2, bn [0] =1.0,

in [1] =1, ib [1] =2, bn [1] =2.0,

x, k=3, er, ea, nx, nev=0, isw=1

(c) 主プログラム

```
/*      C interface example for ASL_dofnnv */

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
void    f(double *x,double *y,int *n,double *alf)
#else
void    f(x,y,n,alf)
double *x;
double *y;
double *alf;
int    *n;
#endif
{
    y[(*n)]  = y[1];
    y[(*n)+1]= -y[0]-(*alf)*(y[0]*y[1]);
}
#ifdef __cplusplus
}
#endif

int main()
{
    double ax;
```

```
double bx;
int *in;
int *ib;
double *bn;
int mx;
double *x;
int m;
double epr;
double epa;
int nx;
int nev;
double *y;
int isw;
int *iwk;
double *wk;
int ierr;
int nwk;
int i,j,k;
FILE *fp;

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

printf( "    *** ASL_dofnnv ***\n" );
printf( "\n     ** Input **\n\n" );
mx=2;
m=3;

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

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

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

fscanf( fp, "%lf", &ax );
fscanf( fp, "%lf", &bx );
printf( "\txa  = %8.3g\n", ax );
printf( "\txb  = %8.3g\n", bx );
printf( "\n\tBoundary Condition\n\n" );
for( i=0 ; i<mx ; i++ )
{
    fscanf( fp, "%d", &in[i] );
}
for( i=0 ; i<mx ; i++ )
{
    fscanf( fp, "%d", &ib[i] );
}
for( i=0 ; i<mx ; i++ )
{
    fscanf( fp, "%lf", &bn[i] );
}
printf( "\t  Position   Index of y       Value\n" );
for( i=0 ; i<mx ; i++ )
{
    printf( "\t%6d     %6d         %8.3g\n",in[i],ib[i],bn[i]);
}
printf( "\n\tSimulations Number of Differential Equations = %6d\n",mx);
printf( "\n\tPoints Where Approximate Values are Computed \n" );
for( i=0 ; i<m ; i++ )
{
    fscanf( fp, "%lf", &x[i] );
    printf( "\t  x[%6d] = %8.3g\n",i,x[i]);
}
printf( "\n\tNumber of Points Where Approximate Values are Computed = %6d\n",m );

fscanf( fp, "%lf", &epr );
```

```
    fscanf( fp, "%lf", &epa );
    fscanf( fp, "%d", &nx );
    fscanf( fp, "%d", &nev );
    fscanf( fp, "%d", &isw );

    nwk = (mx+1)*(mx+1)*(nx+1)+mx*( (2*mx> 17) ? 2*mx :  17 );
    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) * mx ));
    if( iwk  == NULL )
    {
        printf( "no enough memory for array iwk\n" );
        return -1;
    }

    printf( "\n\tRequired Local Relative Precision  = %8.3g\n", epr );
    printf( "\tRequired Local Absolute Precision  = %8.3g\n", epa);
    printf( "\tMaximum Number of Shooting Points  =   %6d\n", nx );
    printf( "\tMaximum Iterative Number  = %6d\n", nev );
    printf( "\tisw =  %6d\n", isw );

    fclose( fp );

    ierr = ASL_dofnnv(f, ax, bx, in, ib, bn, mx, x, m, epr, epa, &nx, &nev, y, isw, iwk, wk);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tPractical Number of Shooting Points = %6d\n", nx );
    printf( "\tPractical Iterative Number = %6d\n", nev );
    printf( "\n\tApproximate Values\n\n" );
    for( i=0 ; i<m ; i++ )
    {
        for( j=0 ; j<mx ; j++ )
        {
        for( k=0; k<2 ; k++ )
        {
        printf( "\t  y[%6d][%6d][%6d] = %8.3g\n",i,j,k,y[i+m*(j+mx*k)] );
        }
        }
    }

    free( in );
    free( ib );
    free( bn );
    free( x );
    free( y );
    free( iwk );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dofnnv ***

 ** Input **

xa  =        0
xb  =        1

Boundary Condition

  Position   Index of y       Value
     0          2               1
     1          2               2

Simulations Number of Differential Equations =      2

Points Where Approximate Values are Computed
  x[     0] =        0
  x[     1] =      0.5
  x[     2] =        1

Number of Points Where Approximate Values are Computed =      3

Required Local Relative Precision  =        0
Required Local Absolute Precision  =        0
Maximum Number of Shooting Points  =       50
Maximum Iterative Number  =      0
isw =       1

 ** Output **

ierr =      0

Practical Number of Shooting Points =      6
Practical Iterative Number =     20

Approximate Values

  y[     0][     0][     0] =    -1.27
```

```
y[     0][     0][     1] =        1
y[     0][     1][     0] =        1
y[     0][     1][     1] =     2.54
y[     1][     0][     0] =    -0.46
y[     1][     0][     1] =     2.16
y[     1][     1][     0] =     2.16
y[     1][     1][     1] =     1.45
y[     2][     0][     0] =    0.655
y[     2][     0][     1] =        2
y[     2][     1][     0] =        2
y[     2][     1][     1] =    -1.96
```

2.3.4 ASL_dofnnf, ASL_rofnnf
連立 1 階常微分方程式 (関数境界)

(1) **機 能**

境界条件を関数として与えるものとして連立 1 階常微分方程式境界値問題を多点射撃法で解く.

(2) **使用法**

倍精度関数:

ierr = ASL_dofnnf (f, fb, xa, xb, n, x, k, er, ea, & nx, & nev, y, isw, iwk, wk);

単精度関数:

ierr = ASL_rofnnf (f, fb, xa, xb, n, x, k, er, ea, & nx, & nev, 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	f	–	–	入 力	x, y の関数として微分方程式を定義する 関数 f(x, y, n, alf) の関数名
2	fb	–	–	入 力	境界条件を定義する関数 fb(ya, yb, n, g) の関数名
3	xa	{D / R}	1	入 力	左側境界の x 座標
4	xb	{D / R}	1	入 力	右側境界の x 座標
5	n	I	1	入 力	方程式の連立数
6	x	{D∗ / R∗}	k	入 力	解を計算する点の x 座標 x_i
7	k	I	1	入 力	計算点の数
8	er	{D / R}	1	入 力	要求局所相対精度 既定値: 倍精度: 10^{-12}, 単精度: 10^{-5}
9	ea	{D / R}	1	入 力	要求局所絶対精度 (既定値：誤差判定のための単位)
10	nx	I*	1	入 力	射撃点の最大個数
				出 力	射撃点の実際の個数
11	nev	I*	1	入 力	最大反復回数 (既定値:100)
				出 力	実際の反復回数
12	y	{D∗ / R∗}	内容参照	出 力	解 $y_j^{(v)}(x_i)$ $\mathrm{y}[(\mathrm{i}-1)+\mathrm{k}\times(\mathrm{j}-1)+\mathrm{k}\times\mathrm{n}\times\mathrm{v}] = y_j^{(v)}(x_i)$ **大きさ**: k×n×2
13	isw	I	1	入 力	パラメータ化処理スイッチ 0:パラメータ化しない 0 以外:パラメータ化する
14	iwk	I*	n	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**: $(\mathrm{n}+1)^2\times(\mathrm{nx}+1)+\mathrm{n}\times\max(2\times\mathrm{n}, 17)$
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) xa $<$ xb

(b) er$\geq e_r$. ここで e_r= 倍精度: 10^{-14}, 単精度: 10^{-5} (既定値にするため, 0.0 を入力する場合は除く)

(c) ea$\geq$ 表現できる絶対値最小値 $\times 2^{24}$ (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)

(e) n ≥ 1

(f) k ≥ 1

(g) xa $\leq$ x[i $-$ 1] $\leq$ xb (i $= 1, 2, \cdots,$ k)

(h) nx $\geq \min(5\mathrm{i} + 1, 51)$
ここで i は xb $-$ xa $\leq$ i となる最小の整数

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	制限条件 (a) を満足しなかった. (ただし, xa $\neq$ xb)	xa と xb を互いに置き換えたものとして処理する.
1500	制限条件 (b), (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	xa=xb	処理を打ち切る.
3010	制限条件 (e) を満足しなかった.	
3020	制限条件 (f) を満足しなかった.	
3030	制限条件 (g) を満足しなかった.	
3040	制限条件 (h) を満足しなかった.	
4000	激しい振動 (大きな導関数) などにより射撃点が追加できなかった.	
4500	初期値問題計算中きざみ幅が小さくなりすぎた (特異点の存在など).	
5000	射撃点の最大個数に到達した.	処理を打ち切る.
5500	最大反復回数に到達した (解が存在しない場合や不定の場合を含む).	その時点での解を返して処理を打ち切る.

(6) **注意事項**

(a) 連立 1 階常微分方程式の非線形問題

$$\left\{\begin{array}{lcl} y_1' & = & f_1(x, y_1, y_2, \cdots, y_n) \\ y_2' & = & f_2(x, y_1, y_2, \cdots, y_n) \\ \vdots & \vdots & \vdots \\ y_n' & = & f_n(x, y_1, y_2, \cdots, y_n) \end{array}\right.$$

において, 各方程式の右辺の各項に変数 y_i の非線形項 (たとえば $y_1 \cdot y_2$, $\frac{y_2}{y_3}$, y_4^2 などの 2 つ以上の乗除算や $\sin(y_1)$, $\mathrm{abs}(y_2)$ などの関数項) がある場合は, この部分に alf を乗じることによってこの非線形問題を

パラメータ化する.

例

$$\begin{cases} y_1' = y_2 \\ y_2' = y_1 - y_1 \cdot y_2 \end{cases}$$

この非線形問題は次のようにパラメータ化し, isw=1 として解く.

$$\begin{cases} y_1' = y_2 \\ y_2' = y_1 - \underline{\mathrm{alf} \cdot y_1 \cdot y_2} \end{cases}$$

なお, 線形問題はパラメータ化する必要がなく isw=0 として解く.

(b) x, y の関数として微分方程式を定義する関数 f(x, y, n, alf) の実際の名前は, 使用者側のプログラムで宣言し, 実際の関数を作成しておかなければならない. 注意事項 (a) に述べた連立 1 階常微分方程式に対応する関数 f(x, y, n, alf)(倍精度) の作り方は, 次に示すとおりである.

void FORTRAN f(double ∗x, double ∗y, int ∗n, double ∗alf)
{
$\mathrm{y}[(*\mathrm{n})] = f_1(*\mathrm{alf}, x, y_1, \cdots, y_i, \cdots, y_n);$
$\vdots$
$\mathrm{y}[\mathrm{i}-1+(*\mathrm{n})] = f_i(*\mathrm{alf}, x, y_1, \cdots, y_i, \cdots, y_n);$
$\vdots$
$\mathrm{y}[2*(*\mathrm{n})-1] = f_n(*\mathrm{alf}, x, y_1, \cdots, y_i, \cdots, y_n);$
}

ただし, $x \leftrightarrow *$x, $n \leftrightarrow *$n, $y_i \leftrightarrow$ y[i − 1], と対応する. なお, $f_i(*\mathrm{alf}, x, \cdots)$ は $f_i(x, \cdots)$ をパラメータ化した式である.

例

$$\begin{cases} y_1' = y_2 \\ y_2' = y_1 - \underline{\mathrm{alf} \cdot y_1 \cdot y_2} \end{cases}$$

パラメータ化した式が上のような場合関数はつぎのようになる

void FORTRAN f(double ∗x, double ∗y, int ∗n, double ∗alf)
{
y[2] = y[1];
y[3] = y[0] − (∗alf) ∗ (y[0] ∗ y[1]);
}

入力引数は n=2, isw=1 となる.

(c) 境界条件を定義する関数 fb(ya, yb, n, g) の実際の名前は, 使用者側のプログラムで宣言し, 実際の関数を作成しておかなければならない. いま, n 個の境界条件が左側境界 $x = a$ での y_i の値 $y_i(a)$ と右側境界 $x = b$ での y_i の値 $y_i(b)$ の関数として

$$\begin{cases} g_1(y_1(a), \cdots, y_i, \cdots, y_n(a), y_1(b), \cdots, y_i(b), \cdots, y_n(b)) = 0 \\ \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ g_v(y_1(a), \cdots, y_i, \cdots, y_n(a), y_1(b), \cdots, y_i(b), \cdots, y_n(b)) = 0 \\ \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ g_n(y_1(a), \cdots, y_i, \cdots, y_n(a), y_1(b), \cdots, y_i(b), \cdots, y_n(b)) = 0 \end{cases}$$

と与えられているとすると関数 fb(ya, yb, n, g)(倍精度) の作り方は, 次に示すとおりである.

void FORTRAN fb(double ∗ya, double ∗yb, int ∗n, double ∗g)
{
g[0] $= g_1(y_1(a), \cdots, y_i, \cdots, y_n(a), y_1(b), \cdots, y_i(b), \cdots, y_n(b))$;
g[1] $= g_2(y_1(a), \cdots, y_i, \cdots, y_n(a), y_1(b), \cdots, y_i(b), \cdots, y_n(b))$;
$\vdots$
g[$n-1$] $= g_n(y_1(a), \cdots, y_i, \cdots, y_n(a), y_1(b), \cdots, y_i(b), \cdots, y_n(b))$;
}

ただし,
$n \leftrightarrow$ ∗n, $y_i(a) \leftrightarrow$ ya[i − 1], $y_i(b) \leftrightarrow$ yb[i − 1] と対応する.

例

境界条件が

$$\begin{cases} y_1(a) - y_2(b) = 0.0 \\ y_2(a) = 1.0 \end{cases}$$

で与えられているとき,

void FORTRAN fb(double ∗ya, double ∗yb, int ∗n, double ∗g)
{
g[0] = ya[0] − yb[1];
g[1] = ya[1] − 1.0;
}

(7) **使用例**

(a) 問 題

$$\begin{cases} y_1' = y_2 \\ y_2' = -y_1 - y_1 \cdot y_2 \end{cases}$$

の連立常微分方程式を次の境界条件で解く.

$$\begin{cases} y_1(0.0) - y_2(1.0) = 0.0 \\ y_2(0.0) = 1.0 \end{cases}$$

(b) 入力データ

関数 f(x, y, n, alf) の関数名: f1, 関数 fb(ya, yb, n, g) の関数名 f2, n=2, xa=0.0, xb=1.0, x, k=3, er, ea, nx, nev=0, isw=1

(c) 主プログラム

```
/*      C interface example for ASL_dofnnf */

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
void    f1(double *x,double *y,int *n,double *alf)
#else
void    f1(x,y,n,alf)
double *x;
double *y;
double *alf;
int    *n;
#endif
{
    y[(*n)]= y[1];
    y[(*n)+1]= -y[0]-(*alf)*(y[0]*y[1]);

}
#ifdef __cplusplus
```

```
}
#endif

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
void    f2(double *ya,double *yb,int *n,double *g)
#else
void    f2(ya,yb,n,g)
double *ya;
double *yb;
double *g;
int    *n;
#endif
{
    g[0]= ya[0]-yb[1];
    g[1]= ya[1]-1.0;
}
#ifdef __cplusplus
}
#endif


int main()
{
    double ax;
    double bx;
    int n;
    double *x;
    int k;
    double epr;
    double epa;
    int nx;
    int nev;
    double *y;
    int isw;
    int *iwk;
    double *wk;
    int ierr;
    int i,j,v,nwk;
    FILE *fp;

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

    printf( "    *** ASL_dofnnf ***\n" );
    printf( "\n     ** Input **\n\n" );
    n=2;
    k=3;

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

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

    fscanf( fp, "%lf", &ax );
    fscanf( fp, "%lf", &bx );
    printf( "\txa  =%8.3g\n", ax );
    printf( "\txb  =%8.3g\n", bx );
    printf( "\n\tBoundary Conditions\n" );
    printf( "\t  ya1-yb2 = 0.0 \n" );
    printf( "\t  ya2     = 1.0    (ya=y(x=0.0),yb=y(x=1.0))\n" );
    printf( "\n\tSimulations Number of differential equations =%6d\n",n);
    printf( "\n\tPoints Where Approximate Values are Computed \n" );
    for( i=0 ; i<k ; i++ )
    {
        fscanf( fp, "%lf", &x[i] );
        printf( "\t  x[%6d] = %8.3g\n",i,x[i]);
    }
    printf( "\n\tNumber of Points Where Approximate Values are Computed =%6d\n",k );

    fscanf( fp, "%lf", &epr );
    fscanf( fp, "%lf", &epa );
    fscanf( fp, "%d", &nx );
    fscanf( fp, "%d", &nev );
```

```
    fscanf( fp, "%d", &isw );

    nwk=(n+1)*(n+1)*(nx+1)+n*( (2*n>17) ? 2*n : 17 );
    wk  = ( double * )malloc((size_t)( sizeof(double) * nwk ));
    if( wk  == NULL )
    {
        printf( "no enough memory for array wk \n" );
        return -1;
    }

    printf( "\n\tRequired Local Relative Precision = %8.3g\n", epr );
    printf( "\tRequired Local Absolute Precision = %8.3g\n", epa);
    printf( "\tMaximum Number of Shooting Points =   %6d\n", nx );
    printf( "\tMaximum Iterative Number = %6d\n", nev );
    printf( "\tisw = %6d\n", isw );

    fclose( fp );

    ierr = ASL_dofnnf(f1, f2, ax, bx, n, x, k, epr, epa, &nx, &nev, y, isw, iwk, wk);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tPractical Number of Shooting Points = %6d\n", nx );
    printf( "\tPractical Iterative Number = %6d\n", nev );
    printf( "\n\tApproximate Values\n\n" );
    for( i=0 ; i<k ; i++ )
    {
        for( j=0 ; j<n ; j++ )
        {
        for( v=0; v<2 ; v++ )
        {
            printf( "\t  y[%6d][%6d][%6d] = %8.3g\n",i,j,v,y[i+k*j+k*n*v] );
        }
        }
    }

    free( x );
    free( y );
    free( iwk );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dofnnf ***

 ** Input **

xa  =       0
xb  =       1

Boundary Conditions
  ya1-yb2 = 0.0
  ya2     = 1.0    (ya=y(x=0.0),yb=y(x=1.0))

Simulations Number of differential equations =     2

Points Where Approximate Values are Computed
  x[     0] =        0
  x[     1] =      0.5
  x[     2] =        1

Number of Points Where Approximate Values are Computed =     3

Required Local Relative Precision =        0
Required Local Absolute Precision =        0
Maximum Number of Shooting Points =       50
Maximum Iterative Number =      0
isw =      1

 ** Output **

ierr =      0

Practical Number of Shooting Points =      6
Practical Iterative Number =     18

Approximate Values

  y[     0][     0][     0] =    0.159
  y[     0][     0][     1] =        1
  y[     0][     1][     0] =        1
  y[     0][     1][     1] =   -0.317
  y[     1][     0][     0] =    0.584
  y[     1][     0][     1] =    0.649
  y[     1][     1][     0] =    0.649
  y[     1][     1][     1] =   -0.963
  y[     2][     0][     0] =    0.785
  y[     2][     0][     1] =    0.159
  y[     2][     1][     0] =    0.159
  y[     2][     1][     1] =   -0.909
```

2.3.5 ASL_dohnnv, ASL_rohnnv 高階常微分方程式 (数値境界)

(1) **機 能**

境界条件を入力値として与えるものとして高階常微分方程式境界値問題を多点射撃法で解く.

(2) **使用法**

倍精度関数:

ierr = ASL_dohnnv (f, xa, xb, in, ic, bn, m, x, k, er, ea, & nx, & nev, y, isw, iwk, wk);

単精度関数:

ierr = ASL_rohnnv (f, xa, xb, in, ic, bn, m, x, k, er, ea, & nx, & nev, 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	f	−	−	入 力	x, y の関数として微分方程式を定義する 関数 f(x, y, m, alf) の関数名
2	xa	{D / R}	1	入 力	左側境界の x 座標
3	xb	{D / R}	1	入 力	右側境界の x 座標
4	in	I*	m	入 力	境界条件の設定位置 (xa 側: 0, xb 側: 0 以外)
5	ic	I*	m	入 力	境界条件の値を与える変数 $y^{(j)}$ の微分階数 j
6	bn	{D∗ / R∗}	m	入 力	変数 $y^{(j)}$ に与える境界条件の値
7	m	I	1	入 力	微分方程式に現れる変数 y の最大微分階数
8	x	{D∗ / R∗}	k	入 力	解を計算する点の x 座標 x_i
9	k	I	1	入 力	計算点の数
10	er	{D / R}	1	入 力	要求局所相対精度 既定値: 倍精度: 10^{-12}, 単精度: 10^{-5}
11	ea	{D / R}	1	入 力	要求局所絶対精度 (既定値: 誤差判定のための単位)
12	nx	I*	1	入 力	射撃点の最大個数
				出 力	射撃点の実際の個数
13	nev	I*	1	入 力	最大反復回数 (既定値:100)
				出 力	実際の反復回数
14	y	{D∗ / R∗}	k×(m+1)	出 力	解 $y^{(v)}(x_i)$ $\mathrm{y}[(\mathrm{i}-1)+\mathrm{k}\times\mathrm{v}] = y^{(v)}(x_i)$
15	isw	I	1	入 力	パラメータ化処理スイッチ 0:パラメータ化しない 0 以外:パラメータ化する
16	iwk	I*	m	ワーク	作業領域
17	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 **大きさ**: $(\mathrm{m}+1)^2\times(\mathrm{nx}+1)+\mathrm{m}\times\max(2\times\mathrm{m}+1, 16)$
18	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) xa $<$ xb

(b) er$\geq e_r$. ここで e_r= 倍精度: 10^{-14}, 単精度: 10^{-5} (既定値にするため, 0.0 を入力する場合は除く)

(c) ea$\geq$ 表現できる絶対値最小値 $\times 2^{24}$ (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)

(e) m ≥ 1

(f) $0 \leq$ ic[i $-$ 1] $<$ m (i $= 1, 2, \cdots,$ m)

(g) k ≥ 1

(h) xa $\leq$ x[i $-$ 1] $\leq$ xb (i $= 1, 2, \cdots,$ k)

(i) nx $\geq \min(5\text{i} + 1, 51)$
ここで i は xb $-$ xa $\leq$ i となる最小の整数

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	制限条件 (a) を満足しなかった. (ただし, xa $\neq$ xb)	xa と xb を互いに置き換えたものとして処理する.
1500	制限条件 (b), (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	xa=xb	処理を打ち切る.
3010	制限条件 (e) を満足しなかった.	
3020	制限条件 (f) を満足しなかった.	
3030	制限条件 (g) を満足しなかった.	
3040	制限条件 (h) を満足しなかった.	
3050	制限条件 (i) を満足しなかった.	
4000	激しい振動 (大きな導関数) などにより射撃点が追加できなかった.	
4500	初期値問題計算中きざみ幅が小さくなりすぎた (特異点の存在など).	
5000	射撃点の最大個数に到達した.	
5500	最大反復回数に到達した. (解が存在しない場合や不定の場合を含む)	その時点での解を返して処理を打ち切る.

(6) **注意事項**

(a) 高階常微分方程式の非線形問題

$$y^{(m)} = f(x, y, y', \cdots, y^{(j)}, \cdots, y^{(m-1)})$$

(ただし, 右辺の $y^{(j)}$ の微分階数 j は $j \leq m-1$ を満たさなければならない.) において, 方程式の右辺の各項に変数 $y^{(j)}$ の非線形 (たとえば $y \cdot y'$, $\frac{y}{y'}$, y^2 などの 2 つ以上の乗除算や, $\sin(y)$, abs(y') などの関数項) がある場合は, この部分に alf を乗じることによってこの非線形問題をパラメータ化する.

例

$$y'' = y' + y^2$$

この非線形問題は次のようにパラメータ化し, isw=1 として解く.

$$y'' = y' + \underline{\mathrm{alf} \cdot y^2}$$

なお, 線形問題はパラメータ化する必要がなく isw=0 として解く.

(b) x, y の関数として微分方程式を定義する関数 f(x, y, m, alf) の実際の名前は, 使用者側のプログラムで宣言し, 実際の関数を作成しておかなければならない. 注意事項 (a) に述べた高階常微分方程式に対応する関数 f(x, y, m, alf)(倍精度) の作り方は, 次に示すとおりである.

```
void FORTRAN f(double *x, double *y, int *m, double *alf)
{
  y[*m - 1] = f(*alf, x, y, ..., y^(j), ..., y^(m-1))
}
```

ただし, $x \leftrightarrow$ *x, $m \leftrightarrow$ *m, $y \leftrightarrow$ y[0], $y^{(j)} \leftrightarrow$ y[j]
と対応する. なお, $f(*\mathrm{alf}, x, \cdots)$ は $f(x, \cdots)$ をパラメータ化した式である.

例

$$y'' = y' + \underline{\mathrm{alf} \cdot y^2}$$

パラメータ化した式が上のような場合関数はつぎのようになる

```
void FORTRAN f(double *x, double *y, int *m, double *alf)
{
   y[2] = y[1] + (*alf) * y[0] * y[0];
}
```

入力引数は m=2, isw=1 となる.

(c) 高階常微分方程式の境界条件が

$$\text{左側境界} \begin{cases} y^{(b_1)} = c_1 \\ y^{(b_2)} = c_2 \\ \quad \vdots \\ y^{(b_p)} = c_p \end{cases} \qquad \text{右側境界} \begin{cases} y^{(b_{p+1})} = c_{p+1} \\ y^{(b_{p+2})} = c_{p+2} \\ \quad \vdots \\ y^{(b_m)} = c_m \end{cases}$$

で与えられているとき, 配列 in, ic, bn には次のような値を設定する.
$\mathrm{in}[i-1] = 0 \ (i = 1, 2, \cdots, p)$
$\mathrm{in}[i-1] = 1 \ (i = p+1, p+2, \cdots, m)$
$\mathrm{ic}[i-1] = b_i (i = 1, 2, \cdots, m)$
$\mathrm{bn}[i-1] = c_i (i = 1, 2, \cdots, m)$

例

境界条件が

$$\text{左側境界} \quad y = 1.0 \quad \text{右側境界} \quad y' = 2.0$$

で与えられているとき,
in [0] =0, ic [0] =0, bn [0] =1.0
in [1] =1, ic [1] =1, bn [1] =2.0

(7) 使用例

(a) 問 題

$$y'' = y' + y^2$$

の常微分方程式を次の境界条件で解く.

$$y|_{x=1.0} = 1.0, y'|_{x=2.0} = 2.0$$

(b) 入力データ

関数 f(x, y, m, alf) の関数名: f, xa=1.0, xb=2.0,

in [0] =0, ic [0] =0, bn [0] =1.0

in [1] =1, ic [1] =1, bn [1] =2.0

m=2, x, k=6, er, ea, nx, nev=0, isw=1

(c) 主プログラム

```
/*      C interface example for ASL_dohnnv */

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
void    f(double *x,double *y,int *m,double *alf)
#else
void    f(x,y,m,alf)
double *x;
double *y;
double *alf;
int    *m;
#endif
{
    y[2]= y[1]+(*alf)*(y[0]*y[0]);
}
#ifdef __cplusplus
}
#endif

int main()
{
    double ax;
    double bx;
    int *in;
    int *ic;
    double *bn;
    int m;
    double *x;
    int k;
    double epr;
    double epa;
    int nx;
    int nev;
    double *y;
    int isw;
    int *iwk;
    double *wk;
    int ierr;
    int i,j,nwk;
    FILE *fp;

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

    printf( "    *** ASL_dohnnv ***\n" );
    printf( "\n     ** Input **\n\n" );
    m=2;
    k=6;

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

    ic  = ( int * )malloc((size_t)( sizeof(int) * m ));
    if( ic  == NULL )
    {
```

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

bn  = ( double * )malloc((size_t)( sizeof(double) * m ));
if( bn  == NULL )
{
    printf( "no enough memory for array  bn\n" );
    return -1;
}
x  = ( double * )malloc((size_t)( sizeof(double) * k ));
if( x  == NULL )
{
    printf( "no enough memory for array  x\n" );
    return -1;
}
y  = ( double * )malloc((size_t)( sizeof(double) * (k*(m+1)) ));
if( y  == NULL )
{
    printf( "no enough memory for array y \n" );
    return -1;
}

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

fscanf( fp, "%lf", &ax );
fscanf( fp, "%lf", &bx );
printf( "\txa  = %8.3g\n", ax );
printf( "\txb  = %8.3g\n", bx );
printf( "\n\tBoundary Condition\n\n" );
for( i=0 ; i<m ; i++ )
{
    fscanf( fp, "%d", &in[i] );
}
for( i=0 ; i<m ; i++ )
{
    fscanf( fp, "%d", &ic[i] );
}
for( i=0 ; i<m ; i++ )
{
    fscanf( fp, "%lf", &bn[i] );
}
printf( "\t  Position   Order of y       Value\n" );
for( i=0 ; i<m ; i++ )
{
    printf( "\t%6d     %6d         %8.3g\n",in[i],ic[i],bn[i]);
}
printf( "\n\tOrder of Differential Equation = %6d\n",m);
printf( "\n\tPoints Where Approximate Values are Computed\n\n" );
for( i=0 ; i<k ; i++ )
{
    fscanf( fp, "%lf", &x[i] );
    printf( "\t  x[%6d] = %8.3g\n",i,x[i]);
}
printf( "\n\tNumber of Points Where Approximate Values are Computed = %6d\n",k );

fscanf( fp, "%lf", &epr );
fscanf( fp, "%lf", &epa );
fscanf( fp, "%d", &nx );
fscanf( fp, "%d", &nev );
fscanf( fp, "%d", &isw );

nwk=(m+1)*(m+1)*(nx+1)+m*( (2*m+1>16) ? 2*m+1 : 16 );
wk  = ( double * )malloc((size_t)( sizeof(double) * nwk ));
if( wk  == NULL )
{
    printf( "no enough memory for array wk \n" );
    return -1;
}


printf( "\tRequired Local Relative Precision = %8.3g\n", epr );
printf( "\tRequired Local Absolute Precision = %8.3g\n", epa);
printf( "\tMaximum Number of Shooting Points =   %6d\n", nx );
printf( "\tMaximum Iterative Number = %6d\n", nev );
printf( "\tisw = %6d\n", isw );
printf( "\n" );

fclose( fp );

ierr = ASL_dohnnv(f, ax, bx, in, ic, bn, m, x, k, epr, epa, &nx, &nev, y, isw, iwk, wk);

printf( "\n     ** Output **\n\n" );
printf( "\tierr = %6d\n", ierr );
printf( "\n\tPractical Number of Shooting Points = %6d\n", nx );
printf( "\tPractical Iterative Number = %6d\n", nev );
printf( "\n\tApproximate Values\n\n" );
for( i=0 ; i<k ; i++ )
{
```

```
            for( j=0 ; j<m+1 ; j++ )
            {
            printf( "\t  y[%6d][%6d] = %8.3g\n",i,j,y[i+k*j] );
            }
        }

        free( in );
        free( ic );
        free( bn );
        free( x );
        free( y );
        free( iwk );
        free( wk );

        return 0;
    }
```

(d) 出力結果

```
        *** ASL_dohnnv ***

         ** Input **

        xa  =        1
        xb  =        2

        Boundary Condition

          Position   Order of y       Value
             0          0                1
             1          1                2

        Order of Differential Equation =      2

        Points Where Approximate Values are Computed

          x[     0] =        1
          x[     1] =      1.2
          x[     2] =      1.4
          x[     3] =      1.6
          x[     4] =      1.8
          x[     5] =        2

        Number of Points Where Approximate Values are Computed =      6
        Required Local Relative Precision =        0
        Required Local Absolute Precision =        0
        Maximum Number of Shooting Points =       10
        Maximum Iterative Number =      0
        isw =      1


         ** Output **

        ierr =      0

        Practical Number of Shooting Points =      6
        Practical Iterative Number =     12

        Approximate Values

          y[     0][     0] =        1
          y[     0][     1] =  -0.0831
          y[     0][     2] =    0.917
          y[     1][     0] =        1
          y[     1][     1] =    0.119
          y[     1][     2] =     1.12
          y[     2][     0] =     1.05
          y[     2][     1] =    0.377
          y[     2][     2] =     1.48
          y[     3][     0] =     1.16
          y[     3][     1] =    0.727
          y[     3][     2] =     2.07
          y[     4][     0] =     1.35
          y[     4][     1] =     1.23
          y[     4][     2] =     3.06
          y[     5][     0] =     1.67
          y[     5][     1] =        2
          y[     5][     2] =     4.79
```

2.3.6 ASL_dohnnf, ASL_rohnnf
高階常微分方程式 (関数境界)

(1) **機 能**

境界条件を関数として与えるものとして高階常微分方程式境界値問題を多点射撃法で解く.

(2) **使用法**

倍精度関数:

ierr = ASL_dohnnf (f, fb, xa, xb, m, x, k, er, ea, & nx, & nev, y, isw, iwk, wk);

単精度関数:

ierr = ASL_rohnnf (f, fb, xa, xb, m, x, k, er, ea, & nx, & nev, 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	f	–	–	入 力	x, y の関数として微分方程式を定義する関数 f(x, y, m, alf) の関数名
2	fb	–	–	入 力	境界条件を定義する関数 fb(ya, yb, m, g) の関数名
3	xa	{D / R}	1	入 力	左側境界の x 座標
4	xb	{D / R}	1	入 力	右側境界の x 座標
5	m	I	1	入 力	微分方程式に現れる変数 y の最大微分階数
6	x	{D∗ / R∗}	k	入 力	解を計算する点の x 座標 x_i
7	k	I	1	入 力	計算点の数
8	er	{D / R}	1	入 力	要求局所相対精度 既定値: 倍精度: 10^{-12}, 単精度: 10^{-5}
9	ea	{D / R}	1	入 力	要求局所絶対精度 (既定値: 誤差判定のための単位)
10	nx	I*	1	入 力	射撃点の最大個数
				出 力	射撃点の実際の個数
11	nev	I*	1	入 力	最大反復回数 (既定値:100)
				出 力	実際の反復回数
12	y	{D∗ / R∗}	k×(m+1)	出 力	解 $y^{(v)}(x_i)$ $y[(i-1)+k\times v] = y^{(v)}(x_i)$
13	isw	I	1	入 力	パラメータ化処理スイッチ 0:パラメータ化しない 0 以外:パラメータ化する
14	iwk	I*	m	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**: $(m+1)^2\times(nx+1)+m\times\max(2\times m+1, 16)$
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) xa $<$ xb

(b) er$\geq e_r$. ここで e_r= 倍精度: 10^{-14}, 単精度: 10^{-5} (既定値にするため, 0.0 を入力する場合は除く)

(c) ea$\geq$ 表現できる絶対値最小値 $\times 2^{24}$ (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)

(e) m ≥ 1

(f) k ≥ 1

(g) xa $\leq$ x[i $-$ 1] $\leq$ xb (i $= 1, 2, \cdots,$ k)

(h) nx $\geq \min(5\mathrm{i} + 1, 51)$
ここで i は xb $-$ xa $\leq$ i となる最小の整数

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	制限条件 (a) を満足しなかった. (ただし, xa $\neq$ xb)	xa と xb を互いに置き換えたものとして処理する.
1500	制限条件 (b), (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	xa=xb	処理を打ち切る.
3010	制限条件 (e) を満足しなかった.	
3020	制限条件 (f) を満足しなかった.	
3030	制限条件 (g) を満足しなかった.	
3040	制限条件 (h) を満足しなかった.	
4000	激しい振動 (大きな導関数) などにより射撃点が追加できなかった.	
4500	初期値問題計算中きざみ幅が小さくなりすぎた (特異点の存在など).	処理を打ち切る.
5000	射撃点の最大個数に到達した.	
5500	最大反復回数に到達した. (解が存在しない場合も含む).	その時点での解を返して処理を打ち切る.

(6) **注意事項**

(a) 高階常微分方程式の非線形問題

$$y^{(m)} = f(x, y, y', \cdots, y^{(j)}, \cdots, y^{(m-1)})$$

(ただし, 右辺の $y^{(j)}$ の微分階数 j は $j \leq m-1$ を満たさなければならない.) において, 方程式の右辺の各項に変数 $y^{(j)}$ の非線形 (たとえば $y \cdot y'$, $\frac{y}{y'}$, y^2 などの 2 つ以上の乗除算や, $\sin(y)$, $\mathrm{abs}(y')$ などの関数項) がある場合は, この部分に alf を乗じることによってこの非線形問題をパラメータ化する.

例

$$y'' = y' + y^2$$

この非線形問題は次のようにパラメータ化し, isw=1 として解く.

$$y'' = y' + \underline{\mathrm{alf} \cdot y^2}$$

なお, 線形問題はパラメータ化する必要がなく isw=0 として解く.

(b) x, y の関数として微分方程式を定義する関数 f(x, y, m, alf) の実際の名前は, 使用者側のプログラムで宣言し, 実際の関数を作成しておかなければならない. 注意事項 (a) に述べた高階常微分方程式に対応する関数 f(x, y, m, alf)(倍精度) の作り方は, 次に示すとおりである.

void FORTRAN f(double ∗x, double ∗y, int ∗m, double ∗alf)
{
$\quad$ y[∗m − 1] = $f(*\mathrm{alf}, x, y, \cdots, y^{(j)}, \cdots, y^{(m-1)})$
}

ただし, $x \leftrightarrow$ ∗x, $m \leftrightarrow$ ∗m, $y \leftrightarrow$ y[0], $y^{(j)} \leftrightarrow$ y[j]
と対応する. なお, $f(*\mathrm{alf}, x, \cdots)$ は $f(x, \cdots)$ をパラメータ化した式である.

例

$$y'' = y' + \underline{\mathrm{alf} \cdot y^2}$$

パラメータ化した式が上のような場合関数はつぎのようになる

void FORTRAN f(double ∗x, double ∗y, int ∗m, double ∗alf)
{
$\quad$ y[2] = y[1] + (∗alf) ∗ y[0] ∗ y[0];
}

入力引数は m=2, isw=1 となる.

(c) 境界条件を定義する関数 fb(ya, yb, m, g) の実際の名前は, 使用者側のプログラムで宣言し, 実際の関数を作成しておかなければならない. いま, m 個の境界条件が左側境界 $x = a$ での $y^{(j)}$ の値 $y^{(j)}(a)$ と右側境界 $x = b$ での $y^{(j)}$ の値 $y^{(j)}(b)$ の関数として

$$\begin{cases} g_1(y(a), \cdots, y^{(j)}(a), \cdots, y^{(m-1)}(a), y(b), \cdots, y^{(j)}(b), \cdots, y^{(m-1)}(b)) = 0 \\ g_2(y(a), \cdots, y^{(j)}(a), \cdots, y^{(m-1)}(a), y(b), \cdots, y^{(j)}(b), \cdots, y^{(m-1)}(b)) = 0 \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ g_m(y(a), \cdots, y^{(j)}(a), \cdots, y^{(m-1)}(a), y(b), \cdots, y^{(j)}(b), \cdots, y^{(m-1)}(b)) = 0 \end{cases}$$

と与えられているとすると関数 fb(ya, yb, m, g) の作り方 (倍精度) は, 次に示すとおりである.

void FORTRAN fb(double ∗ya, double ∗yb, int ∗m, double ∗g)
{
$\quad$ g[0] = $g_1(y(a), \cdots, y^{(j)}(a), \cdots, y^{(m-1)}(a), y(b), \cdots, y^{(j)}(b), \cdots, y^{(m-1)}(b))$;
$\quad$ g[1] = $g_2(y(a), \cdots, y^{(j)}(a), \cdots, y^{(m-1)}(a), y(b), \cdots, y^{(j)}(b), \cdots, y^{(m-1)}(b))$;
$\quad$ ⋮
$\quad$ g[$m-1$] = $g_m(y(a), \cdots, y^{(j)}(a), \cdots, y^{(m-1)}(a), y(b), \cdots, y^{(j)}(b), \cdots, y^{(m-1)}(b))$;
}

ただし,
$m \leftrightarrow$*m, $y(a) \leftrightarrow$ ya[0], $y^{(j)}(a) \leftrightarrow$ ya[j]
$y(b) \leftrightarrow$ yb[0], $y^{(j)}(b) \leftrightarrow$ yb[j]
と対応する.

例

境界条件が

$$\begin{cases} y(a) - y(b) = 0.0 \\ y'(b) = 1.0 \end{cases}$$

で与えられているとき, 関数 fb はつぎのようになる.

```
void FORTRAN fb(double *ya, double *yb, int *m, double *g)
{
    g[0] = ya[0] - yb[0];
    g[1] = yb[1] - 1.0;
}
```

(7) **使用例**

(a) 問 題

$$y'' = y' + y^2$$

の常微分方程式をを次の境界条件で解く.

$$\begin{cases} y(1.0) - y(2.0) = 0.0 \\ y(2.0) = 2.0 \end{cases}$$

(b) 入力データ

関数 f(x, y, m, alf) の関数名: f1,

関数 fb(ya, yb, m, g) の関数名: f2,

xa=1.0, xb=2.0, m=2, x, k=3, er, ea, nx, nev=0, isw=1

(c) 主プログラム

```
/*      C interface example for ASL_dohnnf */

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
void    f1(double *x,double *y,int *m,double *alf)
#else
void    f1(x,y,m,alf)
double *x;
double *y;
double *alf;
int    *m;
#endif
{
    y[2]= y[1]+(*alf)*(y[0]*y[0]);
}
#ifdef __cplusplus
}
#endif

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
void    f2(double *ya,double *yb,int *m,double *g)
#else
void    f2(ya,yb,m,g)
double *ya;
double *yb;
double *g;
int    *m;
#endif
{
    g[0]= ya[0]-yb[0];
    g[1]= yb[0]-2.0;
}
#ifdef __cplusplus
}
```

```
#endif


int main()
{
    double ax;
    double bx;
    int m;
    double *x;
    int k;
    double epr;
    double epa;
    int nx;
    int nev;
    double *y;
    int isw;
    int *iwk;
    double *wk;
    int ierr;
    int i,j,nwk;
    FILE *fp;

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

    printf( "    *** ASL_dohnnf ***\n" );
    printf( "\n     ** Input **\n\n" );
    m=2;
    k=3;

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

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

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

    fscanf( fp, "%lf", &ax );
    fscanf( fp, "%lf", &bx );
    printf( "\txa  =%8.3g\n", ax );
    printf( "\txb  =%8.3g\n", bx );
    printf( "\n\tBoundary Conditions\n" );
    printf( "\t  ya-yb = 0.0 \n" );
    printf( "\t  yb    = 2.0    (ya=y(x=1.0),yb=y(x=2.0))\n" );
    printf( "\n\tOrder of Differential Equation = %6d\n",m);
    printf( "\n\tPoints Where Approximate Values are Computed\n\n" );
    for( i=0 ; i<k ; i++ )
    {
        fscanf( fp, "%lf", &x[i] );
        printf( "\t  x[%6d] = %8.3g\n",i,x[i]);
    }
    printf( "\n\tNumber of Points Where Approximate Values are Computed = %6d\n",k );

    fscanf( fp, "%lf", &epr );
    fscanf( fp, "%lf", &epa );
    fscanf( fp, "%d", &nx );
    fscanf( fp, "%d", &nev );
    fscanf( fp, "%d", &isw );

    nwk=(m+1)*(m+1)*(nx+1)+m*( (2*m+1>16) ? 2*m+1 : 16 );
    wk  = ( double * )malloc((size_t)( sizeof(double) * nwk ));
    if( wk  == NULL )
    {
        printf( "no enough memory for array wk \n" );
        return -1;
    }


    printf( "\n\tRequired Local Relative Precision = %8.3g\n", epr );
    printf( "\tRequired Local Absolute Precision = %8.3g\n", epa);
    printf( "\tMaximum Number of Shooting Points =   %6d\n", nx );
    printf( "\tMaximum Iterative Number = %6d\n", nev );
    printf( "\tisw = %6d\n", isw );

    fclose( fp );

    ierr = ASL_dohnnf(f1, f2, ax, bx, m, x, k, epr, epa, &nx, &nev, y, isw, iwk, wk);
```

```
        printf( "\n     ** Output **\n\n" );
        printf( "\tierr = %6d\n", ierr );
        printf( "\n\tPractical Number of Shooting Points = %6d\n", nx );
        printf( "\tPractical Iterative Number = %6d\n", nev );
        printf( "\n\tApproximate Values\n\n" );
        for( i=0 ; i<k ; i++ )
        {
            for( j=0 ; j<m+1 ; j++ )
            {
            printf( "\t  y[%6d][%6d] = %8.3g\n",i,j,y[i+k*j] );
            }
        }

        free( x );
        free( y );
        free( iwk );
        free( wk );

        return 0;
    }
```

(d) 出力結果

```
        *** ASL_dohnnf ***

         ** Input **

        xa  =       1
        xb  =       2

        Boundary Conditions
          ya-yb = 0.0
          yb    = 2.0    (ya=y(x=1.0),yb=y(x=2.0))

        Order of Differential Equation =      2

        Points Where Approximate Values are Computed

          x[     0] =        1
          x[     1] =      1.5
          x[     2] =        2

        Number of Points Where Approximate Values are Computed =      3

        Required Local Relative Precision =        0
        Required Local Absolute Precision =        0
        Maximum Number of Shooting Points =       50
        Maximum Iterative Number =      0
        isw =      1

         ** Output **

        ierr =      0

        Practical Number of Shooting Points =      6
        Practical Iterative Number =     11

        Approximate Values

          y[     0][     0] =        2
          y[     0][     1] =    -1.32
          y[     0][     2] =     2.68
          y[     1][     0] =     1.64
          y[     1][     1] =   -0.104
          y[     1][     2] =      2.6
          y[     2][     0] =        2
          y[     2][     1] =     1.78
          y[     2][     2] =     5.78
```

2.3.7 ASL_dohnlv, ASL_rohnlv
線形高階常微分方程式

(1) **機 能**

境界条件を入力値として与えるものとして線形高階常微分方程式境界値問題を選点法で解く.

(2) **使用法**

倍精度関数:

ierr = ASL_dohnlv (f, xa, xb, in, ic, bn, m, x, k, er, ea, & nx, y, iwk, wk);

単精度関数:

ierr = ASL_rohnlv (f, xa, xb, in, ic, bn, m, x, k, er, ea, & nx, 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　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	−	−	入 力	線形高階常微分方程式の係数および定数項を定義する関数 f(x, al, m) の関数名
2	xa	{D / R}	1	入 力	左側境界の x 座標
3	xb	{D / R}	1	入 力	右側境界の x 座標
4	in	I*	m	入 力	境界条件の設定位置 (xa 側: 0, xb 側: 0 以外)
5	ic	I*	m	入 力	境界条件の値を与える変数 $y^{(j)}$ の微分階数 j
6	bn	{D*/ R*}	m	入 力	変数 $y^{(j)}$ に与える境界条件の値
7	m	I	1	入 力	微分方程式に現れる変数 y の最大微分階数
8	x	{D*/ R*}	k	入 力	解を計算する点の x 座標 x_i
9	k	I	1	入 力	計算点の数
10	er	{D / R}	1	入 力	要求局所相対精度 既定値: 倍精度: 10^{-12}, 単精度: 10^{-5}
11	ea	{D / R}	1	入 力	要求絶対精度 (既定値: 誤差判定のための単位)
12	nx	I*	1	入 力	選点の最大個数
				出 力	選点の実際の個数
13	y	{D*/ R*}	k×(m+1)	出 力	解 $y^{(v)}(x_i)$ $\mathrm{y}[(\mathrm{i}-1)+\mathrm{k}\times\mathrm{v}] = y^{(v)}(x_i)$
14	iwk	I*	3×m+nx	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**: $2\times\mathrm{nx}\times\mathrm{nx}+5\times\mathrm{nx}+2\times\mathrm{m}+2$
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) xa $<$ xb

(b) er$\geq e_r$. ここで e_r= 倍精度: 10^{-14}, 単精度: 10^{-5} (既定値にするため, 0.0 を入力する場合は除く)

(c) ea≥(表現できる絶対値最小値)$\times 2^{24}$ (既定値にするため 0.0 を入力する場合は除く)

(d) m ≥ 0

(e) $0 \leq$ ic[i $-$ 1] $<$ m (i $= 1, 2, \cdots,$ m)

(f) k ≥ 1

(g) xa $\leq$ x[i $-$ 1] $\leq$ xb (i $= 1, 2, \cdots,$ k)

(h) nx $\geq \max(\min(5 \times$ i $+ 1, 101),$ m $+ 2)$
ここで i は xb $-$ xa $\leq$ i となる最小の整数

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	制限条件 (a) を満足しなかった. (ただし, xa $\neq$ xb)	xa と xb を互いに置き換えたものとして処理する.
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	xa=xb	処理を打ち切る.
3010	制限条件 (d) を満足しなかった.	
3020	制限条件 (e) を満足しなかった.	
3030	制限条件 (f) を満足しなかった.	
3040	制限条件 (g) を満足しなかった.	
3050	制限条件 (h) を満足しなかった.	
4000	連立 1 次方程式が解けなかった.	
5000	選点の最大個数に到達した (解なしの場合を含む).	その時点での解を返して処理を打ち切る.

(6) **注意事項**

(a) x, y の関数として微分方程式を定義する関数 f(x, al, m) の実際の名前は, 使用者側のプログラムで宣言し, 実際の関数を作成しておかなければならない. 線形高階常微分方程式

$$f_1(x)y^{(m)} + f_2(x)y^{(m-1)} + \cdots + f_{m+1}(x)y + f_{m+2}(x) = 0$$

に対応する関数 f(x, al, m)(倍精度) の作り方は, 次に示すとおりである.

```
void FORTRAN f(double *x, double *al, int *m)
{
  al[0] = f1(x);
  al[1] = f2(x);
  ⋮
  al[m + 1] = fm+2(x);
}
```

ただし, $x \leftrightarrow$ *x, $m \leftrightarrow$ *m と対応する.

例

$$y'' + 2xy + x = 0$$

```
void FORTRAN f(double *x, double *al, int *m)
{
   al[0] = 1.0;
   al[1] = 0.0;
   al[2] = 2 * (*x);
   al[3] = (*x);
}
```

入力引数は m=2 とする.

(b) 線形高階常微分方程式の境界条件が

$$\text{左側境界} \begin{cases} y^{(b_1)} = c_1 \\ y^{(b_2)} = c_2 \\ \quad \vdots \\ y^{(b_p)} = c_p \end{cases} \qquad \text{右側境界} \begin{cases} y^{(b_{p+1})} = c_{p+1} \\ y^{(b_{p+2})} = c_{p+2} \\ \quad \vdots \\ y^{(b_m)} = c_m \end{cases}$$

で与えられているとき, 配列 in, ic, bn には次のような値を設定する.

in$[i-1] = 0$ $(i = 1, 2, \cdots, p)$

in$[i-1] = 1$ $(i = p+1, p+2, \cdots, m)$

ic$[i-1] = b_i (i = 1, 2, \cdots, m)$

bn$[i-1] = c_i (i = 1, 2, \cdots, m)$

例

境界条件が

$$\text{左側境界} \quad y = 1.0 \quad \text{右側境界} \quad y' = 2.0$$

で与えられているとき,

in [0] =0, ic [0] =0, bn [0] =1.0

in [1] =1, ic [1] =1, bn [1] =2.0

(7) 使用例

(a) 問 題

$$y'' + 2xy + x = 0$$

の常微分方程式を次の境界条件で解く.

$$y|_{x=0.0113} = 10.0, y'|_{x=0.0188} = 12.0$$

(b) 入力データ

関数 f(x, al, m) の関数名: f, xa=0.0113, xb=0.0188,

in [0] =0, ic [0] =0, bn [0] =10.0

in [1] =1, ic [1] =1, bn [1] =20.0

m=2, x, k=6, er, ea, nx

(c) 主プログラム

```
/*      C interface example for ASL_dohnlv */

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
void    f(double *x,double *al,int *m)
#else
void    f(x,al,m)
double *x;
double *al;
int    *m;
#endif
{
    al[0]= 1.0;
    al[1]= 0.0;
    al[2]= 2.0*(*x);
    al[3]= (*x);
}
#ifdef __cplusplus
}
#endif


int main()
{
    double ax;
    double bx;
    int *in;
    int *ic;
    double *bn;
    int m;
    double *x;
    int k;
    double epr;
    double epa;
    int nx;
    double *y;
    int *iwk;
    double *wk;
    int ierr;
    int i,j,nwk;
    FILE *fp;

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

    printf( "    *** ASL_dohnlv ***\n" );
    printf( "\n     ** Input **\n\n" );
    m=2;
    k=6;

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

    ic  = ( int * )malloc((size_t)( sizeof(int) * m ));
    if( ic  == NULL )
```

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

bn  = ( double * )malloc((size_t)( sizeof(double) * m ));
if( bn  == NULL )
{
    printf( "no enough memory for array  bn\n" );
    return -1;
}
x  = ( double * )malloc((size_t)( sizeof(double) * k ));
if( x  == NULL )
{
    printf( "no enough memory for array  x\n" );
    return -1;
}
y  = ( double * )malloc((size_t)( sizeof(double) * (k*(m+1)) ));
if( y  == NULL )
{
    printf( "no enough memory for array y \n" );
    return -1;
}

fscanf( fp, "%lf", &ax );
fscanf( fp, "%lf", &bx );
printf( "\txa  = %8.3g\n", ax );
printf( "\txb  = %8.3g\n", bx );
printf( "\n\tBoundary Condition\n\n" );
for( i=0 ; i<m ; i++ )
{
    fscanf( fp, "%d", &in[i] );
}
for( i=0 ; i<m ; i++ )
{
    fscanf( fp, "%d", &ic[i] );
}
for( i=0 ; i<m ; i++ )
{
    fscanf( fp, "%lf", &bn[i] );
}
printf( "\t  Position   Order of y       Value\n" );
for( i=0 ; i<m ; i++ )
{
    printf( "\t%6d     %6d         %8.3g\n",in[i],ic[i],bn[i]);
}
printf( "\n\tOrder of Differential Equation = %6d\n",m);
printf( "\n\tPoints Where Approximate Values are Computed\n\n" );
for( i=0 ; i<k ; i++ )
{
    fscanf( fp, "%lf", &x[i] );
    printf( "\t  x[%6d] = %8.3g\n",i,x[i]);
}
printf( "\n\tNumber of Points Where Approximate Values are Computed = %6d\n",k );

fscanf( fp, "%lf", &epr );
fscanf( fp, "%lf", &epa );
fscanf( fp, "%d", &nx );

printf( "\n\tRequired Local Relative Precision  = %8.3g\n", epr );
printf( "\tRequired Local Absolute Precision  = %8.3g\n", epa);
printf( "\tMaximum Number of Selected Points  =   %6d\n", nx );

nwk=2*nx*nx+5*nx+2*m+2;
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) * (3*m+nx) ));
if( iwk  == NULL )
{
    printf( "no enough memory for array iwk\n" );
    return -1;
}

fclose( fp );

ierr = ASL_dohnlv(f, ax, bx, in, ic, bn, m, x, k, epr, epa, &nx, y, iwk, wk);

printf( "\n     ** Output **\n\n" );
printf( "\tierr = %6d\n", ierr );
printf( "\n\tPractical Number of Selected Points = %6d\n", nx );
printf( "\n\tApproximate Values\n\n" );
for( i=0 ; i<k ; i++ )
{
    for( j=0 ; j<m+1 ; j++ )
    {
    printf( "\t  y[%6d][%6d] = %8.3g\n",i,j,y[i+k*j] );
    }
}

free( in );
```

```
    free( ic );
    free( bn );
    free( x );
    free( y );
    free( iwk );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dohnlv ***

 ** Input **

xa  =     0.01
xb  =     0.02

Boundary Condition

  Position   Order of y       Value
     0          0               10
     1          0               12

Order of Differential Equation =      2

Points Where Approximate Values are Computed

  x[     0] =   0.0113
  x[     1] =   0.0143
  x[     2] =   0.0167
  x[     3] =   0.0171
  x[     4] =   0.0183
  x[     5] =   0.0188

Number of Points Where Approximate Values are Computed =      6

Required Local Relative Precision  =    1e-08
Required Local Absolute Precision  =    1e-08
Maximum Number of Selected Points  =      100

 ** Output **

ierr =      0

Practical Number of Selected Points =      6

Approximate Values

  y[     0][     0] =     10.3
  y[     0][     1] =      200
  y[     0][     2] =   -0.243
  y[     1][     0] =     10.9
  y[     1][     1] =      200
  y[     1][     2] =   -0.325
  y[     2][     0] =     11.3
  y[     2][     1] =      200
  y[     2][     2] =   -0.395
  y[     3][     0] =     11.4
  y[     3][     1] =      200
  y[     3][     2] =   -0.408
  y[     4][     0] =     11.7
  y[     4][     1] =      200
  y[     4][     2] =   -0.445
  y[     5][     0] =     11.8
  y[     5][     1] =      200
  y[     5][     2] =   -0.461
```

2.3.8 ASL_dolnlv, ASL_rolnlv
線形 2 階常微分方程式

(1) **機 能**

境界条件を入力値として与えるものとして線形 2 階常微分方程式境界値問題を係数決定法で解く.

(2) **使用法**

倍精度関数:

ierr = ASL_dolnlv (f, xa, xb, in, ic, bn, x, k, er, ea, y, wk);

単精度関数:

ierr = ASL_rolnlv (f, xa, xb, in, ic, bn, x, k, er, ea, y, 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	f	−	−	入 力	線形 2 階常微分方程式の係数および定数項を定義する関数 f(x, al) の関数名
2	xa	{ D / R }	1	入 力	左側境界の x 座標
3	xb	{ D / R }	1	入 力	右側境界の x 座標
4	in	I*	2	入 力	境界条件の設定位置 (xa 側 : 0, xb 側 : 0 以外)
5	ic	I*	2	入 力	境界条件の値を与える変数 $y^{(j)}$ の微分階数 j
6	bn	{ D* / R* }	2	入 力	変数 $y^{(j)}$ に与える境界条件の値
7	x	{ D* / R* }	k	入 力	解を計算する点の x 座標 x_i
8	k	I	1	入 力	計算点の数
9	er	{ D / R }	1	入 力	要求局所相対精度 既定値: 倍精度: 10^{-12}, 単精度: 10^{-5}
10	ea	{ D / R }	1	入 力	要求局所絶対精度 (既定値: 誤差判定のための単位)
11	y	{ D* / R* }	k × 3	出 力	解 $y^{(v)}(x_i)$ y[(i − 1) + k × v] = $y^{(v)}(x_i)$
12	wk	{ D* / R* }	6 × k + 35	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) xa $<$ xb

(b) er$\geq e_r$. ここで e_r= 倍精度: 10^{-14}, 単精度: 10^{-5} (既定値にするため, 0.0 を入力する場合は除く)

(c) ea≥(表現できる絶対値最小値)$\times 2^{24}$ (既定値にするため 0.0 を入力する場合は除く)

(d) $0 \leq$ ic[i $-$ 1] < 2 (i $= 1, 2$)

(e) k ≥ 1

(f) xa $\leq$ x[i $-$ 1] $\leq$ xb (i $= 1, 2, \cdots,$ k)

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	制限条件 (a) を満足しなかった. (ただし, xa $\neq$ xb)	xa と xb を互いに置き換えたものとして処理する.
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	xa=xb	処理を打ち切る.
3010	制限条件 (d) を満足しなかった.	
3020	制限条件 (e) を満足しなかった.	
3030	制限条件 (f) を満足しなかった.	
4000	初期値問題計算中きざみ幅が小さくなりすぎた.	
4500	解なし, または解不定の場合.	

(6) **注意事項**

(a) x, y の関数として微分方程式を定義する関数 f(x, al) の実際の名前は, 使用者側のプログラムで宣言し, 実際の関数を作成しておかなければならない. 線形 2 階常微分方程式

$$f_1(x)y'' + f_2(x)y' + f_3(x)y + f_4(x) = 0$$

に対応する関数 f(x, al) の作り方 (倍精度) は, 次に示すとおりである.

```
void FORTRAN f(double *x, double *al)
{
  al[0] = f1(x);
  al[1] = f2(x);
  al[2] = f3(x);
  al[3] = f4(x);
}
```

ただし, $x \leftrightarrow$ *x と対応する.

例

$$y'' + 2xy + x = 0$$

```
void FORTRAN f(double *x, double *al)
{
  al[0] = 1.0;
```

al[1] = 0.0;
al[2] = 2 ∗ (∗x);
al[3] = (∗x);
}

(b) 線形 2 階常微分方程式の境界条件が

左側境界　$y^{(b_1)} = c_1$　右側境界　$y^{(b_2)} = c_2$

で与えられているとき, 配列 in, ic, bn には次のような値を設定する.

in[0] = 0
in[1] = 1
ic[$i-1$] = b_i　($i = 1, 2$)
bn[$i-1$] = c_i　($i = 1, 2$)

例

境界条件が

左側境界　$y = 1.0$　右側境界　$y' = 2.0$

で与えられているとき,
in [0] =0, ic [0] =0, bn [0] =1.0
in [1] =1, ic [1] =1, bn [1] =2.0

(7) **使用例**

(a) 問 題

$$y'' + 2xy + x = 0$$

の常微分方程式を次の境界条件で解く.

$$y|_{x=0.0} = 1.0, y'|_{x=1.0} = 2.0$$

(b) 入力データ
関数 f(x, al) の関数名: f, xa=0.0, xb=1.0,
in [0] =0, ic [0] =0, bn [0] =1.0
in [1] =1, ic [1] =1, bn [1] =2.0
x, k=6, er, ea

(c) 主プログラム

```
/*      C interface example for ASL_dolnlv */

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
void    f(double *x,double *al)
#else
void    f(x,al)
double *x;
double *al;
#endif
{
    al[0]= 1.0;
    al[1]= 0.0;
    al[2]= 2.0*(*x);
    al[3]= (*x);
}
#ifdef __cplusplus
}
#endif
```

```
int main()
{
    double ax;
    double bx;
    int in[2];
    int ic[2];
    double bn[2];
    double *x;
    int k;
    double epr;
    double epa;
    double *y;
    double *wk;
    int ierr;
    int i,j;
    FILE *fp;

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

    printf( "    *** ASL_dolnlv ***\n" );
    printf( "\n     ** Input **\n\n" );
    k=6;

    x  = ( double * )malloc((size_t)( sizeof(double) * k ));
    if( x  == NULL )
    {
        printf( "no enough memory for array  x\n" );
        return -1;
    }
    y  = ( double * )malloc((size_t)( sizeof(double) * (3*k) ));
    if( y  == NULL )
    {
        printf( "no enough memory for array y \n" );
        return -1;
    }

    wk  = ( double * )malloc((size_t)( sizeof(double) * (6*k+35) ));
    if( wk  == NULL )
    {
        printf( "no enough memory for array wk \n" );
        return -1;
    }

    fscanf( fp, "%lf", &ax );
    fscanf( fp, "%lf", &bx );
    printf( "\txa  = %8.3g\n", ax );
    printf( "\txb  = %8.3g\n", bx );
    printf( "\n\tBoundary Condition\n\n" );
    for( i=0 ; i<2 ; i++ )
    {
        fscanf( fp, "%d", &in[i] );
    }
    for( i=0 ; i<2 ; i++ )
    {
        fscanf( fp, "%d", &ic[i] );
    }
    for( i=0 ; i<2 ; i++ )
    {
        fscanf( fp, "%lf", &bn[i] );
    }
    printf( "\t  Position   Order of y       Value\n" );
    for( i=0 ; i<2 ; i++ )
    {
        printf( "\t%6d     %6d         %8.3g\n",in[i],ic[i],bn[i]);
    }
    printf( "\n\tOrder of Differential Equation = 2\n");
    printf( "\n\tPoints Where Approximate Values are Computed\n\n" );
    for( i=0 ; i<k ; i++ )
    {
        fscanf( fp, "%lf", &x[i] );
        printf( "\t  x[%6d] = %8.3g\n",i,x[i]);
    }
    printf( "\n\tNumber of Points Where Approximate Values are Computed = %6d\n",k );

    fscanf( fp, "%lf", &epr );
    fscanf( fp, "%lf", &epa );

    printf( "\tRequired Local Relative Precision = %8.3g\n", epr );
    printf( "\tRequired Local Absolute Precision = %8.3g\n", epa);

    fclose( fp );

    ierr = ASL_dolnlv(f, ax, bx, in, ic, bn, x, k, epr, epa, y, wk);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tApproximate Values\n\n" );
    for( i=0 ; i<k ; i++ )
    {
        for( j=0 ; j<3 ; j++ )
```

```
            {
            printf( "\t  y[%6d][%6d] = %8.3g\n",i,j,y[i+k*j] );
            }
        }

        free( x );
        free( y );
        free( wk );

        return 0;
    }
```

(d) 出力結果

```
    *** ASL_dolnlv ***

     ** Input **

    xa  =        0
    xb  =        1

    Boundary Condition

      Position   Order of y       Value
         0          0               1
         1          1               2

    Order of Differential Equation = 2

    Points Where Approximate Values are Computed

      x[     0] =        0
      x[     1] =      0.2
      x[     2] =      0.4
      x[     3] =      0.6
      x[     4] =      0.8
      x[     5] =        1

    Number of Points Where Approximate Values are Computed =      6
    Required Local Relative Precision =        0
    Required Local Absolute Precision =        0

     ** Output **

    ierr =      0

    Approximate Values

      y[     0][     0] =        1
      y[     0][     1] =     8.54
      y[     0][     2] =        0
      y[     1][     0] =      2.7
      y[     1][     1] =     8.44
      y[     1][     2] =    -1.28
      y[     2][     0] =     4.35
      y[     2][     1] =     7.94
      y[     2][     2] =    -3.88
      y[     3][     0] =     5.84
      y[     3][     1] =     6.81
      y[     3][     2] =    -7.61
      y[     4][     0] =     7.02
      y[     4][     1] =     4.85
      y[     4][     2] =      -12
      y[     5][     0] =     7.72
      y[     5][     1] =        2
      y[     5][     2] =    -16.4
```

2.4 積分方程式

2.4.1 ASL_doief2, ASL_roief2
第 2 種フレドホルム型積分方程式

(1) **機 能**

第 2 種フレドホルム型積分方程式

$$y(t) - \int_a^b K(t,x)y(x)dx = f(t)$$

をガウスの積分公式を用いて解き, 3 次スプライン関数を用いた補間により

$$-0.4984|a-b| + \frac{a+b}{2} \le t_i \le 0.4984|a-b| + \frac{a+b}{2} \quad (i = 1, 2, \cdots, n)$$

の範囲内の任意の $t = t_i$ に対する $y(t)$ の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_doief2 (ff, fk, xa, xb, ti, n, y);

単精度関数:

ierr = ASL_roief2 (ff, fk, xa, xb, ti, n, 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	ff	{D / R}	−	入 力	既知関数 $f(t)$ を定義する関数 ff(t) の関数名
2	fk	{D / R}	−	入 力	正則核 $K(t,x)$ を定義する関数 fk(t, x) の関数名
3	xa	{D / R}	1	入 力	積分区間の下限 a
4	xb	{D / R}	1	入 力	積分区間の上限 b
5	ti	{D∗ / R∗}	n	入 力	近似解を求める点 t_i
6	n	I	1	入 力	計算点の数 n
7	y	{D∗ / R∗}	n	出 力	近似解 $y(t_i)$
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) xa≠xb

(b) n>1

(c) $-0.4984|\mathrm{xa}-\mathrm{xb}|+\dfrac{\mathrm{xa}+\mathrm{xb}}{2} \leq \mathrm{ti}[\mathrm{i}-1] \leq 0.4984|\mathrm{xa}-\mathrm{xb}|+\dfrac{\mathrm{xa}+\mathrm{xb}}{2}$ $(\mathrm{i}=1,\cdots,\mathrm{n})$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	制限条件 (a) を満足しなかった.	$\mathrm{y}[\mathrm{i}-1]=\mathrm{ff}(\mathrm{ti}[\mathrm{i}-1])$ $(\mathrm{i}=1,\cdots,\mathrm{n})$
1100	制限条件 (c) を満足しなかった.	端点でのスプライン係数を利用し, 補外した値を出力する.
3000	制限条件 (b) を満足しなかった.	処理を打ち切る.
4000	連立 1 次方程式を解く時, エラーが発生した.	

(6) **注意事項**

(a) 関数 ff(t), fk(t, x) の実際の名前は, 使用者側のプログラムで宣言し, 実際の関数を作成しておかなければならない.
関数 (倍精度) の作り方は次のようになる.

```
double FORTRAN ff(double *t)
{
  return f(t);
}

double FORTRAN fk(double *t, double *x)
{
  return K(t,x);
}
```

ただし, $t \leftrightarrow$ *t, $x \leftrightarrow$ *x と対応する.

(7) 使用例

(a) 問 題

$$y(t) - \int_1^2 (x+t)y(x)dx = t$$

を x =1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9 について解く.

(b) 入力データ

ff(t) に対応する関数名:ff, fk(t, x) に対応する関数名:fk, xa=1.0, xb=2.0, ti, n=9

(c) 主プログラム

```
/*      C interface example for ASL_doief2 */

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
double ff(double *tt)
#else
double ff(tt)
double *tt;
#endif
{
    return *tt;
}
#ifdef __cplusplus
}
#endif
#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
double fk(double *tt,double *x)
#else
double fk(tt,x)
double *tt,*x;
#endif
{
    return (*x) + (*tt);
}
#ifdef __cplusplus
}
#endif
int main()
{
    double xa;
    double xb;
    double *t;
    double *y;
    int n;
    int ierr;
    int i;
    FILE *fp;

    fp = fopen( "doief2.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_doief2 ***\n" );
    printf( "\n     ** Input **\n\n" );
    fscanf( fp, "%lf", &xa );
    fscanf( fp, "%lf", &xb );
    fscanf( fp, "%d", &n );

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

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

    for( i=0 ; i<n ; i++ )
```

```
        {
            fscanf( fp, "%lf", &t[i] );
        }

        printf( "\txa   = %8.3g\n", xa );
        printf( "\txb   = %8.3g\n", xb );

        fclose( fp );

        ierr = ASL_doief2(ff, fk, xa, xb, t, n, y);

        printf( "\n     ** Output **\n\n" );
        printf( "\tierr = %6d\n", ierr );
        printf( "\n\tSolution\n" );
        printf( "\n\t i     x[i]     y[i]\n");
        for( i=0 ; i<n ;i++ )
        {
            printf( "\t%2d %8.3g %8.3g \n", i,t[i],y[i]);
        }

        free( t );
        free( y );

        return 0;
    }
```

(d) 出力結果

```
        *** ASL_doief2 ***

         ** Input **

        xa   =        1
        xb   =        2

         ** Output **

        ierr =      0

        Solution

         i     x[i]     y[i]
         0      1.1   -0.856
         1      1.2   -0.832
         2      1.3   -0.808
         3      1.4   -0.784
         4      1.5    -0.76
         5      1.6   -0.736
         6      1.7   -0.712
         7      1.8   -0.688
         8      1.9   -0.664
```

2.4.2 ASL_doiev1, ASL_roiev1
第 1 種ボルテラ型積分方程式

(1) **機 能**

第 1 種ボルテラ型積分方程式

$$\int_a^t K(t,x)y(x)dx = f(t)$$

をマクローリンの公式を用いて解き, 3 次スプライン関数を用いた補間により任意の $x = x_i \ \ (i = 1, 2, \cdots, n)$ における $y(x)$ の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_doiev1 (ff, fk, xa, xb, m, xi, n, y, w);

単精度関数:

ierr = ASL_roiev1 (ff, fk, xa, xb, m, xi, n, y, 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	ff	{D / R}	−	入 力	既知関数 $f(t)$ を定義する関数 ff(t) の関数名
2	fk	{D / R}	−	入 力	正則核 $K(t,x)$ を定義する関数 fk(t, x) の関数名
3	xa	{D / R}	1	入 力	積分区間の下限 a (注意事項 (b) 参照)
4	xb	{D / R}	1	入 力	近似解が計算される点の上限 (注意事項 (b) 参照)
5	m	I	1	入 力	積分区間の分割数 (注意事項 (b) 参照)
6	xi	{D∗ / R∗}	n	入 力	近似解が計算される点 x_i
7	n	I	1	入 力	計算点の数 n
8	y	{D∗ / R∗}	n	出 力	近似解 $y(x_i)$
9	w	{D∗ / R∗}	$5 \times \mathrm{m} + 2$	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>1

(b) m>1

(c) xa<xb

(d) $\mathrm{xa} + \dfrac{\mathrm{xb} - \mathrm{xa}}{2 \times \mathrm{m}} < \mathrm{xi[i-1]} < \mathrm{xb} \quad (\mathrm{i} = 1, \cdots, \mathrm{n})$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	制限条件 (d) を満足しなかった.	端点でのスプライン係数を利用し, 補外した値を出力する.
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 (c) を満足しなかった.	

(6) **注意事項**

(a) 関数 ff(t), fk(t, x) の実際の名前は, 使用者側のプログラムで宣言し, 実際の関数を作成しておかなければならない.
関数 (倍精度) の作り方は次のようになる.

double FORTRAN ff(double ∗t)
{
  return $f(t)$;
}

double FORTRAN fk(double ∗t, double ∗x)
{
  return $K(t, x)$;
}

ただし, $t \leftrightarrow$ ∗t, $x \leftrightarrow$ ∗x と対応する.

(b) 積分の計算区間を定める刻み幅 h には, $\dfrac{\mathrm{xb} - \mathrm{xa}}{\mathrm{m}}$ を使用する. (2.1.2 参照).

(7) **使用例**

(a) 問 題

$$t = \int_0^t (1 + t - x) y(x) dx$$

を x =0.025, 0.075, 0.125, 0.175, 0.225, 0.275, 0.325, 0.375, 0.425, 0.475, 0.500 について解く.

(b) 入力データ
ff(t) に対応する関数名:ff, fk(t, x) に対応する関数名:fk, xa=0.0, xb=0.5, xi, n=11, m=11

(c) 主プログラム

```
/*      C interface example for ASL_doiev1 */

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

```
#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
double ff(double *tt)
#else
double ff(tt)
double *tt;
#endif
{
    return *tt;
}
#ifdef __cplusplus
}
#endif
#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
double fk(double *tt, double *x)
#else
double fk(tt,x)
double *tt,*x;
#endif
{
    return 1 + (*tt) - (*x);
}
#ifdef __cplusplus
}
#endif
int main()
{
    double xa;
    double xb;
    double *t;
    double *y;
    double *w;
    int m;
    int n;
    int ierr;
    int i;
    FILE *fp;

    fp = fopen( "doiev1.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_doiev1 ***\n" );
    printf( "\n     ** Input **\n\n" );
    fscanf( fp, "%lf", &xa );
    fscanf( fp, "%lf", &xb );
    fscanf( fp, "%d", &m );
    fscanf( fp, "%d", &n );

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

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

    w = ( double * )malloc((size_t)( sizeof(double) * (5*m+2) ));
    if( w == NULL )
    {
        printf( "no enough memory for array w\n" );
        return -1;
    }
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &t[i] );
    }

    printf( "\txa   = %8.3g\n", xa );
    printf( "\txb   = %8.3g\n", xb );

    fclose( fp );

    ierr = ASL_doiev1(ff, fk, xa, xb, m, t, n, y, w);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tSolution\n" );
    printf( "\n\t i     x[i]     y[i]\n");
    for( i=0 ; i<n ;i++ )
```

```
        {
               printf( "\t%2d %8.3g %8.3g \n", i,t[i],y[i]);
        }

        free( t );
        free( y );
        free( w );

        return 0;
    }
```

(d) 出力結果

```
        *** ASL_doiev1 ***

         ** Input **

        xa   =        0
        xb   =      0.5

         ** Output **

        ierr =      0

        Solution

         i     x[i]     y[i]
         0    0.025    0.976
         1    0.075    0.928
         2    0.125    0.882
         3    0.175     0.84
         4    0.225    0.799
         5    0.275     0.76
         6    0.325    0.723
         7    0.375    0.687
         8    0.425    0.654
         9    0.475    0.622
        10      0.5    0.606
```

2.5 偏微分方程式

2.5.1 ASL_dopdh2, ASL_ropdh2
非同次 Helmholtz 方程式 (2 次元)

(1) **機 能**

与えられた矩形領域内において 2 次元 5 点差分近似を用いて非同次の Helmholtz 方程式,

$$u_{xx} + u_{yy} + \lambda u = f(x, y)$$

を解く.

(2) **使用法**

倍精度関数:

ierr = ASL_dopdh2 (dlmd, func, nx, ny, xl, xu, yl, yu, s1, s2, s3, s4, imax, eps, u, imx, isw, iw, w);

単精度関数:

ierr = ASL_ropdh2 (dlmd, func, nx, ny, xl, xu, yl, yu, s1, s2, s3, s4, imax, eps, u, imx, 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	dlmd	{D / R}	1	入 力	u の定数 λ
2	func	−	−	入 力	x,y の関数として微分方程式を定義する 関数 f(x,y) の関数名
3	nx	I	1	入 力	X 軸方向の分割数 n_x
4	ny	I	1	入 力	Y 軸方向の分割数 n_y
5	xl	{D / R}	1	入 力	領域の左側の X 座標 x_L
6	xu	{D / R}	1	入 力	領域の右側の X 座標 x_U
7	yl	{D / R}	1	入 力	領域の下側の Y 座標 y_L
8	yu	{D / R}	1	入 力	領域の上側の Y 座標 y_U
9	s1	{D∗ / R∗}	nx+1	入 力	下辺の Dirichlet 境界条件の値 (isw[0] $\neq 1$ の時) isw[0] $= 1$ の時は領域のみ確保.
10	s2	{D∗ / R∗}	ny+1	入 力	右辺の Dirichlet 境界条件の値 (isw[1] $\neq 1$ の時) isw[1] $= 1$ の時は領域のみ確保.

項番	引数と戻り値	型	大きさ	入出力	内　　容
11	s3	$\left\{\begin{matrix}\mathrm{D}* \\ \mathrm{R}*\end{matrix}\right\}$	nx+1	入 力	上辺の Dirichlet 境界条件の値 (isw[2] $\neq 1$ の時) isw[2] $= 1$ の時は領域のみ確保.
12	s4	$\left\{\begin{matrix}\mathrm{D}* \\ \mathrm{R}*\end{matrix}\right\}$	ny+1	入 力	左辺の Dirichlet 境界条件の値 (isw[3] $\neq 1$ の時) isw[3] $= 1$ の時は領域のみ確保.
13	imax	I	1	入 力	連立 1 次方程式を反復法で解く際の最大反復回数.
14	eps	$\left\{\begin{matrix}\mathrm{D} \\ \mathrm{R}\end{matrix}\right\}$	1	入 力	打ち切り残差ノルム $\parallel \boldsymbol{b} - A\boldsymbol{u} \parallel / \parallel \boldsymbol{b} \parallel$ **指定可能範囲:** $\geq$ アンダフロー判定値 (注意事項 (e) 参照)
15	u	$\left\{\begin{matrix}\mathrm{D}* \\ \mathrm{R}*\end{matrix}\right\}$	内容参照	出 力	解 $u(x_i, y_j)$ $\mathrm{u}(i,j) = u(x_L+i(x_U-x_L)/n_x, y_L+j(y_U-y_L)/n_y)$ **大きさ:** $(\mathrm{imx}+1) \times (\mathrm{ny}+1)$
16	imx	I	1	入 力	配列 u の整合寸法
17	isw	I*	4	入 力	境界条件選択スイッチ isw[0] $= 0$: 下辺に Dirichlet 条件を付加 $= 1$: 下辺に Neumann 条件を付加 isw[1] $= 0$: 右辺に Dirichlet 条件を付加 $= 1$: 右辺に Neumann 条件を付加 isw[2] $= 0$: 上辺に Dirichlet 条件を付加 $= 1$: 上辺に Neumann 条件を付加 isw[3] $= 0$: 左辺に Dirichlet 条件を付加 $= 1$: 左辺に Neumann 条件を付加
18	iw	I*	7	ワーク	作業領域
19	w	$\left\{\begin{matrix}\mathrm{D}* \\ \mathrm{R}*\end{matrix}\right\}$	内容参照	ワーク	作業領域 **大きさ:** $13 \times (\mathrm{nx}+1) \times (\mathrm{ny}+1) + 5$
20	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) nx ≥ 2, ny ≥ 2

(b) 各辺に付加する境界条件は, 少なくとも 1 辺を Dirichlet 条件としなければならない.

(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.	
1400	ILU 分解により生成された対角項が元の対角項の絶対値の ϵ (特異性防止定数) 倍より小さくなった. (注意事項 (c) 参照)	元の対角項 $\times\epsilon$ を ILU 分解により生成される対角項にし, 処理を続ける.
2000	最大反復回数に達した.	その時点で得られた結果を返す.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3100	制限条件 (b) を満足しなかった.	
4000	右辺ベクトル$\boldsymbol{b}$のノルムが $\sqrt{\text{オーバフロー判定値}}$ より大きい (注意事項 (d),(e) 参照)	
4100	残差のノルムが $\sqrt{\text{オーバフロー判定値}}$ より大きい. (注意事項 (d),(e) 参照)	
4200	$\|(\boldsymbol{r}, \boldsymbol{r}^*)\|$ がアンダフロー判定値より小さい.	
4210	$\|(\boldsymbol{r}, \boldsymbol{r}^*)\|$ がオーバフロー判定値より大きい.	
4300	$\|(A\boldsymbol{p}, \boldsymbol{p}^*)\|$ がアンダフロー判定値より小さい.	
4310	$\|(A\boldsymbol{p}, \boldsymbol{p}^*)\|$ がオーバフロー判定値より大きい.	
5000	対角項に絶対値がアンダフロー判定値より小さいものがある.	

(6) **注意事項**

(a) x, y の関数として微分方程式を定義する関数 func(x,y) の実際の名前は使用者側のプログラムで宣言し, 実際の関数を作成しておかなければならない.

非同次 Helmholtz 方程式

$$u_{xx} + u_{yy} + \lambda u = f(x, y)$$

を解く場合の右辺 $f(x, y)$ となる関数 (倍精度) の作り方は以下に示す通りである.

```
double FORTRAN func(double *x, double *y)
{
    return f(x,y);
}
```

但し,$x \leftrightarrow$ *x,$y \leftrightarrow$ *y, と対応する.

例

$$f(x, y) = 6x + 6y$$

この場合, 関数は次のようになる.

```
double FORTRAN func(double *x, double *y)
{
    return 6.0*(*x) + 6.0*(*y);
}
```

(b) Dirichlet 条件 s1, s2, s3, s4 は, 仮想格子の頂点の値を除く nx + 1 個 (s1, s3), ny + 1 個 (s1, s3) の値を, 座標の小さい方から順番に配列に格納する.

(c) 特異性防止定数 ϵ は 0.1 とする.

(d) ノルムは, 以下に示す l^2 ノルムを採用する.

$$\|\boldsymbol{r}\|_2 = \left(\sum_{i=1}^{n} {r_i}^2 \right)^{1/2} \quad \text{但し}, \boldsymbol{r} = (r_1, r_2, r_3, \cdots, r_n)^T$$

(e) オーバフロー判定値は最大値 $\times 10^{-3}$, アンダフロー判定値は正の最小値 $\times 10^{3}$ を設定している.

(7) 使用例

(a) 問 題 (2 次元 Poisson 問題)

2 次元 Poisson 方程式

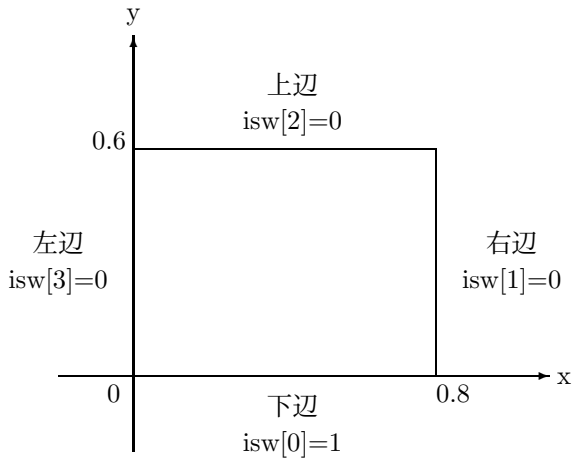
$$u_{xx} + u_{yy} = f(x, y)$$

を境界条件

$$\begin{cases} \frac{\partial u}{\partial y} = 0 & (\text{領域の下辺で}) \\ u = 1 & (\text{その他の辺で}) \end{cases}$$

のもとで解くことを考える.

下図のように与えられた 2 次元領域を, nx × ny の直交格子に離散化する.



(b) 入力データ

関数 f(x, y) の手続き名:FUNC,

dlmd=0.0,

nx=40, ny=50,

xl=0.0, xu=0.6, yl=0.0, yu=0.8,

imax=200,eps=1.e−10,

isw=(1, 0, 0, 0)

(c) 主プログラム

```
/*      C interface example for ASL_dopdh2 */

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
double f(double *x, double *y)
#else
double f(x,y)
double *x;
double *y;
#endif
{
    return 6.0*(*x) + 6.0*(*y);
}
#ifdef __cplusplus
}
#endif

int main()
{
    double dlmd;
    int idvs;
```

```
int jdvs;
double x0;
double x1;
double y0;
double y1;
double *s1;
double *s2;
double *s3;
double *s4;
int imax;
double eps;
double *u;
int imx=100;
int *isw;
int *iwk;
double *wk;
int ierr;

int i,j,nxi,nyi;
FILE *fp;

fp = fopen( "dopdh2.dat", "r" );
if( fp == NULL )
{
    printf( "file open error\n" );
    return -1;
}

printf( "    *** ASL_dopdh2 ***\n" );
printf( "\n     ** Input **\n\n" );

isw = ( int * )malloc((size_t)( sizeof(int) * 4 ));
if( isw == NULL )
{
    printf( "no enough memory for array isw\n" );
    return -1;
}

fscanf( fp, "%d", &isw[0]);
fscanf( fp, "%d", &isw[1]);
fscanf( fp, "%d", &isw[2]);
fscanf( fp, "%d", &isw[3]);

fscanf( fp, "%lf",    &dlmd       );
fscanf( fp, "%d,%d",  &idvs,&jdvs );
fscanf( fp, "%lf,%lf",&x0,&x1     );
fscanf( fp, "%lf,%lf",&y0,&y1     );
fscanf( fp, "%d",     &imax       );
fscanf( fp, "%lf",    &eps        );

printf( "\tisw   = %6d %6d %6d %6d\n", isw[0],isw[1],isw[2],isw[3] );
printf( "\tdlmd  = %8.3g\n",      dlmd        );
printf( "\tnx,ny =   %6d   %6d\n",   idvs,jdvs);
printf( "\tx0,x1 = %8.3g %8.3g\n",x0,x1       );
printf( "\ty0,y1 = %8.3g %8.3g\n",y0,y1       );
printf( "\timax  = %6d\n",         imax        );
printf( "\teps   = %8.3g\n",      eps         );

s1 = ( double * )malloc((size_t)( sizeof(double) * (idvs+1) ));
if( s1 == NULL )
{
    printf( "no enough memory for array s1\n" );
    return -1;
}

s2 = ( double * )malloc((size_t)( sizeof(double) * (jdvs+1) ));
if( s2 == NULL )
{
    printf( "no enough memory for array s2\n" );
    return -1;
}

s3 = ( double * )malloc((size_t)( sizeof(double) * (idvs+1) ));
if( s3 == NULL )
{
    printf( "no enough memory for array s3\n" );
    return -1;
}

s4 = ( double * )malloc((size_t)( sizeof(double) * (jdvs+1) ));
if( s4 == NULL )
{
    printf( "no enough memory for array s4\n" );
    return -1;
}

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

iwk = ( int * )malloc((size_t)( sizeof(int) * 7 ));
if( iwk == NULL )
{
```

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

    wk  = ( double * )malloc((size_t)( sizeof(double) * ((imx+1)*(jdvs+1)*13+5) ));
    if( wk  == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }
    fclose( fp );

    /* Boundary Condition */

    for( i=0 ; i<idvs+1  ; i++ ){
        s1[i] = 1.0;
        s3[i] = 1.0;
    }
    for( j=0 ; j<jdvs+1  ; j++ ){
        s2[j] = 1.0;
        s4[j] = 1.0;
    }

    ierr = ASL_dopdh2(dlmd, f, idvs, jdvs, x0, x1, y0, y1, s1, s2, s3, s4, imax, eps, u, imx, isw, iwk, wk);

    nxi = 5;
    nyi = 6;
    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\n\tSolution u[i]\n\n" );

    for( j=jdvs ; j>=0 ; j -= nyi )
    {
        printf( "\t%6d", j );
        for( i=0 ; i<idvs+1  ; i += nxi )
            printf( "%8.3g", u[i + (imx+1)*j] );
        printf( "\n" );
    }
    printf( "    y-Direction      0       5      10" );
    printf( "      15      20\n" );
    printf( "\t     x-Direction\n" );
    printf( "\n" );

    free(s1 );
    free(s2 );
    free(s3 );
    free(s4 );
    free(u  );
    free(isw);
    free(iwk);
    free(wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dopdh2 ***

 ** Input **

isw   =      1      0      0      0
dlmd  =        1
nx,ny =       20       30
x0,x1 =        0      1.2
y0,y1 =        0      1.8
imax  =   1000
eps   =    1e-09

 ** Output **

ierr =      0

Solution u[i]

    30   0.931   0.742   0.668   0.693     0.9
    24   0.754  -0.128    -0.5  -0.313    0.67
    18   0.731  -0.313  -0.777  -0.524   0.639
    12   0.763  -0.201  -0.656  -0.417    0.67
     6   0.812-0.00269  -0.417   -0.22   0.718
     0   0.844   0.121   -0.27 -0.0966   0.751
y-Direction      0       5      10      15      20
     x-Direction
```

2.5.2 ASL_dopdh3, ASL_ropdh3
非同次 Helmholtz 方程式 (3 次元)

(1) **機 能**

与えられた矩形領域内において 3 次元 7 点差分近似を用いて非同次 Helmholtz 方程式,

$$u_{xx} + u_{yy} + u_{zz} + \lambda u = f(x, y, z)$$

を解く.

(2) **使用法**

倍精度関数:

ierr = ASL_dopdh3 (dlmd, func, nx, ny, nz, xl, xu,yl,yu, zl, zu, s1, s2, s3, s4, s5, s6, imx, jmx, imax, eps, u, isw, iw, w);

単精度関数:

ierr = ASL_ropdh3 (dlmd, func, nx, ny, nz, xl, xu,yl,yu, zl, zu, s1, s2, s3, s4, s5, s6, imx, jmx, imax, eps, u, 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　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	dlmd	{ D / R }	1	入 力	u の定数 λ
2	func	−	−	入 力	x,y,z の関数として微分方程式を定義する関数 f(x,y,z) の関数名
3	nx	I	1	入 力	X 軸方向の分割数 n_x
4	ny	I	1	入 力	Y 軸方向の分割数 n_y
5	nz	I	1	入 力	Z 軸方向の分割数 n_z
6	xl	I	1	入 力	領域の左側の X 座標 x_L
7	xu	I	1	入 力	領域の右側の X 座標 x_U
8	yl	I	1	入 力	領域の左側の Y 座標 y_L
9	yu	I	1	入 力	領域の右側の Y 座標 y_U
10	zl	I	1	入 力	領域の左側の Z 座標 z_L
11	zu	I	1	入 力	領域の右側の Z 座標 z_U
12	s1	I*	内容参照	入 力	下面の Dirichlet 境界条件の値 (isw[0] $\neq 1$ の時) isw[0] $= 1$ の時は領域のみ確保. **大きさ:** (imx $+ 1$) $\times$ (nz $+ 1$)
13	s2	I*	内容参照	入 力	右側面の Dirichlet 境界条件の値 (isw[1] $\neq 1$ の時) isw[1] $= 1$ の時は領域のみ確保. **大きさ:** (jmx $+ 1$) $\times$ (nz $+ 1$)

項番	引数と戻り値	型	大きさ	入出力	内　　容
14	s3	I*	内容参照	入 力	上面の Dirichlet 境界条件の値 (isw[2] $\neq 1$ の時) isw[2] $= 1$ の時は領域のみ確保. **大きさ:** $(\mathrm{imx}+1)\times(\mathrm{nz}+1)$
15	s4	I*	内容参照	入 力	左側面の Dirichlet 境界条件の値 (isw[3] $\neq 1$ の時) isw[3] $= 1$ の時は領域のみ確保. **大きさ:** $(\mathrm{jmx}+1)\times(\mathrm{nz}+1)$
16	s5	I*	内容参照	入 力	前面の Dirichlet 境界条件の値 (isw[4] $\neq 1$ の時) isw[4] $= 1$ の時は領域のみ確保. **大きさ:** $(\mathrm{imx}+1)\times(\mathrm{ny}+1)$
17	s6	I*	内容参照	入 力	背面の Dirichlet 境界条件の値 (isw[5] $\neq 1$ の時) isw[5] $= 1$ の時は領域のみ確保. **大きさ:** $(\mathrm{imx}+1)\times(\mathrm{ny}+1)$
18	imx	I	1	入 力	配列 s1,s3,s5,s6 の整合寸法
19	jmx	I	1	入 力	配列 s2,s4 の整合寸法
20	imax	I	1	入 力	連立 1 次方程式を反復法で解く際の最大反復回数.
21	eps	I	1	入 力	打ち切り残差ノルム $\parallel \boldsymbol{b} - A\boldsymbol{u} \parallel / \parallel \boldsymbol{b} \parallel$ **指定可能範囲:** $\geq$ アンダフロー判定値 (注意事項 (e) 参照)
22	u	$\left\{\begin{matrix}\mathrm{D}*\\ \mathrm{R}*\end{matrix}\right\}$	内容参照	出 力	解 $u(x_i, y_j, z_k)$ $\mathrm{u}(i,j,k) = u\ (x_L + i(x_U - x_L)/n_x\ ,\ y_L + j(y_U - y_L)/n_y\ ,\ z_L + k(z_U - z_L)/n_z)$ **大きさ:** $(\mathrm{imx}+1)\times(\mathrm{jmx}+1)\times(\mathrm{nz}+1)$

項番	引数と戻り値	型	大きさ	入出力	内　　容
23	isw	I*	6	入 力	Neumann 条件付加処理スイッチ isw[0] =0　:下面に Dirichlet 条件を付加 =1　:下面に Neumann 条件を付加 isw[1] =0　:右側面に Dirichlet 条件を付加 =1　:右側面に Neumann 条件を付加 isw[2] =0　:上面に Dirichlet 条件を付加 =1　:上面に Neumann 条件を付加 isw[3] =0　:左側面に Dirichlet 条件を付加 =1　:左側面に Neumann 条件を付加 isw[4] =0　:前面に Dirichlet 条件を付加 =1　:前面に Neumann 条件を付加 isw[5] =0　:背面に Dirichlet 条件を付加 =1　:背面に Neumann 条件を付加
24	iw	I*	7	ワーク	作業領域
25	w	$\left\{\begin{matrix}\mathrm{D}* \\ \mathrm{R}*\end{matrix}\right\}$	内容参照	ワーク	作業領域 **大きさ:** $15 \times (\mathrm{nx}+1) \times (\mathrm{ny}+1) \times (\mathrm{nz}+1)+5$
26	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) nx ≥ 2, ny ≥ 2, nz ≥ 2

(b) 各面に付加する境界条件は, 少なくとも 1 面を Dirichlet 条件としなければならない.

(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.	
1400	ILU 分解により生成された対角項が元の対角項の絶対値の ϵ (特異性防止定数) 倍より小さくなった. (注意事項 (c) 参照)	元の対角項 $\times\epsilon$ を ILU 分解により生成される対角項にし, 処理を続ける.
2000	最大反復回数に達した.	その時点で得られた結果を返す.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3100	制限条件 (b) を満足しなかった.	
4000	右辺ベクトル$\boldsymbol{b}$のノルムが $\sqrt{\text{オーバフロー判定値}}$ より大きい (注意事項 (d),(e) 参照)	
4100	残差のノルムが $\sqrt{\text{オーバフロー判定値}}$ より大きい. (注意事項 (d),(e) 参照)	
4200	$\|(\boldsymbol{r}, \boldsymbol{r}^*)\|$ がアンダフロー判定値より小さい.	
4210	$\|(\boldsymbol{r}, \boldsymbol{r}^*)\|$ がオーバフロー判定値より大きい.	
4300	$\|(A\boldsymbol{p}, \boldsymbol{p}^*)\|$ がアンダフロー判定値より小さい.	
4310	$\|(A\boldsymbol{p}, \boldsymbol{p}^*)\|$ がオーバフロー判定値より大きい.	
5000	対角項に絶対値がアンダフロー判定値より小さいものがある.	

(6) **注意事項**

(a) x, y の関数として微分方程式を定義する関数 func(x,y,z) の実際の名前は使用者側のプログラムで宣言し, 実際の関数を作成しておかなければならない.

非同次 Helmholtz 方程式

$$u_{xx} + u_{yy} + u_{zz} + \lambda u = f(x, y, z)$$

を解く場合の右辺 $f(x, y, z)$ となる関数 (倍精度) の作り方は以下に示す通りである.

```
double FORTRAN func(double *x, double *y, double *z)
{
    return f(x,y,z);
}
```

但し,$x \leftrightarrow$ *x,$y \leftrightarrow$ *y,$z \leftrightarrow$ *z と対応する.

例

$$f(x, y) = 6x + 6y + 6z$$

この場合, 関数は次のようになる.

```
double FORTRAN func(double *x, double *y, double *z)
{
    return 6.0*(*x) + 6.0*(*y) + 6.0*(*z);
}
```

(b) Dirichlet 条件 s1, s2, s3, s4, s5, s6 は, 仮想格子の頂点の値を除く (nx + 1) × (nz + 1) 個 (s1, s3), (ny + 1) × (nz + 1) 個 (s2, s4),(nx + 1) × (ny + 1) 個 (s5, s6) の値を, それぞれ 2 次元の配列として格納する.

(c) 特異性防止定数 ϵ は 0.1 とする.

(d) ノルムは, 以下に示す l^2 ノルムを採用する.

$$\|\boldsymbol{r}\|_2 = \left(\sum_{i=1}^{\mathrm{n}} {r_i}^2 \right)^{1/2} \quad \text{但し}, \boldsymbol{r} = (r_1, r_2, r_3, \cdots, r_{\mathrm{n}})^T$$

(e) オーバフロー判定値は最大値 $\times 10^{-3}$, アンダフロー判定値は正の最小値 $\times 10^{3}$ を設定している.

(7) 使用例

(a) 問 題 (3 次元 Poisson 問題)

3 次元 Poisson 方程式

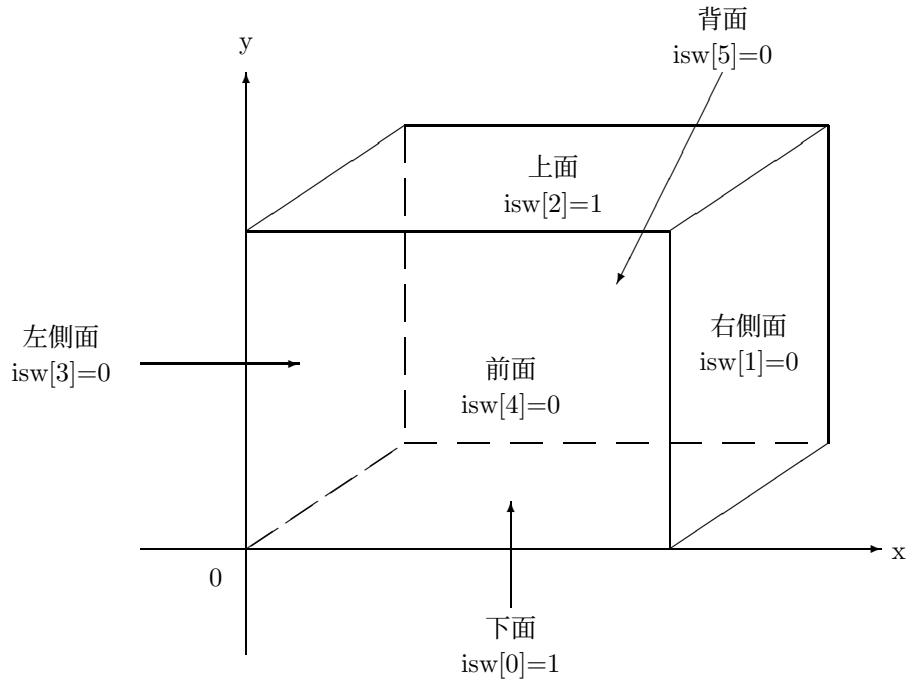
$$u_{xx} + u_{yy} + u_{zz} = f(x, y, z)$$

を境界条件

$$\begin{cases} \frac{\partial u}{\partial y} = 0 & \text{(領域の下面と上面で)} \\ u = 1 & \text{(その他の面で)} \end{cases}$$

のもとで解くことを考える.

下図のように与えられた 3 次元領域を,nx × ny × nz の直交格子に離散化する.



(b) 入力データ

関数 f(x, y, z) の手続き名: FUNC,

dlmd=0.0,

nx=20, ny=30, nz=40,

xl=0.0, xu=0.8, yl=0.0, yu=0.6, zl=0.0, zu=1.2,

imax=200, eps=1.e−10,

isw=(1, 0, 1, 0, 0, 0)

(c) 主プログラム

```
/*      C interface example for ASL_dopdh3 */

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
double f(double *x, double *y, double *z)
#else
double f(x,y,z)
double *x;
double *y;
double *z;
```

```
#endif
{
    return 6.0e0*(*x) + 6.0e0*(*y) + 6.0e0*(*z);
}
#ifdef __cplusplus
}
#endif

int main()
{
    double dlmd;
    int idvs;
    int jdvs;
    int kdvs;
    double x0;
    double x1;
    double y0;
    double y1;
    double z0;
    double z1;
    double *s1;
    double *s2;
    double *s3;
    double *s4;
    double *s5;
    double *s6;
    int imax;
    double eps;
    double *u;
    int imx=50;
    int jmx=50;
    int *isw;
    int *iwk;
    double *wk;
    int ierr;

    int i,j,k,nxi,nyi,nzi,kst,jst;
    FILE *fp;

    fp = fopen( "dopdh3.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dopdh3 ***\n" );
    printf( "\n     ** Input **\n\n" );

    isw = ( int * )malloc((size_t)( sizeof(int) * 6 ));
    if( isw == NULL )
    {
        printf( "no enough memory for array isw\n" );
        return -1;
    }

    fscanf( fp, "%d", &isw[0]);
    fscanf( fp, "%d", &isw[1]);
    fscanf( fp, "%d", &isw[2]);
    fscanf( fp, "%d", &isw[3]);
    fscanf( fp, "%d", &isw[4]);
    fscanf( fp, "%d", &isw[5]);

    fscanf( fp, "%lf",    &dlmd                );
    fscanf( fp, "%d,%d,%d",  &idvs,&jdvs,&kdvs );
    fscanf( fp, "%lf,%lf",&x0,&x1              );
    fscanf( fp, "%lf,%lf",&y0,&y1              );
    fscanf( fp, "%lf,%lf",&z0,&z1              );
    fscanf( fp, "%d",     &imax                );
    fscanf( fp, "%lf",    &eps                 );

    printf( "\tisw   = %6d %6d %6d %6d %6d %6d\n",
        isw[0],isw[1],isw[2],isw[3],isw[4],isw[5] );
    printf( "\tdlmd  = %8.3g\n",      dlmd               );
    printf( "\tidvs  =   %6d\n",idvs                    );
    printf( "\tjdvs  =   %6d\n",jdvs                    );
    printf( "\tkdvs  =   %6d\n",kdvs                    );
    printf( "\tx0,x1 = %8.3g %8.3g\n",x0,x1             );
    printf( "\ty0,y1 = %8.3g %8.3g\n",y0,y1             );
    printf( "\tz0,z1 = %8.3g %8.3g\n",z0,z1             );
    printf( "\timax  = %6d\n",        imax              );
    printf( "\teps   = %8.3g\n",      eps               );

    s1 = ( double * )malloc((size_t)( sizeof(double) * (imx+1)*(kdvs+1) ));
    if( s1 == NULL )
    {
        printf( "no enough memory for array s1\n" );
        return -1;
    }

    s2 = ( double * )malloc((size_t)( sizeof(double) * (jmx+1)*(kdvs+1) ));
    if( s2 == NULL )
    {
        printf( "no enough memory for array s2\n" );
        return -1;
```

```
}

s3 = ( double * )malloc((size_t)( sizeof(double) * (imx+1)*(kdvs+1) ));
if( s3 == NULL )
{
    printf( "no enough memory for array s3\n" );
    return -1;
}

s4 = ( double * )malloc((size_t)( sizeof(double) * (jmx+1)*(kdvs+1) ));
if( s4 == NULL )
{
    printf( "no enough memory for array s4\n" );
    return -1;
}

s5 = ( double * )malloc((size_t)( sizeof(double) * (imx+1)*(jdvs+1) ));
if( s5 == NULL )
{
    printf( "no enough memory for array s3\n" );
    return -1;
}

s6 = ( double * )malloc((size_t)( sizeof(double) * (imx+1)*(jdvs+1) ));
if( s6 == NULL )
{
    printf( "no enough memory for array s4\n" );
    return -1;
}

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

iwk = ( int * )malloc((size_t)( sizeof(int) * 7 ));
if( iwk == NULL )
{
    printf( "no enough memory for array iwk\n" );
    return -1;
}

wk  = ( double * )malloc((size_t)( sizeof(double) * (imx+1)*(jmx+1)*(kdvs+1)*13+5 ));
if( wk  == NULL )
{
    printf( "no enough memory for array wk\n" );
    return -1;
}
fclose( fp );

/* Boundary Condition */

for( i=0 ; i<idvs+1  ; i++ ){
    for( j=0 ; j<jdvs+1  ; j++ ){
        s5[i+(imx+1)*j] = 1.0e0;
        s6[i+(imx+1)*j] = 1.0e0;
    }
}
for( j=0 ; j<jdvs+1  ; j++ ){
    for( k=0 ; k<kdvs+1  ; k++ ){
        s2[j+(jmx+1)*k] = 1.0e0;
        s4[j+(jmx+1)*k] = 1.0e0;
    }
}
for( i=0 ; i<idvs+1  ; i++ ){
    for( k=0 ; k<kdvs+1  ; k++ ){
        s1[i+(imx+1)*k] = 1.0e0;
        s3[i+(imx+1)*k] = 1.0e0;
    }
}

ierr = ASL_dopdh3
(dlmd,f,idvs,jdvs,kdvs,x0,x1,y0,y1,z0,z1,s1,s2,s3,s4,s5,s6,
 imx,jmx,imax,eps,u,isw,iwk,wk);

nxi = 5;
nyi = 6;
nzi = 5;
printf( "\n     ** Output **\n\n" );
printf( "\tierr = %6d\n", ierr );

printf( "\n\tSolution u[i]\n\n" );

for( k=0 ; k<kdvs+1 ; k += nzi )
{
    kst = k*(imx+1)*(jmx+1);
    printf( "\n\tz-Direction = %6d\n\n",k );
    for( j=jdvs ; j>=0 ; j -= nyi )
    {
        printf( "\t%6d", j );
        jst = (imx+1)*j;
        for( i=0 ; i<idvs+1  ; i += nxi )
        {
            printf( "%8.3g", u[kst + jst + i] );
```

```
            }
        printf( "\n" );
        }
        printf( "\n" );
        printf( "    y-Direction      0       5      10" );
        printf( "      15      20\n" );
        printf( "\t     x-Direction\n" );
        printf( "\n" );
    }

    free(s1 );
    free(s2 );
    free(s3 );
    free(s4 );
    free(s5 );
    free(s6 );
    free(u  );
    free(isw);
    free(iwk);
    free(wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dopdh3 ***

 ** Input **

isw   =      1      0      1      0      0      0
dlmd  =        0
idvs  =       20
jdvs  =       30
kdvs  =       20
x0,x1 =        0      0.8
y0,y1 =        0      0.6
z0,z1 =        0      1.2
imax  =    200
eps   =    1e-05

 ** Output **

ierr =      0

Solution u[i]


z-Direction =      0

    30    0.98   0.916   0.887   0.895   0.968
    24   0.981   0.919   0.891   0.899   0.969
    18   0.983   0.925   0.897   0.905   0.971
    12   0.986   0.932   0.905   0.911   0.974
     6   0.988   0.938   0.912   0.917   0.976
     0   0.989   0.941   0.915   0.921   0.977

y-Direction      0       5      10      15      20
     x-Direction


z-Direction =      5

    30   0.922   0.639   0.519   0.583   0.897
    24   0.925   0.647   0.529   0.591   0.899
    18    0.93   0.665   0.549   0.609   0.904
    12   0.935   0.685   0.572   0.628    0.91
     6    0.94   0.702   0.593   0.646   0.914
     0   0.942   0.711   0.602   0.654   0.917

y-Direction      0       5      10      15      20
     x-Direction


z-Direction =     10

    30   0.893   0.504   0.347   0.444   0.866
    24   0.895   0.513   0.357   0.453   0.869
    18     0.9   0.532   0.379   0.471   0.874
    12   0.906   0.553   0.405   0.493    0.88
     6   0.911   0.572   0.427   0.512   0.885
     0   0.914   0.581   0.438   0.521   0.887

y-Direction      0       5      10      15      20
     x-Direction


z-Direction =     15

    30    0.89   0.511   0.366   0.454   0.864
    24   0.892   0.519   0.375   0.463   0.867
    18   0.897   0.536   0.395    0.48   0.872
    12   0.903   0.556   0.419     0.5   0.877
     6   0.907   0.574   0.439   0.517   0.882
     0    0.91   0.582   0.449   0.526   0.885

y-Direction      0       5      10      15      20
     x-Direction
```

```
z-Direction =     20

    30   0.958    0.85   0.812   0.829   0.947
    24    0.96   0.853   0.816   0.833   0.948
    18   0.962   0.859   0.823   0.839    0.95
    12   0.964   0.866    0.83   0.845   0.952
     6   0.966   0.872   0.837   0.851   0.954
     0   0.967   0.875   0.841   0.855   0.956

y-Direction      0       5      10      15      20
     x-Direction
```

第 3 章　数値微分

3.1　概 要

本章では, 与えられた点における関数の数値微分値を求める関数について説明する.
1 変数関数の数値微分では, 以下の関数が用意されている.

(1) 関数の数値微分
この関数は, 与えられた点における 1 変数関数の任意階微分値を求める.
1 つの多変数関数の数値微分では, 以下の関数が用意されている.

(2) 多変数関数の傾斜ベクトル

(3) 多変数関数のヘッセ行列
(2) の関数は, 与えられた点における多変数関数の 1 階偏微分値 (傾斜ベクトル) を求める.
(3) の関数は, 与えられた点における多変数関数の 2 階偏微分値 (ヘッセ行列) を求める.

複数の多変数関数の数値微分では, 以下の関数が用意されている.

(4) 複数の多変数関数のヤコビ行列
この関数は, 複数の多変数関数に対し与えられた点における 1 階偏微分値 (ヤコビ行列) を求める.

3.1.1 使用上の注意

(1) 要求相対精度としては, 既定値よりも大きな値を入力することが望ましい.

(2) 関数の引数として渡す関数の作り方は次の通りである.

例

関数の数値微分 (f は主プログラムと関数とで同じ名前とする.)

- 主プログラム

```
~
ierr = { ASL_dqfodx
       { ASL_rqfodx  (f, ···);
~
```

- 関数

```
double FORTRAN f (double *x)
{
  ~
  return f(*x);
}
```

例

多変数関数の傾斜ベクトルと多変数関数のヘッセ行列 (f は主プログラムと関数とで同じ名前とする.)

- 主プログラム

```
~
ierr = { ASL_dqmo**
       { ASL_rqmo**  (f, ···);
~
```

- 関数

```
double FORTRAN f (double *x)
{
  ~
  return f(x[0], ···, x[nx − 1]);
}
```

例

複数の多変数関数のヤコビ行列 (sub は主プログラムと関数とで同じ名前とする.)

- 主プログラム

```
~
ierr = { ASL_dqmojx
       { ASL_rqmojx  (sub, ···);
~
```

- 関数

```
void FORTRAN sub (double *x, int *nx, double *f, int *nf)
{
  ~
  f[0] = f_1(x[0], ···, x[(*nx) − 1]);
    ~
  f[(*nf) − 1] = f_(*nf)(x[0], ···, x[(*nx) − 1]);
}
```

3.1.2 使用しているアルゴリズム

3.1.2.1 リチャードソン補外

関数 $f(x)$ の x における微分の値 $f^{(1)}(x)$ を差分によって求める.
$f(x+h)$, $f(x-h)$ のテーラー展開より,

$$f^{(1)}(x) = \frac{f(x+h)-f(x-h)}{2h} + c_1h^2 + c_2h^4 + \cdots + c_ih^{2i} + \cdots \tag{3.1}$$

ただし

$$c_i = -\frac{f^{(2i+1)}(x)}{(2i+1)!} \ (i = 1, 2, \cdots)$$

と書ける.
刻み幅 h の中心差分を

$$D_0^{(0)} = \frac{f(x+h)-f(x-h)}{2h}$$

とおくと, (3.1) 式より $D_0^{(0)}$ と $f^{(1)}(x)$ との誤差は h^2 のオーダーである.
いま刻み幅を半分の $\dfrac{h}{2}$ にしたときの中心差分を

$$D_0^{(1)} = \frac{f(x+\frac{h}{2})-f(x-\frac{h}{2})}{2(\frac{h}{2})}$$

とおき, $D_0^{(0)}$ と $D_0^{(1)}$ から

$$D_1^{(0)} = \frac{1}{3}(4D_0^{(1)} - D_0^{(0)})$$

なる量を定義すると, この $D_1^{(0)}$ と $f^{(1)}(x)$ との誤差は (3.1) 式と, (3.1) 式で $h \to \frac{h}{2}$ と置き換えた式から

$$f^{(1)}(x) - D_1^{(0)} = -\frac{1}{4}c_2h^4 + \cdots$$

と表せ, h の 4 乗のオーダーとなり $D_1^{(0)}$ は $D_0^{(0)}$ より高次の $f^{(1)}(x)$ に対する近似を与える.
この手順を一般化し, (3.1) 式の h の高次の項を順次消去すると $f^{(1)}(x)$ の高次の近似が順次得られる. まず, 適当な h から出発して

$$D_0^{(k)} = \frac{f(x+\frac{h}{2^k})-f(x-\frac{h}{2^k})}{2(\frac{h}{2^k})} \ (k = 0, 1, 2, \cdots)$$

を計算する. 次に, $m = 1, 2, \cdots$ に対して (リチャードソン補外)

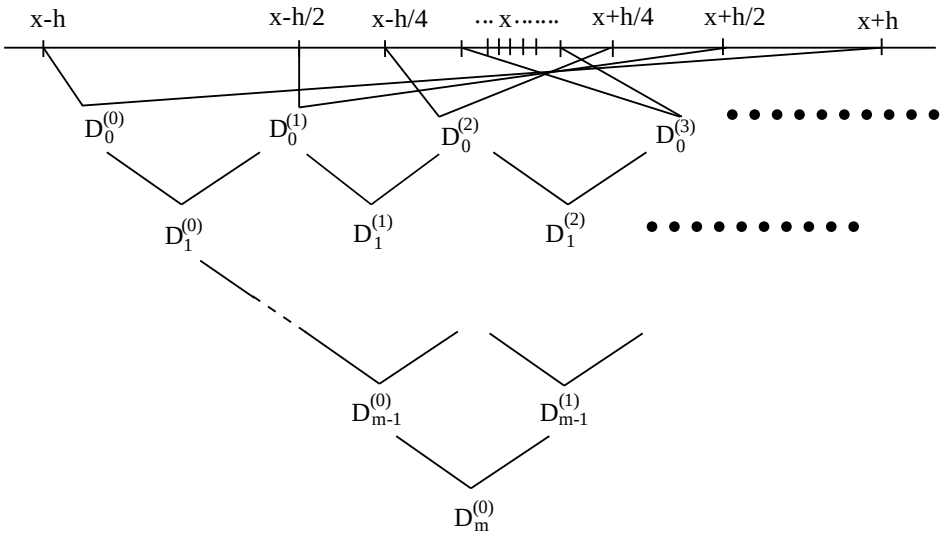
$$\begin{aligned} D_m^{(k)} &= \frac{4^mD_{m-1}^{(k+1)} - D_{m-1}^{(k)}}{4^m-1} \\ &= D_{m-1}^{(k+1)} + \frac{1}{4^m-1}(D_{m-1}^{(k+1)} - D_{m-1}^{(k)}) \end{aligned}$$

を計算する (図 3.1.2.1 参照). $D_m^{(0)}$ の誤差は $h^{2(m+1)}$ のオーダーになる.
収束判定には条件

$$\frac{|D_m^{(0)} - D_{m-1}^{(0)}|}{|D_m^{(0)}|} \le 3 \times \text{EPSN} \quad (\text{EPSN は要求相対精度})$$

を用い, これが成立したとき収束したものとし, $D_m^{(0)}$ を $f^{(1)}(x)$ の近似値として採用する.

図 3–1　リチャードソン補外



3.1.2.2　関数の数値微分

任意階微分を行うため高階微分公式を考える.

n が偶数のとき n 階微分公式を

$$f^{(n)}(x) \sim \frac{\sum_k b_k(f(x+kh)+f(x-kh))}{h^n}$$

ただし, k は $1 \le k \le \frac{n}{2}$ を満たすとする. 係数 b_k は二項定理から

$$\begin{aligned} b_k &= {}_{n/2}C_k \quad (k \text{ が奇数}) \\ b_k &= -{}_{n/2}C_k \quad (k \text{ が偶数}) \end{aligned}$$

として与えられる.

n が奇数のときは n 階微分公式を

$$f^{(n)}(x) \sim \frac{\sum_k b_k(f(x+kh)-f(x-kh))}{2h^n}$$

ただし, k は $1 \le k \le \lfloor \frac{n}{2} \rfloor + 1$ を満たすとする. ($\lfloor x \rfloor$ は x を超えない最大の整数を表す.)

係数 b_k は二項定理から

$$\begin{aligned} b_k &= {}_{\lfloor \frac{n}{2} \rfloor+1}C_k \quad (k \text{ が奇数}) \\ b_k &= -{}_{\lfloor \frac{n}{2} \rfloor+1}C_k \quad (k \text{ が偶数}) \end{aligned}$$

となり, 更に $n \ge 5$ のときは得られた b_k に対して

$$b_k \leftarrow b_k - b_{k-2} \ (k = \lfloor \frac{n}{2} \rfloor + 1, \cdots, 3)$$

として求められる.

こうして得られた n 階微分公式に対して, リチャードソン補外を行う.

また, 刻み幅が

$$\max(x\varepsilon, \varepsilon^{\frac{1}{n+1}}) \ (\varepsilon\text{は誤差判定のための単位})$$

よりも小さくなったならば, 処理を打ち切る.

3.1.2.3 多変数関数の傾斜ベクトル

複数の変数 $x_i, \cdots, x_{nx}$ において, x_i $(i = 1, \cdots, nx)$ 以外を固定する.
そこで, リチャードソン補外を用いて x_i についての 1 階微分値を求める.
すなわち,

$$\frac{\partial f}{\partial x_i} \ (i = 1, \cdots, nx)$$

が関数 f の x_i における傾斜ベクトルである.
リチャードソン補外を行う際, 刻み幅が

$$\max_{i=1,\cdots,nx} (x_i\varepsilon, \varepsilon^{0.25}) \ (\varepsilon\text{は誤差判定のための単位})$$

よりも小さくなったならば, 処理を打ち切る.

3.1.2.4 多変数関数のヘッセ行列

変数 $x_1, \cdots, x_{nx}$ におけるそれぞれの 1 階微分値を求める.
その 1 階微分値についてリチャードソン補外を行う.
複数の変数 $x_1, \cdots, x_{nx}$ において, x_i $(i = 1, \cdots, nx)$ 以外を固定し, 変数各々の 1 階微分値に対する x_i $(i = 1, \cdots, nx)$ の 1 階微分値を求める.
すなわち,

$$\frac{\partial^2 f}{\partial x_i \partial x_j} \ (i = 1, \cdots, nx; \ j = 1, \cdots, nx)$$

が, 関数に対するヘッセ行列の (i, j) 要素である.
リチャードソン補外を行う際, 刻み幅が

$$\max_{i=1,\cdots,nx} (x_i\varepsilon, \varepsilon^{0.25}) \ (\varepsilon\text{は誤差判定のための単位})$$

よりも小さくなったならば, 処理を打ち切る.

3.1.2.5 複数の多変数関数のヤコビ行列

傾斜ベクトルを求めるアルゴリズムと同様で, 複数の関数 $f_1, \cdots, f_{nf}$ に対して変数 $x_1, \cdots, x_{nx}$ におけるそれぞれの 1 階微分値を求める.
すなわち,

$$\frac{\partial f_i}{\partial x_j} \ (i = 1, \cdots, nf; \ j = 1, \cdots, nx)$$

はヤコビ行列の (i, j) 要素である.
リチャードソン補外を行う際, 刻み幅が

$$\max_{i=1,\cdots,nx} (x_i\varepsilon, \varepsilon^{0.25}) \ (\varepsilon\text{は誤差判定のための単位})$$

よりも小さくなったならば, 処理を打ち切る.

3.1.3 参考文献

(1) 一松信, 戸川隼人編, “数値計算における誤差”, 共立出版, (1975).

(2) 森正武著, “数値計算プログラミング”, 岩波書店, (1986).

3.2 数値微分

3.2.1 ASL_dqfodx, ASL_rqfodx 関数の数値微分

(1) **機 能**

差分とリチャードソン補外を使って, 関数 $f(x)$ の n 階微分値 $f^{(n)}(x)$ を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dqfodx  (f, x, n, eps, & mr, h, & del, wk);
```

単精度関数:

```
ierr = ASL_rqfodx  (f, x, n, eps, & mr, h, & del, 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 $f(x)$ を定義する関数 f(x) の関数名 (注意事項 (a) 参照)
2	x	{D / R}	1	入 力	微分点 x
3	n	I	1	入 力	微分階数 n
4	eps	{D / R}	1	入 力	要求相対精度 (既定値: (誤差判定のための単位)$^{\frac{2}{(n+2)}}$)
5	mr	I*	1	入 力	最大反復数 (既定値: 100)
				出 力	実際の反復数
6	h	{D / R}	1	入 力	最初の刻み幅
7	del	{D* / R*}	1	出 力	微分点における関数の n 階微分値 $f^{(n)}(x)$
8	wk	{D* / R*}	内容参照	ワーク	作業領域 (注意事項 (b) 参照) **大きさ:** $\lfloor n/2 \rfloor + mr + 2$ ($\lfloor x \rfloor$ は, x を超えない最大の整数を表す.)
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) eps $\geq \varepsilon \times 64$ (既定値にするため，0.0 を入力する場合を除く)
(ε は誤差判定のための単位)

(b) mr>0 (既定値にするため，0 を入力する場合を除く)

(c) n>0

(d) $h > \max(x \times \varepsilon, \varepsilon^{1/(n+3)})$
(ε は誤差判定のための単位)

(5) **エラーインディケータ (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.
2000	要求精度になる前に, 誤差が大きくなった. または, 刻み幅が小さくなった.	その時点での計算値を出力して処理を打ち切る.
2500	要求精度にならず最大反復数に達した.	
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.

(6) **注意事項**

(a) 関数 f の作り方は次のようにする.

```
double FORTRAN f(double *x)
{
        return (f(*x));
}
```

(b) 引数第 8 項 wk の領域を配列として確保する際に, $mr \leq 0$ のときは配列の大きさを $\lfloor n/2 \rfloor + 102$ にする.

(c) 要求相対精度としては, 既定値よりも大きい値を入力することが望ましい.

(d) 引数第 6 項 h の値は 0.5 ～ 1.0 程度であるのが望ましい.

(e) 収束判定は {| 補外値 − 前回の補外値 | ≤ | eps × 補外値 |} が成立したとき収束したものとする.

(f) 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.

(g) 単精度で微分階数 n を n>5 とすると, 精度低下を起こしやすいので倍精度を利用した方がよい. ただし, 倍精度であっても $n \leq 15$ 程度が望ましい.

(7) **使用例**

(a) 問 題
関数 $f(x) = \sin(x)$ の $x = -3.75$ における 5 階微分の近似値を求める.

(b) 入力データ
関数名: f
x=−3.75, n=5, eps=0.0, mr=15, h=0.5

(c) 主プログラム

```
/*      C interface example for ASL_dqfodx */

#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 sin(*x);
}
#ifdef __cplusplus
}
#endif

int main()
{
    double x;
    int n;
    double epsn;
    int mr;
    double h;
    double d;
    double *wk;
    int ierr;
    int nwk;
    FILE *fp;

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

    printf( "    *** ASL_dqfodx ***\n" );
    printf( "\n     ** Input **\n\n" );
    n=5;

    fscanf( fp, "%lf", &x );
    fscanf( fp, "%lf", &epsn );
    fscanf( fp, "%d", &mr );
    fscanf( fp, "%lf", &h );
    printf( "\tOrder of Differentiation = %6d\n",n );
    printf( "\tx   = %8.3g\n",x );
    printf( "\teps = %8.3g\n",epsn );
    printf( "\tmr  =   %6d\n",mr );
    printf( "\th   = %8.3g\n",h );

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

    fclose( fp );

    ierr = ASL_dqfodx(f, x, n, epsn, &mr, h, &d, wk);

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

    printf( "\n\tIteration Number   =   %6d\n",mr );
    printf( "\tDifferential Value = %8.3g\n",d );

    free( wk  );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dqfodx ***

 ** Input **

Order of Differentiation =      5
x   =    -3.75
eps =        0
mr  =       15
h   =      0.5

 ** Output **

ierr =      0

Iteration Number   =        3
Differential Value =   -0.821
```

3.2.2 ASL_dqmogx, ASL_rqmogx
多変数関数の傾斜ベクトル

(1) **機 能**

差分とリチャードソン補外を使って, 多変数関数 $f(\boldsymbol{x})(\boldsymbol{x} = \{x_j\}; j = 1, \cdots, nx)$ の傾斜ベクトル $\partial f/\partial \boldsymbol{x} = \{\partial f/\partial x_j\}$ を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dqmogx (f, x, nx, eps, mr, h, grad, wk);

単精度関数:

ierr = ASL_rqmogx (f, x, nx, eps, mr, h, grad, 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	f	{D / R}	–	入 力	関数 $f(\boldsymbol{x})$ を定義する関数 f(x) の関数名
2	x	{D∗ / R∗}	nx	入 力	微分点 $(x_1, \cdots, x_{nx})$
3	nx	I	1	入 力	独立変数 x の数 nx
4	eps	{D / R}	1	入 力	要求相対精度 (既定値: (誤差判定のための単位)$^{\frac{2}{3}}$)
5	mr	I	1	入 力	最大反復数 (既定値 : 100)
6	h	{D / R}	1	入 力	最初の刻み幅
7	grad	{D∗ / R∗}	nx	出 力	微分点における関数の傾斜ベクトル $\partial f/\partial \boldsymbol{x}$
8	wk	{D∗ / R∗}	mr+1	ワーク	作業領域 (注意事項 (b) 参照)
9	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) eps≥ $\varepsilon \times 64$ (既定値にするため, 0.0 を入力する場合を除く)
(ε は誤差判定のための単位)

(b) mr> 0 (既定値にするため, 0 を入力する場合を除く)

(c) nx> 0

(d) h $\geq \max(x_i \times \varepsilon, \varepsilon^{0.25})$ $(i = 1, \cdots, \mathrm{nx})$
(ε は誤差判定のための単位)

(5) **エラーインディケータ (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.
2000	要求精度になる前に, 誤差が大きくなった. または, 刻み幅が小さくなった.	その時点での計算値を出力して処理を打ち切る.
2500	要求精度にならず最大反復数に達した.	
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.

(6) **注意事項**

(a) 関数 f の作り方は次のようにする.

```
double FORTRAN f(double *x)
{
        return (f(x[0], ···, x[nx−1]));
}
```

(b) 引数第 8 項 wk の領域を確保する際に, mr$\leq$ 0 のときは配 列の大きさを 101 にする.

(c) 要求相対精度としては, 既定値よりも大きい値を入力することが望ましい.

(d) 引数第 6 項 h の値は, 0.5 ～ 1.0 程度であるのが望ましい.

(e) 収束判定は {| 補外値 − 前回の補外値 | $\leq$ | eps × 補外値 |} が成立したとき収束したものとする.

(f) 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.

(g) 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.

(7) **使用例**

(a) 問 題

関数 $f(x_1, x_2, x_3) = x_1x_2x_3$ の $(x_1, x_2, x_3) = (2.1, 8.59, 0.315)$ における傾斜ベクトルを求める.

(b) 入力データ

関数名: f

x [0] =2.1, x [1] =8.59, x [2] =0.315, nx=3, eps=1.0e−9, mr=15, h=0.5

(c) 主プログラム

```
/*      C interface example for ASL_dqmogx */

#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[0]*x[1]*x[2]);
}
#ifdef __cplusplus
}
#endif

int main()
{
    double *x1;
    int nx;
    double epsn;
    int mr;
    double h;
    double *grad;
    double *wk;
    int ierr;
    int i;
    FILE *fp;

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

    printf( "    *** ASL_dqmogx ***\n" );
    printf( "\n     ** Input **\n\n" );
    nx=3;
    mr=15;
    x1 = ( double * )malloc((size_t)( sizeof(double) * nx ));
    if( x1  == NULL )
    {
        printf( "no enough memory for array x1\n" );
        return -1;
    }
    grad  = ( double * )malloc((size_t)( sizeof(double) * nx ));
    if( grad  == NULL )
    {
        printf( "no enough memory for array grad\n" );
        return -1;
    }
    wk  = ( double * )malloc((size_t)( sizeof(double) * (mr+1) ));
    if( wk  == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

    for( i=0 ; i<nx ; i++ )
    {
        fscanf( fp, "%lf", &x1[i] );
    }
    fscanf( fp, "%lf", &epsn );
    fscanf( fp, "%lf", &h );
    printf( "\tn   =   %6d\n",nx );
    printf( "\tx   = " );
    for( i=0 ; i<nx ; i++ )
    {
        printf( "%8.3g",x1[i] );
    }
    printf( "\n" );
    printf( "\teps = %8.3g\n",epsn );
    printf( "\tmr  =   %6d\n",mr );
    printf( "\th   = %8.3g\n",h );

    fclose( fp );

    ierr = ASL_dqmogx(f, x1, nx, epsn, mr, h, grad, wk);

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

    printf( "\n" );
    for( i=0 ; i<nx ; i++ )
    {
        printf( "\tgrad[%6d] = %8.3g\n", i,grad[i] );
    }

    free( x1 );
    free( grad );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dqmogx ***

 ** Input **

n   =        3
x   =      2.1    8.59   0.315
```

```
eps =    1e-09
mr  =       15
h   =      0.5

 ** Output **

ierr =       0

grad[     0] =     2.71
grad[     1] =    0.662
grad[     2] =       18
```

3.2.3 ASL_dqmohx, ASL_rqmohx
多変数関数のヘッセ行列

(1) **機 能**

リチャードソン補外を使って, 多変数関数 $f(\boldsymbol{x})(\boldsymbol{x}=\{x_j\}; j=1,\cdots,nx)$ のヘッセ行列 $H=\{\partial^2 f/(\partial x_i \partial x_j)\}(i=1,\cdots,nx;\ j=1,\cdots,nx)$ を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dqmohx  (f, x, nx, eps, mr, h, hes, iwk, wk);
```

単精度関数:

```
ierr = ASL_rqmohx  (f, x, nx, eps, mr, h, hes, 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	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∗}	nx	入 力	微分点 $(x_1,\cdots,x_{nx})$
3	nx	I	1	入 力	独立変数 x の数 nx
4	eps	{D / R}	1	入 力	要求相対精度 (既定値: (誤差判定のための単位)$^{\frac{4}{9}}$)
5	mr	I	1	入 力	最大反復数 (既定値 : 100)
6	h	{D / R}	1	入 力	最初の刻み幅
7	hes	{D∗ / R∗}	nx × nx	出 力	微分点における関数のヘッセ行列 $H=\{\partial^2 f/(\partial x_i \partial x_j)\}$
8	iwk	I*	nx	ワーク	作業領域
9	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 (注意事項 (b) 参照) **大きさ:** (nx + mr + 1) × (5 + mr)
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) eps $\geq \varepsilon \times 64$ (既定値にするため, 0.0 を入力する場合を除く)
(ε は誤差判定のための単位)

(b) mr>0 (既定値にするため, 0 を入力する場合を除く)

(c) nx>0

(d) h $\geq \max(x_i \times \varepsilon, \varepsilon^{0.25})$ $(i=1,\cdots,\mathrm{nx})$
(ε は誤差判定のための単位)

(5) **エラーインディケータ (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.
2000	要求精度になる前に, 誤差が大きくなった. または, 刻み幅が小さくなった.	その時点での計算値を出力して処理を打ち切る.
2500	要求精度にならず最大反復数に達した.	
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.

(6) **注意事項**

(a) 関数 f の作り方は次のようにする.

```
double FORTRAN f(double *x)
{
        return (f(x[0], ···, x[nx−1]));
}
```

(b) 引数第 9 項 wk の領域を配列として確保する際に, mr≤0 のときは配列の大きさを (nx+101) × 105 とする.

(c) 要求相対精度としては, 既定値よりも大きい値を入力することが望ましい.

(d) 引数第 6 項 h の値は, 0.5 ∼ 1.0 程度であるのが望ましい.

(e) 収束判定は {| 補外値 − 前回の補外値 | ≤ | eps × 補外値 |} が成立したとき収束したものとする.

(f) 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.

(g) 微分点でのヘッセ行列の各要素は配列 hes に以下のよう格納される.
$\mathrm{hes}[(i-1)+\mathrm{nx}*(j-1)] = \partial^2 f/\partial x_i \partial x_j \ (i=1,2,\cdots,\mathrm{nx};\ j=1,2,\cdots,\mathrm{nx})$

(h) 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.

(7) **使用例**

(a) 問 題
関数 $f(x_1,x_2,x_3) = x_1x_2x_3$ の $(x_1,x_2,x_3) = (5.5, 1.0, 8.0)$ におけるヘッセ行列を求める.

(b) 入力データ
関数名: f
x [0] =5.5, x [1] =1.0, x [2] =8.0, nx=3, eps=1.0e−10, mr=15, h=0.5

(c) 主プログラム

```
/*      C interface example for ASL_dqmohx */

#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[0]*x[1]*x[2]);
}
#ifdef __cplusplus
}
#endif


int main()
{
    double *x1;
    int nx;
    double epsn;
    int mr;
    double h;
    double *ans;
    int *iwk;
    double *wk;
    int ierr;
    int i,j;
    FILE *fp;

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

    printf( "    *** ASL_dqmohx ***\n" );
    printf( "\n     ** Input **\n\n" );
    nx=3;
    mr=15;
    x1  = ( double * )malloc((size_t)( sizeof(double) * nx ));
    if( x1  == NULL )
    {
        printf( "no enough memory for array x1\n" );
        return -1;
    }
    ans  = ( double * )malloc((size_t)( sizeof(double) * (nx*nx) ));
    if( ans  == NULL )
    {
        printf( "no enough memory for array ans\n" );
        return -1;
    }
    wk  = ( double * )malloc((size_t)( sizeof(double) * ((nx+mr+1)*(5+mr)) ));
    if( wk  == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }
    iwk  = ( int * )malloc((size_t)( sizeof(int) * nx ));
    if( iwk  == NULL )
    {
        printf( "no enough memory for array iwk\n" );
        return -1;
    }

    for( i=0 ; i<nx ; i++ )
    {
        fscanf( fp, "%lf", &x1[i] );
    }
    fscanf( fp, "%lf", &epsn );
    fscanf( fp, "%lf", &h );
    printf( "\tNumber of Variable = %6d\n",nx );
    printf( "\n\tx   = " );
    for( i=0 ; i<nx ; i++ )
    {
        printf( "%8.3g ",x1[i] );
    }
    printf( "\n" );
    printf( "\n\teps = %8.3g\n",epsn );
    printf( "\tmr  =   %6d\n",mr );
    printf( "\th   = %8.3g\n",h );

    fclose( fp );

    ierr = ASL_dqmohx(f, x1, nx, epsn, mr, h, ans, iwk, wk);

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

    free( x1 );
```

```
    free( ans );
    free( wk );
    free( iwk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dqmohx ***

 ** Input **

Number of Variable =      3

x   =      5.5        1        8

eps =    1e-10
mr  =       15
h   =      0.5

 ** Output **

ierr =      0

Hesse[i][j] =

       0           8           1
       8           0         5.5
       1         5.5           0
```

3.2.4 ASL_dqmojx, ASL_rqmojx
複数の多変数関数のヤコビ行列

(1) **機 能**

リチャードソン補外を使って, 複数の多変数関数 $\boldsymbol{f}(\boldsymbol{x}) = \{f_i(x_j)\}(i = 1, \cdots, nf;\ j = 1, \cdots, nx)$ のヤコビ行列 $J = \{\partial f_i/\partial x_j\}$ を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dqmojx (sub, x, nx, nf, eps, mr, h, rjac, iwk, wk);

単精度関数:

ierr = ASL_rqmojx (sub, x, nx, nf, eps, mr, h, rjac, 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	関数 $\boldsymbol{f}(\boldsymbol{x})$ を定義する関数 sub(x, nx, f, nf) の関数名 (注意事項 (a) 参照)
2	x	{D∗ / R∗}	nx	入 力	微分点 $(x_1, \cdots, x_{nx})$
3	nx	I	1	入 力	独立変数 x の数 nx
4	nf	I	1	入 力	関数の連立数 nf
5	eps	{D / R}	1	入 力	要求相対精度 (既定値: (誤差判定のための単位)$^{\frac{2}{3}}$)
6	mr	I	1	入 力	最大反復数 (既定値 : 100)
7	h	{D / R}	1	入 力	最初の刻み幅
8	rjac	{D∗ / R∗}	nf×nx	出 力	ヤコビ行列 $J = \{\partial f_i/\partial x_j\}$ の値
9	iwk	I*	nf	ワーク	作業領域
10	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 (注意事項 (b) 参照) **大きさ:** nf × (5 + mr)
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) eps $\geq \varepsilon \times 64$ (既定値にするため，0.0 を入力する場合を除く)
(ε は誤差判定のための単位)

(b) mr>0 (既定値にするため，0 を入力する場合を除く)

(c) nx > 0 かつ nf > 0

(d) h $\geq \max(x_i \times \varepsilon, \varepsilon^{0.25})$ $(i = 1, \cdots, \mathrm{nx})$
(ε は誤差判定のための単位)

(5) **エラーインディケータ (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.
2000	要求精度になる前に, 誤差が大きくなった. または, 刻み幅が小さくなった.	その時点での計算値を出力して処理を打ち切る.
2500	要求精度にならず最大反復数に達した.	
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.

(6) **注意事項**

(a) 関数 sub の作り方は次のようにする.

```
void FORTRAN sub(double *x, int *nx, double *f, int *nf)
{
        f[0]= f_1(x_1, ..., x_nx);
        f[1]= f_2(x_1, ..., x_nx);
         ⋮
        f[(*nf)-1]= f_*nf(x_1, ..., x_nx);
}
```

(b) 引数第 10 項 wk の領域を配列として確保する際に, mr≤0 のときは配列の大きさを nf × 105 とする.

(c) 要求相対精度としては, 既定値よりも大きい値を入力することが望ましい.

(d) 引数第 7 項 h の値は, 0.5 ∼ 1.0 程度であるのが望ましい.

(e) 収束判定は {| 補外値 − 前回の補外値 | ≤ | eps × 補外値 |} が成立したとき収束したものとする.

(f) 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.

(g) 微分点でのヤコビ行列の各要素は配列 rjac に以下のよう格納される.
rjac$[(i-1)+\mathrm{nf}*(j-1)] = \partial f_i/\partial x_j$ $(i = 1, 2, \cdots, \mathrm{nf};\ j = 1, 2, \cdots, \mathrm{nx})$

(h) 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.

(7) 使用例

(a) 問 題

多変数関数のベクトル

$$\boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} f_1(\boldsymbol{x}) \\ f_2(\boldsymbol{x}) \\ f_3(\boldsymbol{x}) \\ f_4(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} x_1x_2x_3 \\ x_1 + x_2 + x_3 \\ x_1x_2 + x_2x_3 + x_3x_1 \\ x_1x_2x_3 + x_1x_2 + x_1 \end{bmatrix}$$

の $(x_1, x_2, x_3) = (6.8, 9.1, 3.4)$ におけるヤコビ行列を求める.

(b) 入力データ

関数名: f

x [0] =6.8, x [1] =9.1, x [2] =3.4, nx=3, nf=4, eps=1.0e−8, mr=15, h=0.5

(c) 主プログラム

```
/*      C interface example for ASL_dqmojx */

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
void   f(double *x,int *nx,double *f,int *nf)
#else
void   f(x,nx,f,nf)
double *x;
double *f;
int    *nx;
int    *nf;
#endif
{
    f[0]=x[0]*x[1]*x[2];
    f[1]=x[0]+x[1]+x[2];
    f[2]=x[0]*x[1]+x[1]*x[2]+x[2]*x[0];
    f[3]=x[0]*x[1]*x[2]+x[0]*x[1]+x[0];
}
#ifdef __cplusplus
}
#endif

int main()
{
    double *x1;
    int nx;
    int nf;
    double epsn;
    int mr;
    double h;
    double *ans;
    int *iwk;
    double *wk;
    int ierr;
    int i,j;
    FILE *fp;

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

    printf( "    *** ASL_dqmojx ***\n" );
    printf( "\n     ** Input **\n\n" );
    nx=3;
    nf=4;
    mr=15;
    x1  = ( double * )malloc((size_t)( sizeof(double) * nx ));
    if( x1  == NULL )
    {
        printf( "no enough memory for array x1\n" );
        return -1;
    }
    ans  = ( double * )malloc((size_t)( sizeof(double) * (nf*nx) ));
    if( ans  == NULL )
    {
        printf( "no enough memory for array ans\n" );
        return -1;
    }
    wk  = ( double * )malloc((size_t)( sizeof(double) * (nf*(5+mr)) ));
    if( wk  == NULL )
```

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

    for( i=0 ; i<nx ; i++ )
    {
        fscanf( fp, "%lf", &x1[i] );
    }
    fscanf( fp, "%lf", &epsn );
    fscanf( fp, "%lf", &h );
    printf( "\tNumber of Variable = %6d\n",nx );
    printf( "\tNumber of Function = %6d\n",nf );
    printf( "\n\tx   =" );
    for( i=0 ; i<nx ; i++ )
    {
        printf( " %8.3g ",x1[i] );
    }
    printf( "\n" );
    printf( "\n\teps = %8.3g\n",epsn );
    printf( "\tmr  =   %6d\n",mr );
    printf( "\th   = %8.3g\n",h );

    fclose( fp );

    ierr = ASL_dqmojx(f, x1, nx, nf, epsn, mr, h, ans, iwk, wk);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tJacobi[i][j] =\n\n" );
    for( j=0 ; j<nf ; j++ )
    {
        for( i=0 ; i<nx ; i++ )
        {
        printf( "\t%8.3g", ans[j+nf*i] );
        }
        printf( "\n" );
    }

    free( x1 );
    free( ans );
    free( wk );
    free( iwk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dqmojx ***

 ** Input **

Number of Variable =      3
Number of Function =      4

x   =      6.8       9.1       3.4

eps =    1e-08
mr  =       15
h   =      0.3

 ** Output **

ierr =      0

Jacobi[i][j] =

    30.9        23.1        61.9
       1           1           1
    12.5        10.2        15.9
      41        29.9        61.9
```

第 4 章　数値積分

4.1　概 要

本章では, 自動積分法を用いた関数の数値積分を行う関数について説明する.

数値積分には, 与えられたデータ点を補間しながら積分値を求めるものと与えられた被積分関数の積分値を求めるものとがあるが, ここでは後者を取り扱う. なお, 前者については, 第 3 章「スプライン関数」が利用できる.

関数の数値積分を行う場合, その被積分関数の性質により最適の解法を選択して解けば高速, 高精度で解が得られる. 本ライブラリでは, 次の性質に対応した関数を用意している.

(1) **任意の関数**
振動型, 端点特異型, 内点特異型, ピーク型など, どのような被積分関数でも, 性質に対する情報なしで高速に積分値を求め, さらに特異点に関する情報も出力する. 特異点が著しいときや振動が激しいときで, 高精度がどうしても得られない場合でも近似値を計算し, あわせてその誤差も出力する. より高精度を得たいときは, 特異点の情報を利用できる他の関数を利用すればよい. 最も汎用性のある関数であるが, 無限区間の振動型には不向きである. なお, 必要最小限の入出力引数だけで利用しやすい関数も設けている.

(2) $f(x) \cdot w(x)$ **の型の関数**
$w(x)$ としては, 三角関数, 端点に特異性のある代数および対数関数, $1/(x-c)$ のいずれかで, これを重みの関数として積分値を求める.

(3) **振動型, ピーク型関数**
特異性が著しくない場合に用いる. 振動型かピーク型かに対応して isw の値を選択すれば, より計算効率が上がる.

(4) **特異型関数**
特異性がなくても積分値が得られるが, 特異性が著しい場合には他の関数より効率がよい. 振動型関数に対しては利用しない方がよい. 特異点位置が端点の場合と内点の場合の関数があるが, 内点の場合は特異点位置情報が必要である.

(5) **特異型であるがその情報が不明な関数**
(1) 任意の関数に対する関数と同様にどのような被積分関数でも性質に対する情報なしで積分値を求める. (1) のものより特異性の著しい場合に適する. また, 要求精度が満たされない場合には, (1) のように近似値を求める処理には移らず, 処理が途中で打ち切られることがあり, 演算時間も多く必要とする. したがって, この関数は特異型であることが分かっているが特異点位置が不明か精度よく得られない場合で, 精度よく積分値を求めたい場合に利用する.

4.1.1 使用上の注意

(1) 各関数の使い分けを流れ図を使い説明する.

図 4−1　有限区間関数利用図

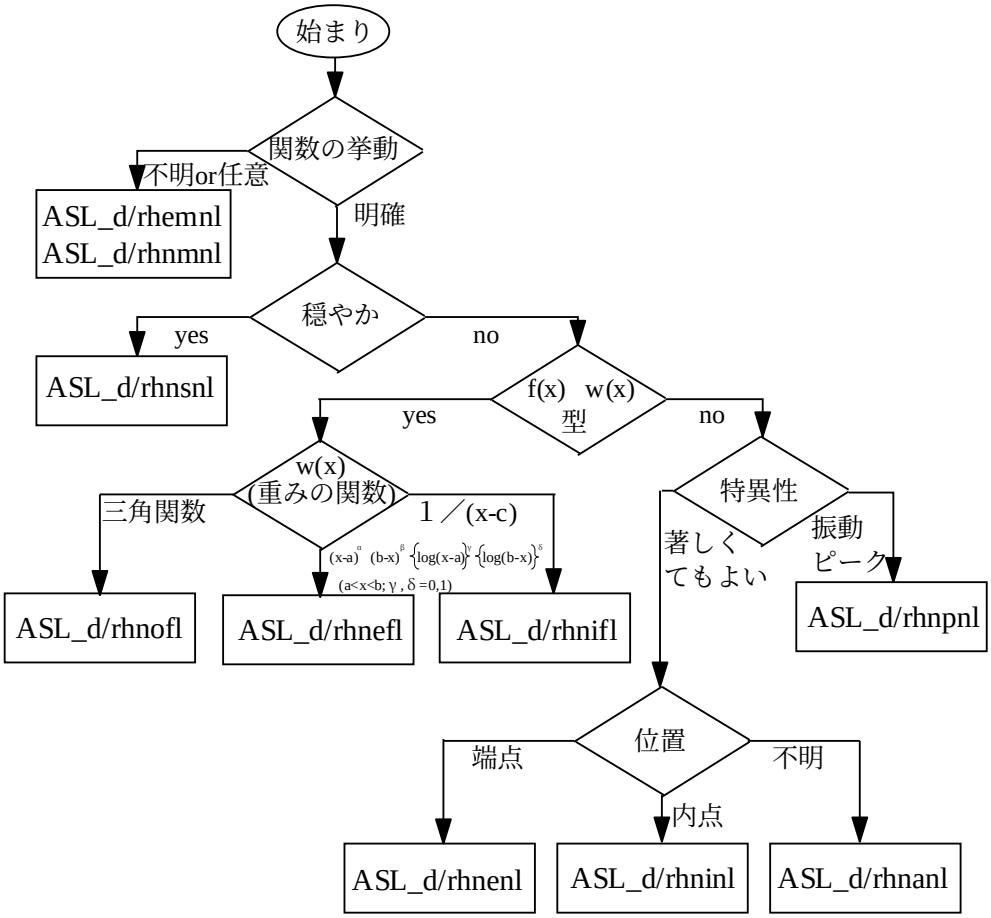
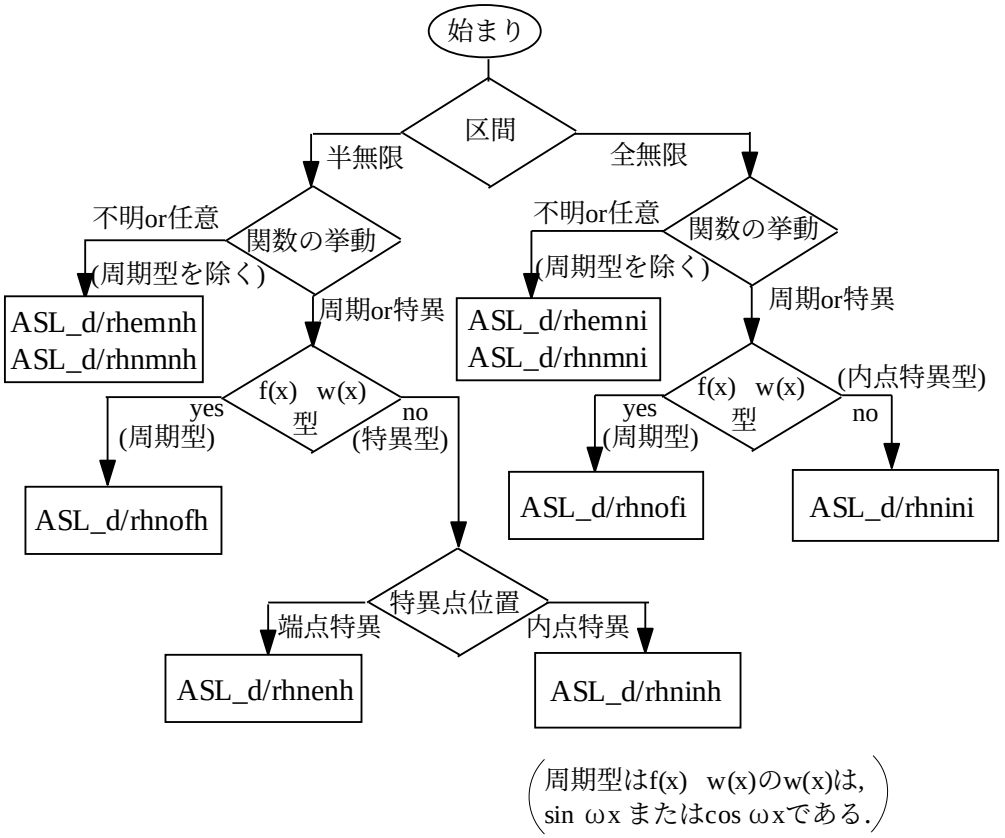


図 4−2　無限区間関数利用図



(2) これらの関数を使用するプログラム内では, 次の注意が必要である.

① 積分区間内では, オーバフローが生じないよう対策をとる必要がある. (たとえば特異点での関数値を 0.0 とする).

例 1 次元積分

```
/* C interface example for ASL_dhemnl */
#include <stdio.h>
#include <math.h>
#include <asl.h>
```

・関数 f

```
double FORTRAN f( double *x )··· 主プログラムにおける第 1 引数 f と同じ名前とする.
{
 static double uf=0.1e-77;
 if(fabs(*x-1) < uf)
     return 0.0;
 else
     return 1.0/sqrt(fabs(*x-1));
}
```

ゼロ割りによるオーバフロー対策
(オーバフローが生じないときは不要)

・主プログラム

```
int main()
{
        ≀
    ierr = ASL_dhemnl(f, a, b, epsrel, &result, &abserr);
        ≀
    return 0;
}
```

例 2 次元積分

```
/* C interface example for ASL_dhnfnm */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <asl.h>
```

・関数 f

```
double FORTRAN f(double *x, double *y) ··· 主プログラムにおける第 1 引数 f と同じ名前とする.
{
    ≀
}
```

・関数 a

```
double FORTRAN a(double *y) ··· 主プログラムにおける第 2 引数 a と同じ名前とする.
{
    ≀
}
```

・関数 b

```
double FORTRAN b(double *y) ··· 主プログラムにおける第 3 引数 b と同じ名前とする.
{
    ≀
}
```

・主プログラム

```
int main()
{
        ≀
    ierr = ASL_dhemnl(f, a, b, c, d, epsr, epsa, ···);
        ≀
}
```

例 多次元積分

```
/* C interface example for ASL_dhnfml */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <asl.h>
```

・関数 f

```
double FORTRAN f(double *x, int *m) ··· 主プログラムにおける第 1 引数 f と同じ名前とする.
{
    return(x[0], x[1], ···, x[*m-1] を変数とした式)
}
```

・関数 r

```
double FORTRAN r(int *i, double *x, double *a, double *b, int *m) ··· 主プログラムにおける第 2 引数 r と同じ名前とする.
{
switch(*i)
   {
   case 1:
    a[0]= ···  }
    b[0]= ···  }  (x[1], x[2], ···, x[*m-1] を変数とした式)
   case  2:
    a[1]= ···  }
    b[1]= ···  }  (x[2], ···, x[*m-1] を変数とした式)
      ⋮
   case  *i が*m のとき
    a[*m-1 番目]= ···  }
    b[*m-1 番目]= ···  }  (定数)
   }
```

```
    }
```

・主プログラム

```
int  main()
{
        ≀
    ierr = ASL_dhemnl(f, r, m, epsr, epsa, ···);
        ≀
}
```

(3) エラーインディケータ (戻り値) は, いくつかのエラーが発生したとき, 最も重大なエラー値が出力され, 他のエラー情報が隠れてしまう可能性がある.

(4) 各関数は, 入力された要求相対精度や要求絶対精度に応じた精度で積分値を計算する. また, 出力として積分値の他に絶対誤差推定値も求める.

注意 要求相対精度, 要求絶対精度, 絶対誤差推定値の定義は以下の通り.

数値的に求められる積分値の近似値を Q, 厳密解を Q_0 とおく. このとき絶対誤差および相対誤差を以下の式で定義する.

$$\begin{cases} (\text{絶対誤差}) & = & |\,Q - Q_0\,| \\ (\text{相対誤差}) & = & |\,(Q - Q_0)/Q\,| \end{cases}$$

要求相対精度 ER および要求絶対精度 EA とは, それぞれ相対誤差および絶対誤差として要求上許容できる値の上限を指す. 絶対誤差推定値 AE とは, 数値的な誤差評価方法により見積られる絶対誤差の上限値である.

要求相対精度 ER および/または要求絶対精度 EA が入力値として与えられた場合, 関数は以下の条件を満たす精度で積分値の近似値 Q を求めようとする.

$$\begin{cases} AE/Q & < & ER \\ AE & < & EA \end{cases}$$

(5) 多次元有限区間積分の関数を除く本章のすべての関数では, 要求絶対精度の既定値で使われる値は (絶対値最小値)$\times 2^{24}$ とする.

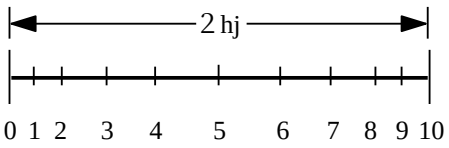
4.1.2 使用しているアルゴリズム

4.1.2.1 適応型ニュートン・コーツ則 (任意の関数の積分)

ニュートン・コーツ 9 点則を基本とした適応型自動積分法で誤差推定の強化, 収束判定の緩和, 特異点の検出と処理の機能を追加している.
自動積分法は, 被積分関数 $f(x)$, 積分区間 $[a,b]$, 要求絶対精度 ε_0 が与えられたとき, 積分区間の分割を自動的に行い, 許容誤差内の精度の積分値を計算する算法である. この分割を制御するために, 各細分区間の積分値とその誤差, および収束判定が必要となる.
ここではそれぞれを次のようにして求める.

(1) 積分値とその誤差
自動分割された区間 j における積分値 Q_j とその補正値 ξ_j については, 文献 (1) に詳細が記されており, ここでもその手法を利用する. 例えば, 9 点則では区間幅を $2h_j$ として図 4−3 のように分割したとき, Q_j と ξ_j は次式のように表せる.

図 4−3 内分点のとり方 (例 9 点則)



$$Q_j = \frac{h_j}{14175}\{989(f_0+f_{10})+5888(f_2+f_8)-928(f_3+f_7) +10496(f_4+f_6)-4540f_5\}$$

$$\xi_j = \frac{-4736h_j}{468242775}\{3003(f_0+f_{10})-16384(f_1+f_9)+27720(f_2+f_8) -38220(f_3+f_7)+56056(f_4+f_6)-64350f_5\}$$

ニュートン・コーツ n 点則を利用した場合, $Q_j+\xi_j$ のもつ誤差 ε_j は, $n+1$ 次の微係数がほぼ一定とすると, $\varepsilon_j = \mid \xi_j \mid /(2^{n+1}-1)$ となる. しかし, 現実には, $n+1$ 次以上の微係数を無視することができず, これより大きくなることが多い. ある区間を i, それをさらに 2 分して得られるそれぞれの区間を j, $j+1$ として, ξ_i と ξ_j より ε_j を推定するには, 次の方法を用いる.
$\varepsilon + \mid \xi \mid$ と $\mid \xi \mid$ の比を一定として,

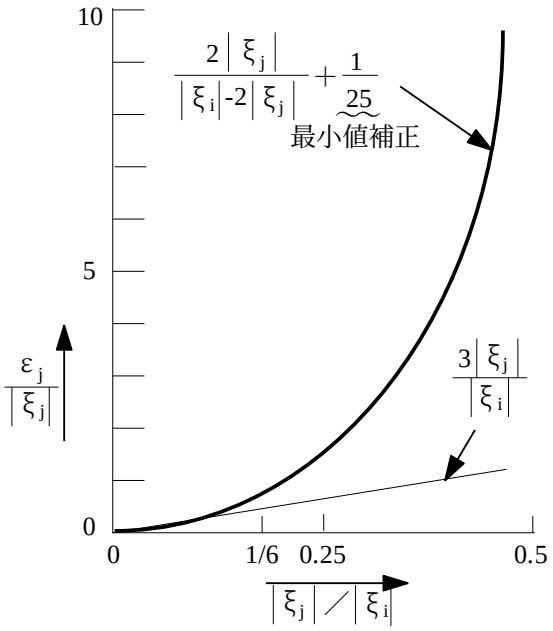
$$C = \frac{\mid \xi_j + \xi_{j+1} \mid}{\mid \xi_i \mid}$$

($n+1$ 次の微係数が一定のとき, $C = \frac{1}{2^{n+1}}$.) 隣り合う区間で関数の性質がほぼ等しいとすると, $\xi_j \simeq \xi_{j+1}$ とすることができるので, これより次式を得る.

$$\varepsilon_j = \frac{C}{1-C} \mid \xi_j \mid = \frac{2\xi_j^2}{\mid \xi_i \mid -2 \mid \xi_j \mid}$$

この ε_j の値を全区間に対して加算したものを全体の誤差量とする.
なお, このプログラム上では, ε_j 最小値を $\mid \xi_j \mid /25$(単精度では, $\mid \xi_j \mid /8$) でおさえ, 誤差の推定が小さすぎないようにする. また特異点付近のように $\mid \xi_i \mid \leq \mid \xi_j \mid \times 6$ と ξ の減少率の小さい所では, 誤差が過大に評価されることを防ぐため, $1/(1-C)=1.5$ として, $\varepsilon_j = 3C \mid \xi_j \mid /2 = 3\xi_j^2/\mid \xi_i \mid$ とする (図 4−4 参照). また, 収束判定での ε_j 推定値は, 安全性を考え $\varepsilon_j = \mid \xi_j \mid /2.0$ と大きい値を推定する.

図 4–4　ε_j の値の補正



実際にとるべき $|\xi_j|/|\xi_i|$ が 0.25 を越えると一般に収束せず, 特異点として処理されるべきであるが, この付近では, $|\xi_j|/|\xi_i|$ の変動が著しく, 収束時の $|\xi_j|/|\xi_i|$ の値が 1.0 に近いこともある. この場合は誤差 ε_j を大きく見積もりすぎるので, ほぼ $|\xi_j|/|\xi_i| = 0.25$ での誤差に補正するため, $|\xi_j|/|\xi_i| \geq 1/6$ では図の点線で誤差を近似する.

(2) 各区間の収束判定方法

$h_0 = (b-a)/2$, $h_j =$ (細分区間 j の幅)$/2$ $(j = 1, \cdots, n)$ とする.

収束したとみなされた解の誤差が $[-\varepsilon_j, \varepsilon_j]$ 上に一様な確率密度 $g(x)$ をもつとすると, $g(x) = 1/2\varepsilon_j$ となる. このときこの区間での分散は $\varepsilon_j^2/3$, したがって, 全区間の分散は, $\sum_{j=1}^{n} \varepsilon_j^2/3$ となる. $\xi_j = \sqrt{h_j/h_0} \cdot \varepsilon_0$ (ε_0 は $[a, b]$ の積分値の要求絶対精度) とすると, 全区間の積分値に対する分散は, $\sigma^2 = \varepsilon_0^2/3$ となるため, 誤差は $\sigma = \varepsilon_j/\sqrt{3}$ の正規分布をなし, $-\varepsilon_0$ から $+\varepsilon_0$ の範囲に入る確率は 91.6%となる. したがって ε_j を (1) で求めた $|\xi_j|/2.0$ とし, 次式が成立したとき分割を終了し $Q_j + \xi_j$ をその区間の積分値とする.

$$|\xi_j|/2.0 < \sqrt{h_j/h_0} \cdot (\max(\varepsilon_0, \varepsilon_0' \cdot I))$$

ε_0' : $[a, b]$ の積分値の要求相対精度

I : 全積分値推定値

$$\left[\begin{array}{l} Q_1\text{を全区間をニュートン・コーツ 9 点則で求めた積分値として } I = Q_1 + \xi_1 + \xi_2 \\ + \cdots + \xi_n + q \qquad \text{(特異点処理を行ったときの補正量)} \end{array} \right]$$

(3) 特異点が分割点にあるときの処理

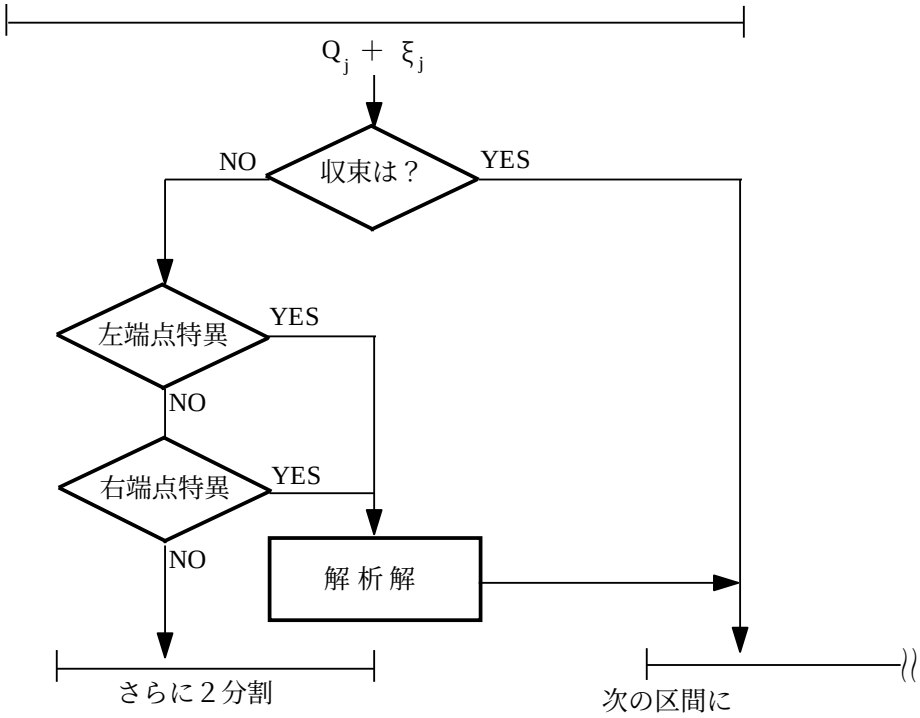
今, 特異点が適当な正の整数 m に対して, 全区間の 2^m 等分点のどこかに位置しておれば, ある段階より区間の端にくる.

特異点の検出は, 図 4–5 のフローにしたがい, 内分点での関数値を利用して行う. このとき, 検出操作は区間幅が全体の 2^{-l}(l は 5 の倍数) となるときに行う.

特異点の検出および処理は次式にしたがう.

- 代数型特異 $(f(x) = \alpha X^{-p} + q)$

図 4–5　計算手順



左端特異

$$\frac{f_2 - f_3}{f_3 - f_5} \simeq \frac{f_3 - f_5}{f_5 - f_{10}} = d \text{ が成立}$$

解析解 I_j

$$I_j = \frac{2h_j \cdot (f_{10} - p \cdot q)}{(1-p)},\ p = \frac{\log(d)}{\log(2)},\ q = \frac{(2^{-p} \cdot f_5 - f_{10})}{(2^{-p} - 1)}$$

右端特異

$$\frac{f_8 - f_7}{f_7 - f_5} \simeq \frac{f_7 - f_5}{f_5 - f_0} = d \text{ が成立}$$

解析解 I_j

$$I_j = \frac{2h_j \cdot (f_0 - p \cdot q)}{(1-p)},\ p = \frac{\log(d)}{\log(2)},\ q = \frac{(2^{-p} \cdot f_5 - f_0)}{(2^{-p} - 1)}$$

- 対数型特異

左端特異

$$f_2 - f_3 \simeq f_3 - f_5 \simeq f_5 - f_{10} = d \text{ が成立}$$

解析解 I_j

$$I_j = 2h_j \cdot (f_5 + d \cdot (1 - \log(2))/\log(2))$$

右端特異

$$f_8 - f_7 \simeq f_7 - f_5 \simeq f_5 - f_0 = d \text{ が成立}$$

解析解 I_j

左端特異の式と同じ

代数型, 対数型ともに $\simeq$ 式の判定は, その左辺と右辺の差を δ として,

$$\delta \leq \varepsilon_0' / \sqrt{10 h_j / h_0} (\varepsilon_0' : \text{要求相対精度})$$

が成立したときに, 特異点処理を行うものとする. またその誤差 e_j は

$$e_j = | \, (| \, \delta \, | + \text{誤差判定のための単位}) \cdot I_j \, |$$

とする.

(4) (3) で検出されなかった特異点の処理

特異点が (3) の方法で検出されなかったときは, $y = \alpha x^p (0 < p < 1)$ の代数型特異で近似した値を利用する. すなわち, 絶対値最大の値をとる点を k とすると $f_k = 0$ ならば,

$$\delta_s = E_s \cdot \frac{(p+1)}{(2 - 2^{-p}) \cdot (1-p)}$$

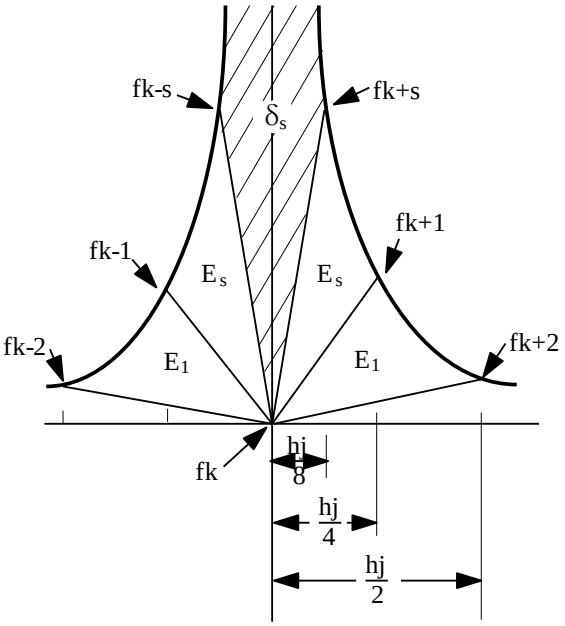
となり, 端点でも内点でも次式が成立する.

$$p = \log(2E_s / E_1) / \log(2)$$

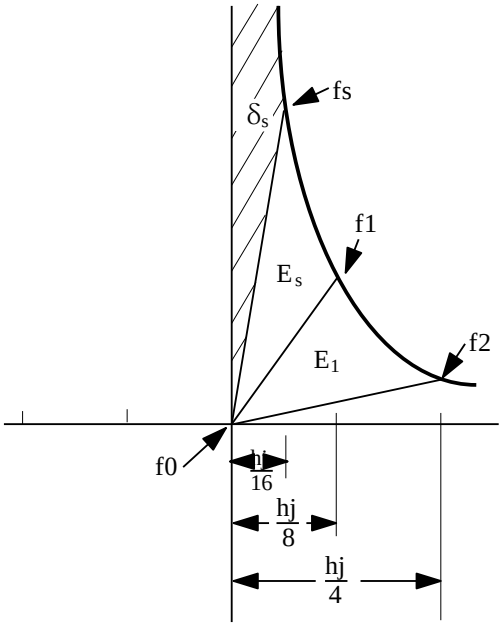
ここで, E_1, E_s は関数値より解析的に計算する. f_{k+s} や f_{k-s} は, δ_s を計算するために用いられる関数値である (図 4−6(a), (b) 参照).

図 4−6　特異点の処理

(a) 内点特異点の場合



(b) 端点特異の場合

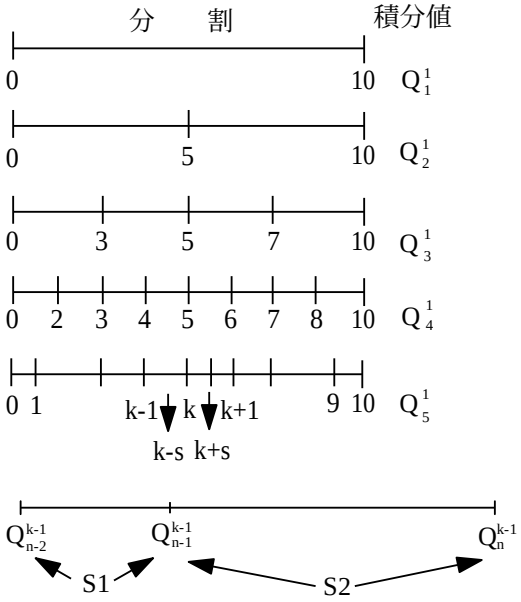


積分値は, δ_s を加えない部分の台形則積分値と, 解析的に求めた δ_s を加えて計算する. また, ここでの誤差は, δ_s の部分を台形則で求めた値との差とする.

なお, (3) でも (4) でも検出されなかった特異点は, エイトキンの補外を改良したアルゴリズムでできるだけ真値に近い値を得るようにする.

- エイトキンの補外の改良
  次の図のように, 各分割に対する積分値を $Q_n^1 \quad (n = 1 \cdots 5)$ とする.

図 4−7　各分割とその積分値



($S2$ が $S1$ に比較して大きい程, Q_n^{k-1} 側の重みを小さくし, Q_{n-1}^{k-1} 側に近い値を返す.)

次の補外式を用い, $Q_n^k (k = 1, 2, 3; n = 2 \cdot k - 1, \cdots, 5)$ のテーブルを作成する.

$$Q_n^k = Q_n^{k-1} - \frac{(Q_n^{k-1} - Q_{n-1}^{k-1})^2}{Q_n^{k-1} - 2 \cdot Q_{n-1}^{k-1} + Q_{n-2}^{k-2}}$$

ただし, $|\ Q_n^{k-1} - Q_{n-1}^{k-1}\ |$ $\geq$ $|\ Q_{n-1}^{k-1} - Q_{n-2}^{k-1}\ |$ のときは,
$Q_n^k = (Q_n^{k-1} + m \cdot Q_{n-1}^{k-1})/(m+1)$ (m : 重み) で, Q_n^k を求める. 最後に Q_5^3 を解とし, 誤差は $|\ Q_5^3 - Q_5^1\ |$ とする. このとき重みは, 次の計算により求める.

$$m = |\ \frac{Q_n^{k-1} - Q_{n-1}^{k-1}}{Q_{n-1}^{k-1} - Q_{n-2}^{k-1}}\ | \cdot \frac{1}{15}$$

(5) 無限区間に対する応用
変数変換を利用して [0, 1] 区間で計算する.

$$\int_a^\infty f(x)dx = \int_0^1 \frac{F(t)}{t}^2 dt, \qquad F(t) = f(a + \frac{1-t}{t})$$

$$\int_{-\infty}^\infty f(x)dx = \int_0^1 \frac{F(t) + G(t)}{t^2} dt, \quad F(t) = f(\frac{1-t}{t}), \quad G(t) = f(\frac{t-1}{t})$$

(6) 2 重積分に対する応用
領域が矩形の場合は,

$$I = \int_c^d \int_a^b f(x,y)dxdy$$

において $F(y) = \int_a^b f(x,y)dx$ として $F(y)$ を先に求めて, これを利用しながら $\int_c^d F(y)dy$ を計算する.
矩形領域でない場合で,

$$I = \int_c^d \int_{a(y)}^{b(y)} f(x,y)dxdy$$

の積分は, $A = a(y), B = b(y), F(y) = \int_A^B f(x,y)dx$ として, $F(y)$ を求めながら $\int_c^d F(y)dy$ を計算する.

4.1.2.2 ガウス–クロンロッドの方法

一群のガウス公式を使うときの問題点は, 標本点が球多項式の零点であるために, 次式を増加させたとき前に計算した関数値を再利用できないことである. そこでクロンロッドは n 点のガウス則に $n+1$ 点を追加し, $2n+1$ 個の点に基づく次数 $3n+1$ の積分則を作った. ただし, n 点のガウス則の重みは保存されないので, $2n+1$ 点に対しては新たな重みを利用する必要がある (重みおよび分点は参考文献 (6) 参照).
非適応型では, 10 点ガウス則よりはじめ, 21 点, 43 点, 87 点, 175 点, と点数を増やしたクロンロッド則により積分値を更新し, その更新される値の差が要求精度以下になれば収束したものとみなし更新された値を積分値として返す.
適応型自動積分では, 適応型ニュートン・コーツ則同様に積分区間の誤差が許容精度内になるまで細分を続け, 各細分区間の積分値を加算して全積分値とする. このときの積分値および誤差はガウス–クロンロッド 7-15 点, 10-21 点, 15-31 点, 20-41 点, 25-51 点, 30-61 点の対に対する公式を利用して計算する. また特異点情報がない場合では, この 10-21 点公式を基本とし, $\varepsilon-$ アルゴリズムの積分値の補外を行う.
なお, 三角関数や代数, 対数関数を重みの関数とする場合で, 特異点以外の所や振動の著しくない部分は 7-15 点の対に対する適応型自動積分を行うが, 特異点を含む区間や振動の激しい区間にはクレンショーカーチス則を適用する.
ガウス–クロンロッドの方法での区間 j での誤差 e_j を推定するには,

$$e'_j = \int_j |\ f(x) - Q_j/h_j\ | dx$$

$$e_j = e'_j \cdot \min\{1, \xi_j/e'_j\}$$

として求める. ただし, Q_j, ξ_j はそれぞれ区間 j での積分値とその補正値である.

4.1.2.3 クレンショーカーチス法 (重みの関数をもつ関数)

被積分関数 $f(x)$ をチェビシェフ多項式によって近似し, この展開された多項式を積分する. これより積分値を求める方法がクレンショーカーチス法である.
関数 $f(x)$ のチェビシェフ多項式による展開は次のようになる.

$$f(x) = \frac{1}{2}a_0 + a_1T_1(x) + a_2T_2(x) + \cdots$$

係数 a_k はフーリエ・チェビシェフ係数であり, 次の公式で与えられる.

$$a_k = \frac{2}{\pi}\int_{-1}^{1}\frac{f(x)T_k(x)}{(1-x^2)^{1/2}}dx = \frac{2}{\pi}\int_0^{\pi} f(\cos(\theta))\cdot\cos(k\theta)d\theta$$

a_k の計算は台形則で近似できる.

$$\begin{aligned} a_k \simeq \alpha_k &= \frac{1}{N}(f(x_0)T_k(x_0) + 2\sum_{j=1}^{N-1} f(x_j)T_k(x_j) + f(x_N)T_k(x_N)) \\ &= \frac{1}{N}(f(1) + 2\sum_{j=1}^{N-1} f(\cos(\frac{\pi j}{N}))\cdot\cos(\frac{\pi kj}{N}) + f(-1)\cdot(-1)^k) \end{aligned}$$

$f(x)$ の多項式を第 N 項で打ち切ると次式を得る.

$$\begin{aligned} f(x) &\simeq \frac{1}{2}a_0 + a_1T_1(x) + a_2T_2(x) + \cdots + a_NT_N(x) \\ &\simeq \frac{1}{2}\alpha_0 + \alpha_1T_1(x) + \cdots + \alpha_{N-1}T_{N-1}(x) + \frac{1}{2}\alpha_NT_N(x) \end{aligned}$$

ここで α_k は FFT(高速フーリエ変換) で求めることができる. 特に $d = 12$ のときは, 特に単純で $Tolstov(1962)$ により処理方法が示されている. ここでは, $d = 12$ と $d = 24$ を利用している.
関数に重みの関数 $w(x)$ がついた場合のチェビシェフ展開後の積分値は, 次のようになる.

$$\int_{-1}^{1} w(x)f(x)dx = \frac{\alpha_0}{2}\int_{-1}^{1} w(x)dx + \sum_{k=1}^{N-1}\alpha_k\int_{-1}^{1} w(x)T_k(x)dx + \frac{\alpha_N}{2}\int_{-1}^{1} w(x)T_N(x)dx$$

- $w(x)$ が $\sin\omega x$ のとき

$$\begin{aligned} \int_{c_1}^{c_2} w(x)f(x)dx \simeq\ & \frac{1}{2}(c_2-c_1)\Big[\frac{\alpha_0}{\lambda}\sin\Big(\frac{(c_1+c_2)\omega}{2}\Big)\cdot\sin(\lambda) \\ & + \sum_{k=1}^{N-1}\alpha_k\Big\{\cos\Big(\frac{(c_1+c_2)\omega}{2}\Big)\int_{-1}^{1}\sin(\lambda x)\cdot T_k(x)dx \\ & + \sin\Big(\frac{(c_1+c_2)\omega}{2}\Big)\int_{-1}^{1}\cos(\lambda x)\cdot T_k(x)dx\Big\} \\ & + \frac{\alpha_N}{2}\Big\{\cos\Big(\frac{(c_1+c_2)\omega}{2}\Big)\int_{-1}^{1}\sin(\lambda x)\cdot T_N(x)dx \\ & + \sin\Big(\frac{(c_1+c_2)\omega}{2}\Big)\int_{-1}^{1}\cos(\lambda x)\cdot T_N(x)dx\Big\}\Big] \end{aligned}$$

ただし, $\lambda = (c_2-c_1)\omega/2$ とする. ここで, $\int_{-1}^{1}\sin(\lambda x)\cdot T_k(x)dx$ を $S_k(\lambda)$ として, $\int_{-1}^{1}\cos(\lambda x)\cdot T_k(x)dx$ を $C_k(\lambda)$ とすると, $S_k(\lambda)$, $C_k(\lambda)$ は次の漸化式で表せる.

$$\begin{aligned} &\lambda^2(k-1)(k-2)S_{k+2}(\lambda) - 2(k^2-4)(\lambda^2-2k^2+2)S_k(\lambda) + \lambda^2(k+1)(k+2)S_{k-2}(\lambda) \\ &\quad = -8(k^2-4)\sin(\lambda) - 24\lambda\cos(\lambda) \end{aligned}$$

$$\begin{aligned} S_1(\lambda) &= 2(\sin(\lambda) - \lambda\cos(\lambda))\lambda^{-2} \\ S_3(\lambda) &= \lambda^{-2}\sin(\lambda)(18-48\lambda^{-2}) + \lambda^{-1}\cos(\lambda)(48\lambda^{-2}-2) \end{aligned}$$

$$\lambda^2(k-1)(k-2)C_{k+2}(\lambda) \quad - \quad 2(k^2-4)(\lambda^2-2k^2+2)C_k(\lambda) \quad + \quad \lambda^2(k+1)(k+2)C_{k-2}(\lambda)$$
$$= 24\lambda\sin(\lambda) - 8(k^2-4)\cos(\lambda)$$

$$\begin{aligned} C_0(\lambda) &= 2\lambda^{-1}\sin(\lambda) \\ C_2(\lambda) &= 8\lambda^{-2}\cos(\lambda) - \lambda^{-3}(2\lambda^2-8)\sin(\lambda) \\ C_4(\lambda) &= 32\lambda^{-4}(\lambda^2-12)\cos(\lambda) + 2\lambda^{-5}(\lambda^4-80\lambda^2+192)\sin(\lambda) \\ S_{2k}(\lambda) &= C_{2k+1}(\lambda) = 0(k=0,1,2\cdots) \end{aligned}$$

$w(x)$ が $\cos(\omega x)$ のときも同様に計算できる．なお, $\lambda \leq 2$ のときはこの方法よりガウス–クロンロッド則を利用した方が有効である．

- $w(x)$ が代数型や対数型の端点特異をなす関数のとき
  重みの関数を $u(x)$ とすると次のように表せる．

$$u(x) = (x-a)^\gamma(b-x)^\delta\{\log(x-a)\}^\mu\{\log(b-x)\}^\nu(a \leq c_1 < c_2 \leq b, \mu, \nu = 0, 1)$$

  c_1 が端点で $c_1 = a$ のとき

$$\begin{aligned} \int_a^{c_2} u(x)f(x)dx \quad &\simeq \quad \Big(\frac{c_2-a}{2}\Big)^{\gamma+1}\sum_{i=0}^{\mu}\{\log(c_2-a)\}^{\mu-i} \\ &\cdot\left[\frac{\alpha_0}{2}\int_{-1}^{1}(1+x)^\gamma\left\{\log\Big(\frac{1+x}{2}\Big)\right\}^i dx + \sum_{k=1}^{N-1}\alpha_k G_{k,i}(\gamma) + \frac{\alpha_N}{2}G_{N,i}(\gamma)\right] \end{aligned}$$

  ここで

$$G_{k,i}(\gamma) = \int_{-1}^{1}(1+x)^\gamma\cdot\{\log(\frac{1+x}{2})\}^i\cdot T_k(x)dx, \quad k = 1,\cdots,N; i = 0,1$$

  c_2 が端点で $c_2 = b$ のとき

$$\begin{aligned} \int_{c_1}^{b} u(x)f(x)dx \quad &\simeq \quad \Big(\frac{b-c_1}{2}\Big)^{\delta+1}\sum_{i=0}^{\nu}\{\log(b-c_1)\}^{\nu-i} \\ &\cdot\left[\frac{\alpha_0}{2}\int_{-1}^{1}(1+x)^\delta\left\{\log\Big(\frac{1+x}{2}\Big)\right\}^i dx + \sum_{k=1}^{N-1}\alpha_k H_{k,i}(\delta) + \frac{\alpha_N}{2}H_{N,i}(\delta)\right] \end{aligned}$$

  ここで

$$H_{k,i}(\delta) = (-1)^k G_{k,i}(\delta), \quad k = 1,\cdots,N; i = 0,1$$

- $w(x)$ が $1/(x-c)$ の内点特異をもつとき
  $v(x) = 1/(x-c)$ として

$$\int_{c_1}^{c_2} v(x)f(x)dx \simeq \frac{\alpha_0}{2}\log|\frac{c'-1}{c'+1}| + \sum_{k=1}^{N-1}\alpha_k V_k(c') + \frac{\alpha_N}{2}V_N(c^{'})$$

  ここで $c_1 < c < c_2$ とすると

$$V_k(c') = \oint_{-1}^{+1}\frac{T_k(x)}{x-c'}dx, \quad c' = (2c-c_2-c_1)/(c_2-c_1)$$

この $V_k(c')$ は, 次の漸化式を解くことにより求められる.

$$V_{k+1}(c') - 2c^{'}V_k(c^{'}) + V_{k-1}(c^{'}) = \begin{cases} 0 & k:\text{奇数} \\ 4/(1-k^2) & k:\text{偶数} \end{cases}$$

$$V_0(c') = \log \mid (1-c^{'})/(1+c^{'}) \mid$$

$$V_1(c') = 2 + c^{'}V_0(c^{'})$$

なお, c 点では細分されないようにして適応型積分を行う.

4.1.2.4 $\varepsilon-$ アルゴリズム

与えられた数列 $a_n(n=0,1,\cdots)$ に対して

$$\begin{aligned} \tau_n^{(-1)} &= 0 \qquad (n=0,1,2\cdots) \\ \tau_n^{(0)} &= a_n \qquad (n=0,1,2\cdots) \end{aligned}$$

とおき, それから出発して $k=1,2,3\cdots$ の順に

$$\tau_n^{(k)} = \tau_{n-1}^{(k-2)} + \frac{1}{\tau_n^{(k-1)} - \tau_{n-1}^{(k-1)}} \qquad (n=k,k+1,\cdots)$$

により次々と $\tau_n^{(1)}, \tau_n^{(2)}, \cdots$ を作っていく. こうして $\tau_n^{(2)}, \tau_n^{(6)}, \tau_n^{(8)}, \cdots$ の数列を作ればこれは, a_n より速く $a_m(m=\infty)$ の値に収束する.

$$\begin{array}{cccc} \tau_1^{(0)} & & & \\ \tau_2^{(0)} & \tau_2^{(1)} & & \\ \tau_3^{(0)} & \tau_3^{(1)} & \tau_3^{(2)} & \\ \tau_4^{(0)} & \tau_4^{(1)} & \tau_4^{(2)} & \tau_4^{(3)} \\ \vdots & \vdots & \vdots & \vdots \end{array}$$

この方法は, 特異点情報がない場合や振動型関数の場合において, 適応型ガウス–クロンロッドの 10-21 点法や 7-15 点法を利用した各段階の積分近似値を補外して, より真に近い近似値を得るのに利用される. また, 振動型関数の無限区間積分では, 有限区間の数サイクル分までの積分値から補外するために利用される. (特異点情報がない場合や振動型の場合で $\varepsilon-$ アルゴリズムを適用するには, 最小の細分区間がさらに 2 分され更新されていくときとする.)

4.1.2.5 2 重指数関数型公式 (内点, 端点特異型関数の積分)

積分区間の端点に特異性のある積分や, 半無限, 全無限区間の積分に有効な方法であり, 変数変換により速く減少する $(-\infty,\infty)$ 区間の関数 $f(\phi(u))\phi'(u)$ に変換し, これを適当に打ち切った範囲で台形則により積分値を求める. さらに, きざみ幅を 1/2 にしながら積分値を要求精度に達するまで更新する. その方法は次のようになる.

(1) 最初のきざみ幅 $h_0=0.25$ と定め, $W=f(\phi(0))\cdot\phi'(0)$ とする.

(2) $n=1,2,3,\cdots$ について

$$W_n = W_{n-1} + f(\phi(nh_0))\phi'(nh_0) + f(\phi(-nh_0))\phi'(-nh_0)$$

とし, 加える項が次のようになるところで打ち切る.

$$\mid W_n - W_{n-1} \mid < \max(\mid W_n \mid \times(\text{要求相対精度}), (\text{要求絶対精度})/(\frac{\pi}{2}\cdot h_0)) \times 0.01$$

打ち切った両端を $-Nh_0$, Mh_0 とおき, $Q_0 = h_0 W_n$ とする.

(3) きざみ幅 h_i を $\frac{1}{2}h_{i-1}$ とし, $[-Nh_0, Mh_0]$ 区間の台形則積分値 $Q_i(i = 1, \cdots, 10)$ を求める. また $Q_i - Q_{i-1} = \xi_i$ とする.

$$4\xi_i^2/(\mid \xi_{i-1} \mid - \mid \xi_i \mid) + \mid \xi_i \mid < \max(\text{要求相対精度} \times Q_i, \text{要求絶対精度})$$

が成立すれば Q_i を積分値, $4\xi_i^2/(\mid \xi_{i-1} \mid - \mid \xi_i \mid) + \xi_i$ をその絶対誤差として返す.

(4) (b) で打ち切った両端の外側の積分値を, 最後に加えた項とその前回に加えた項の値より代数的に補外して推定し, 積分値に加え, その絶対値を絶対誤差値に加える.

なお, 収束判定方法と誤差計算式は, 特異型関数の積分を考慮した. すなわち適応型ニュートン・コーツ則の非適応型台型則に対応して $\varepsilon_i = \xi_i^2/(\mid \xi_{i-1} \mid - \mid \xi_i \mid)$ を得る. ここで, 収束率 C が常に一定であれば, これにより収束判定をすることができるが, 実際は一定でなく ε_i の値が実際より小さい値のときに収束したものとされる. したがって真の誤差より大きく誤差を見積もり, 安全な収束判定をするために, ε_i の計算結果に安全率として 4. を乗じ, さらに $\mid \xi_i \mid$ を加えて誤差を見積もった. これにより, 誤差を $\mid \xi_i \mid$ とする一般的な方法による誤収束 (実際の誤差が推定誤差より大きく, 要求精度に達しないままで計算が終了する) をまぬがれることができる.
さらに, 有限区間積分で桁落ち防止変換をしない場合は nh が大きいとき, (たとえば単精度で 2.5 程度, 倍精度で 3.3 程度) $\phi(nh) = 1.0$ で $f(\phi(nh))$ の端点 ± 1.0 に特異性がある場合はここでの関数値が不正確になる. したがって変換後の積分が正確に求めることができたとしても変換時の誤差が大きくこれも積分値の誤差に考慮しなければならない. 本プログラムはできるだけ端点に近い所まで関数値を計算させるとともに変換時の誤差が大きいほど (c) の収束がおそくなることからこの誤差を推定し, 絶対誤差推定値に加えて出力する.
特異性のない関数に対する以上の収束判定や誤差推定方法は, やや誤差を大きく見積る程度で問題なく積分することができる.
ここで $\phi(nh)$ は次のとおりである.
$\int_{-1}^{1} f(x)dx$ のとき

$$x_n = \phi(nh) = \tanh(\frac{\pi}{2}\sinh(nh))$$

$$Q_h = \frac{\pi}{2}h \sum_{n=-\infty}^{\infty} f(\tanh(\frac{\pi}{2}\sinh(nh)))\frac{\cosh(nh)}{\cosh^2(\frac{\pi}{2}\sinh(nh))}$$

$\int_{0}^{\infty} f(x)dx$ のとき

$$x_n = \phi(nh) = \exp(\frac{\pi}{2}\sinh(nh))$$

$$Q_h = \frac{\pi}{2}h \sum_{n=-\infty}^{\infty} f(\exp(\frac{\pi}{2}\sinh(nh)))\cosh(nh)\exp(\frac{\pi}{2}\sinh(nh))$$

$\int_{0}^{\infty} f(x)dx$ で $f(x)$ が $\exp(-x)$ の形の因数を含んでいるとき

$$x_n = \phi(nh) = \exp(\frac{\pi}{2}(nh - \exp(-nh)))$$

$$Q_h = \frac{\pi}{2}h \sum_{n=-\infty}^{\infty} f(\exp\frac{\pi}{2}(nh - \exp(-nh)))(1 + \exp(-nh))\exp\frac{\pi}{2}(nh - \exp(-nh))$$

なお, 内点特異型関数に対しては, 特異点で区間分割し, 各区間を端点特異として解き, その積分値を合計して全体の積分値を求めるものとする.

注 意 桁落ち防止について
積分 $\int_{-1}^{1} f(x)dx$ で $f(x)$ の分母に, $(1+x)^{\alpha}, (1-x)^{\beta}, (0<\alpha,\beta<1)$ の因数を含むときは, 桁数落ちが起こる. そのため $u=nh$ として

(1) $-1 \leq x < 0$ のとき

$$\begin{aligned} 1+x &= 1+\tanh(\frac{\pi}{2}\sinh(u)) \\ &= \exp(\frac{\pi}{2}\sinh(u))/\cosh(\frac{\pi}{2}\sinh(u)) = -t_1 \end{aligned}$$

(2) $0 \leq x \leq 1$ のとき

$$\begin{aligned} 1-x &= 1-\tanh(\frac{\pi}{2}\sinh(u)) \\ &= \exp(-\frac{\pi}{2}\sinh(u))/\cosh(\frac{\pi}{2}\sinh(u)) = t_2 \end{aligned}$$

と分けて,

$$\begin{aligned} \int_{-1}^{1} f(x)dx &= \int_{-1}^{0} f(x)dx + \int_{0}^{1} f(x)dx \\ &= \int_{-1}^{0} f(-1-t_1)dt_1 + \int_{0}^{1} f(1-t_2)dt_2 \end{aligned}$$

とする.

4.1.2.6 振動型関数の無限区間積分

重み関数を $\cos(\omega x)$ または $\sin(\omega x)$ とするとき, 重み関数を含む被積分関数を $f(x), f(x)=0$ になる周期を λ, 位相のずれを θ, 任意の整数を m とすると, 次式が成立する.

$$Q = \int_{0}^{\infty} f(x)dx, f(m\lambda+\theta) = 0$$

いま M をある程度大きな正の数として次の変換を考える.

$$x = M\phi(t), \phi(-\infty) = 0, \phi(+\infty) = 0$$

こうするともとの積分は次のようになる.

$$Q = \int_{-\infty}^{\infty} f(M\phi(t))M\phi'(t)dt$$

これを刻み幅 h の台形則で求めるとすれば次式が得られる.

$$Q = Mh\sum_{n=-\infty}^{\infty} f(M\phi(nh+\frac{\theta}{M}))\phi'(nh+\frac{\theta}{M})$$

ここで $\phi(t)$ として, 次式を満たす関数とする.

$$\lim_{t\to\infty}\phi(t) = t$$

さらに $Mh=\lambda$ になるよう h をとると, 大きな n に対し次式が成り立つ.

$$f(M\phi(nh+\frac{\theta}{M})) = f(Mnh+\theta) = f(n\lambda+\theta) = 0$$

このような $\phi(t), \phi'(t)$ として次のものを選ぶ

$$\begin{cases} \phi(t) &= \dfrac{t}{1-\exp(-K\sinh t)} \\ \phi'(t) &= \dfrac{1-(1+Kt\cosh t)\cdot\exp(-K\sinh t)}{(1-\exp(-K\sinh t))^2} \end{cases} \qquad (K=6 \text{ とする})$$

こうすれば $t \to +\infty$ において $\phi(t)$ は t に近づき, $f(M\phi(t)) = 0$ になり, $t \to -\infty$ とすれば $\phi'(t)$ は 0 に近づく. また, $t = 0$ の近くでは, このまま計算すると桁落ちするので桁落ちを防ぐよう次のようにする.

$$\begin{cases} \phi(t) &= 1/K \\ \phi'(t) &= 0.5 \end{cases} \qquad |\, t \,| < \sqrt{\text{誤差判定のための単位}} \text{のとき}$$

積分の手順については, 最初のきざみ幅 h を次式で求める.

$$h = 3.23/(-\log(\text{要求絶対精度}))$$

積分の総和の範囲は, リチャードソン補外の式から次式が成立するまで両側に 1 ずつ拡げる. この範囲での総和で積分値が得られる.

$$\max\{t_1/15, (19.t_1/t_2)/45\}Mh < \text{要求絶対精度}/16$$

ここで現在の総和範囲両端での $f(M\phi(nh + \frac{\theta}{M}))\phi'(nh + \frac{\theta}{M})$ の和を t_1, その 1 つ前のステップでの t_1 の値を t_2 とする. このようにして求めた積分値を S_1 とし, きざみ幅 h を次々に半分にして同様に積分値 $S_2, S_3 \cdots$ を求め,

$$|\, S_n - S_{n-1} \,| \leq \sqrt{\text{要求絶対精度}/10}$$

となったところで S_n を積分値とする.

4.1.2.7 多次元有限区間積分

1 次元方向に標本点数 N のガウス・ロンバーグ則公式を適用し数値積分する. この結果を G_N とする. 次の次元方向には, 前の次元での G_N をガウス・ロンバーグ則公式を適用し数値積分する. これを次々と次元に対しくり返す. カルテシアン積と積公式を用いると $G_N \times G_N \times \cdots$ で表現できる.
ここで N の値を $N_{n+1} = N_n + 2, N_1 = 2, (n = 1, 2, \cdots)$ と増加していけば, 真の解に近づく数列が得られる. この数列を次の θ アルゴリズムを改良した方法で加速し, 近似解を得る.
θ アルゴリズム

$$\theta_{-1}^{(n)} = 0, \theta_0^{(n)} = S^{(n)} \qquad (S^{(n)} \text{は得られる近似解数列})$$

とし,

$$\theta_{2k+1}^{(n)} = \theta_{2k-1}^{(n+1)} + \frac{1}{\Delta\theta_{2k}^{n}}$$

近似解

$$\begin{aligned}
&if \;\; not \;\; (|\, \Delta\theta_{2k}^{(n)} \,| > |\, \Delta\theta_{2k}^{(n+1)} \,| > |\, \Delta\theta_{2k}^{(n+2)} \,|) \; then \\
&\quad if \;\; (|\, \Delta\theta_{2k}^{n+1} \,| > |\, \Delta\theta_{2k}^{(n+2)} \,|) \; then \\
&\qquad \theta_{2k+2}^{(n)} = \theta_{2k}^{(n+2)} + 1/\Delta\theta_{2k+1}^{(n+1)} \qquad * \\
&\quad else \; if \; (sign(\Delta\theta_{2k}^{(n+2)}) = sign(\Delta\theta_{2k}^{(n+1)})) \; then \\
&\qquad \theta_{2k+2}^{(n)} = \theta_{2k}^{(n+2)} + 1.5 \cdot \Delta\theta_{2k}^{(n+1)} \qquad ** \\
&\quad else \\
&\qquad \theta_{2k+2}^{(n)} = (\theta_{2k}^{(n+3)} + \theta_{2k}^{(n+1)})/2 \qquad ** \\
&\quad end \\
&else \;\; if \;\; (|\, \Delta\theta_{2k+1}^{(n+1)} \,| \leq |\, \Delta\theta_{2k+1}^{(n)} \,|) \;\; then \\
&\quad \theta_{2k+2}^{(n)} = \theta_{2k}^{(n+2)} + 1/\Delta\theta_{2k+1}^{(n+1)} \qquad * \\
&else \\
&\quad \theta_{2k+2}^{(n)} = \theta_{2k}^{(n+1)} + \frac{\Delta\theta_{2k}^{(n+1)} \cdot \Delta\theta_{2k+1}^{(n+1)}}{\Delta^2\theta_{2k+1}^{(n)}} \\
&end
\end{aligned}$$

ただし, $sign(x)$ は x の符号を与える. また積分近似解の推定誤差 ε は, $*$ のついていない処理に対しては,

$$\varepsilon = \left\{ \begin{matrix} \text{倍精度} : 6 \\ \text{単精度} : 9 \end{matrix} \right\} \cdot \max(\mid \theta_{2k+2}^{(n)} - \theta_{2k}^{(n+2)} \mid, \mid \theta_{2k+2}^{(n)} - \theta_{2k}^{(n+3)} \mid)$$

とし, $*$ のついている処理には ε の 2 倍, $**$ のついている処理には ε の 10 倍を考える.

4.1.2.8 特殊関数を被積分関数に含む定積分および両無限積分

(1) チェビシェフ多項式と 0 次ベッセル関数の積の定積分 $\int_0^1 J_0(\alpha x)T_n(2x-1)xdx$

$f(x)$ のチェビシェフ級数展開

$$f(x) = \sum_{n=0}^{\infty} C_n T_n(2x-1)$$

の項別積分から, Dini 展開係数に現われるベッセル関数との定積分を

$$\int_0^1 J_0(\alpha x)f(x)xdx = \sum_{n=0}^{\infty} C_n \int_0^1 J_0(\alpha x)T_n(2x-1)xdx$$

と計算できるようにするために, 定積分 $\int_0^1 J_0(\alpha x)T_n(2x-1)xdx$ を利用することができる.
ここで, $\alpha \leq 15.0$ では直接にガウス・ルジャンドル則公式を利用した方が安定であり, $\alpha \leq 10.0$ では, 漸化式の演算が演算精度の不足から異常となる. また, n は 30 程度までとする (演算精度の不足).
1. $W_{0,1} = \int_0^1 J_0(\alpha x)dx$ を求める.
近似式

$$\int_0^1 J_0(\alpha x)dx = (2N+1)^{-1} \sum_{k=0}^{2N} \frac{\sin(\alpha \cos(2\pi k/(2N+1)))}{\alpha \cos(2\pi k/(2N+1))}$$

を用いる.
2. 漸化式

$$I_n = \int_0^1 J_0(\alpha x)T_n(2x-1)xdx$$

は, 0 次ベッセル関数の満たす微分方程式を用いて $J_0(\alpha x)$ をこの微分および 2 階微分で表し, 部分積分によって変形すると

$$I_n = J_1(\alpha)/\alpha + 2n^2 J_0(\alpha)/\alpha^2 - \alpha^{-2}(W_{n,3} + W_{n,2})$$

となる. ここで,

$$W_{n,1} = \int_0^1 J_0(\alpha x)T_n(2x-1)dx$$

$$W_{n,2} = \int_0^1 J_0(\alpha x)(T_n(2x-1))''xdx$$

$$W_{n,3} = \int_0^1 J_0(\alpha x)(T_n(2x-1))'dx$$

である. さらに,

$$W_{0,1} = \int_0^1 J_0(\alpha x)dx$$

$$W_{0,2} = W_{0,3} = W_{1,2} = 0$$

$$W_{1,3} = 2W_{0,1}, W_{1,1} = 2I_0 - W_{0,1}, W_{2,2} = 16I_0, W_{2,3} = 16I_0 - 8W_{0,1}$$

$$W_{n,1} + 2W_{n-1,1} + W_{n-2,1} = 4I_{n-1}(n \geq 2)$$

$$W_{n,3}/n - W_{n-2,3}/(n-2) = 4W_{n-1,1}(n \geq 3)$$

$$W_{n,2}/n - W_{n-2,2}/(n-2) = (n-1)/nW_{n,3} + 2W_{n-1,3} + (n-1)/(n-2)W_{n-2,3}(n \geq 3)$$

を用いる. これらは, チェビシェフ多項式 $T_n(x)$ の性質より得られる.

(2) 任意の関数 $f(x)$ と 0 次ベッセル関数の積の定積分 $\int_0^1 J_0(\alpha x)f(x)xdx$
区間 [0,1] を細分区間にわけて, 各々の区間 $[\alpha_1, \alpha_3]$ とその中点 α_2 について, $\alpha_j^2(j = 1, 2, 3)$ での値が $f(\alpha_j)$ である, 2 次以下の多項式 $P(x)$ を定めて, $f(x)$ を $P(x^2)$ で近似し, $\int_{\alpha_1}^{\alpha_3} J_0(\alpha x)P(x^2)xdx$ を解析的に計算して, 細分区間の全体にわたって, これを加算する.

(3) 無限積分 $\int_{-\infty}^{\infty} e^{-x^2}f(x)dx$
$\int_{-\infty}^{\infty} e^{-x^2}f(x)dx$ を

$$\sum_{1 \leq j \leq n, -3 \leq m \leq 3} w_j e^{-(z_j+2m)^2} f(z_j + 2m)$$

$(z_j, w_j(j = 1, 2, \cdots, n)$ はガウス積分点 (n 次ルジャンドル多項式の零点), 重みである) で近似する.

4.1.3 参考文献

(1) 二宮市三, “適応型ニュートン・コーツ積分法の改良”, 情報処理 Vol. 21 No5(1980)

(2) Davis & Rabinowitz, 森正武訳, “計算機による数値積分法”, 日本コンピュータ協会 (1981)

(3) Forsythe & Malcolm & 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)

(4) 森正武, “曲線と曲面”, 教育出版 (1974)

(5) 戸田英雄, 小野令美, “入門数値計算”, オーム社 (1983)

(6) 大浦拓哉, 森正武, “振動型半無限積分に対する変数変換型公式”, 京都大学数理科学考究録, 数値解析と科学計算, Vol. 717 (1990)

(7) 伊理正夫, 藤野和建, “数値計算の常識”, 共立出版 (1985)

(8) Fritsch, F. N. , Kahaner, D. K. , Lyness, J. N. , “Double Integration Using One-Dimensional Adaptive Quadrature Routines: A Software Interface Problem”, ACM Trans. Math. Softw. Vol. 7, pp. 46–75 (1979)

(9) 鶴見兼久, “$\varepsilon-$ アルゴリズムによる特異積分の数値計算法”, 情報処理学会全国大会講演論文集, Vol. 20, pp. 449-450(1975)

(10) 森正武, “最適の数値積分公式は何か 1, 2”, bit Vol. 3, No5, 6 (1971)

(11) 鳥居達生, 古川信次, 二宮市三, “ある種の特異積分に対する Clenshaw-Curtis 型積分法”, 情報処理学会全国大会講演論文集, Vol. 20, pp. 451-452 (1979)

(12) 井阪秀高, “適応型ニュートン・コーツ則の誤差評価と特異点処理”, 情報処理学会全国大会講演論文集, Vol. 32, pp. 1723-1724 (1981)

(13) 井阪秀高, “多重積分への加速法の適用”, 情報処理学会全国大会講演論文集, Vol. 33, pp. 1859-1860 (1981)

(14) Patterson; “The Optimal Addition of Points to Quadrature Formulae” Math. Comp. Vol.22, 1968

(15) Piessens, Branders; “A Note of the Optimal Addition of Abscissas to Quadrature Formulas of Gauss and Lobatto Type”, Math. Comp. Vol. 28, 1974

(16) Wynn; “On the Convergence and Stability of the Epsilon Algorithm”, J. SIAM Num. Anal. Vol. 3, No. 1, 1966

(17) Monegato; “A Note on Extended Gaussian Quadrature Rules”, Math. Comp. Vol. 30, 1976

(18) 伊理正夫, “数値計算”, 朝倉, 理工系基礎の数学 12

(19) 森 正武, “数値解析法”, 朝倉, 現代物理学講座 7

4.2 有限区間積分

4.2.1 ASL_dhemnl, ASL_rhemnl
任意の関数

(1) **機 能**

関数の有限区間積分を自動的に行う. 被積分関数に特異性があってもその性質を判定し, 自動的に処理を行う. 入出力引数は必要最小限とし, 使用しやすくなっている.

(2) **使用法**

倍精度関数:

ierr = ASL_dhemnl (f, a, b, er, & q, & ae);

単精度関数:

ierr = ASL_rhemnl (f, a, b, er, & q, & ae);

(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	被積分関数 $f(x)$ を定義する関数名
2	a	{D / R}	1	入 力	積分の下端
3	b	{D / R}	1	入 力	積分の上端
4	er	{D / R}	1	入 力	要求相対精度 (注意事項 (c) 参照) (4.1.1 参照)
5	q	{D∗ / R∗}	1	出 力	積分値
6	ae	{D∗ / R∗}	1	出 力	絶対誤差推定値 (4.1.1 参照)
7	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) a < b

(b) er ≥ 誤差判定のための単位 ×64 (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.	
1000	特異点が 5 個以上ある.	正常終了.
1200	制限条件 (a) を満足しなかった.	区間 (b, a) の積分値に -1 を乗じる or 積分値 $= 0.0$
1500	制限条件 (b) を満足しなかった.	既定値にセットして処理する.
2000	細分区間数が 500 回に達した.	徐々に最小分割幅を広げ, 近似解を得るようにする.
2400	ある細分区間の細分がこれ以上不可能となった.	要求された精度の解が得られないまま処理が終了する.
2500	解の精度が要求精度に達しない.	
3500	結果の信用性がない (誤差が結果より大きい).	処理を打ち切る.
4000	一つの細分区間で 2 回以上のオーバフローが発生した.	

(6) **注意事項**

(a) 積分区間内では, 関数値 f のオーバフローが生じないよう対策をとる必要がある (たとえば, 特異点での関数値を 0.0 とする).

(b) 被積分関数が内点特異型のときは, 特異点が全区間を 2^n 等分した点の上にくるようにして解くことが望ましい. また, 倍精度での要求精度は既定値より緩くしないと解の精度が悪くなったり, 特異点の数が実際より多く出力されることがある.
被積分関数が著しいピーク型のときは, 倍精度関数により要求精度を高くして解くことが望まれる. その他のときや分からないときは, 必要とする精度を要求精度として解けばよい.

(c) 変数 er は, 0.0 を入力すれば既定値がセットされる.
既定値 = 誤差判定のための単位 $\times 64$

(d) この関数は, 適応型ニュートン・コーツ 9 点則を基本とし, 特異点処理能力を強化したアルゴリズムを利用する.

(7) **使用例**

(a) 問 題
$\int_0^1 \sqrt{x} \log x dx$ を求める.

(b) 入力データ
被積分関数 $f(x)$ に対応する関数名: f
($x = 0.0$ のとき f=0.0 とする).
a=0.0, b=1.0, er=0.0

(c) 主プログラム

```
/*      C interface example for ASL_dhemnl */

#include <stdio.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
{
    if( *x == 0.0 )
        return 0.0;
    else
        return sqrt(*x) * log(*x);
}
#ifdef __cplusplus
}
#endif

int main()
{
    double a;
    double b;
    double epsrel;
    double result;
    double abserr;
    int ierr;
    FILE *fp;

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

    printf( "    *** ASL_dhemnl ***\n" );
    printf( "\n     ** Input **\n\n" );
    fscanf( fp, "%lf", &a );
    fscanf( fp, "%lf", &b );
    fscanf( fp, "%lf", &epsrel );

    printf( "\ta   = %8.3g\n", a );
    printf( "\tb   = %8.3g\n", b );
    printf( "\ter  = %8.3g\n", epsrel );

    fclose( fp );

    ierr = ASL_dhemnl(f, a, b, epsrel, &result, &abserr);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tIntegral Approximation\n" );
    printf( "\t  q   = %8.3g\n", result );
    printf( "\n\tEstimate of Absolute Error\n" );
    printf( "\t  ae  = %8.3g\n", abserr );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dhemnl ***

 ** Input **

a   =        0
b   =        1
er  =        0

 ** Output **

ierr =      0

Integral Approximation
  q   =   -0.444

Estimate of Absolute Error
  ae  =  1.6e-15
```

4.2.2 ASL_dhnsnl, ASL_rhnsnl
穏やかな関数

(1) **機 能**

特異性のない穏やかな関数の有限区間積分をする.

(2) **使用法**

倍精度関数:

ierr = ASL_dhnsnl (f, a, b, er, ea, & q, & ae, & nev);

単精度関数:

ierr = ASL_rhnsnl (f, a, b, er, ea, & q, & ae, & nev);

(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	被積分関数 $f(x)$ を定義する関数名
2	a	{D / R}	1	入 力	積分の下端
3	b	{D / R}	1	入 力	積分の上端
4	er	{D / R}	1	入 力	要求相対精度 (既定値 : 誤差判定のための単位 ×64) (4.1.1 参照)
5	ea	{D / R}	1	入 力	要求絶対精度 (既定値 : 絶対値最小値 $\times 2^{24}$) (4.1.1 参照)
6	q	{D∗ / R∗}	1	出 力	積分値
7	ae	{D∗ / R∗}	1	出 力	絶対誤差推定値 (4.1.1 参照)
8	nev	I*	1	出 力	被積分関数評価回数
9	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) a < b

(b) er ≥ 誤差判定のための単位 ×64(既定値にするため, 0.0 を入力する場合は除く)

(c) ea ≥ 絶対値最小値 $\times 2^{24}$(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.	
1200	制限条件 (a) を満足しなかった.	区間 (b, a) の積分値に -1 を乗じる or 積分値 $= 0.0$
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を続ける.
2500	解の精度が要求精度に達しない.	要求された精度の解が得られないままで処理が終了する.
3500	結果の信用性がない (誤差が結果より大きい).	処理を打ち切る.

(6) 注意事項

(a) 引数の内容の欄に既定値が記されている場合は, 0.0 を入力すれば既定値がセットされる.

(b) この関数は, 10 点ガウス則より始め, 21 点, 43 点, 87 点, 175 点と点数を増したクロンロッド則により積分値を更新していく非適応型アルゴリズムを利用する.

(7) 使用例

(a) 問 題

$\int_0^1 (x^2 - 2x + 1)dx$ を求める.

(b) 入力データ

被積分関数 $f(x)$ に対応する関数名: f

a=0.0, b=1.0, er=0.0, ea=0.0

(c) 主プログラム

```
/*      C interface example for ASL_dhnsnl */

#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) - 2.0*(*x) + 1.0) ;
}
#ifdef __cplusplus
}
#endif


int main()
{
    double a;
    double b;
    double ers;
    double eas;
    double q;
    double ae;
    int nev;
    int ierr;
    FILE *fp;

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

    printf( "    *** ASL_dhnsnl ***\n" );
    printf( "\n     ** Input **\n\n" );
```

```
    fscanf( fp, "%lf", &a );
    fscanf( fp, "%lf", &b );
    fscanf( fp, "%lf", &ers );
    fscanf( fp, "%lf", &eas );

    printf( "\ta   = %8.3g\n", a );
    printf( "\tb   = %8.3g\n", b );
    printf( "\ter  = %8.3g\n", ers );
    printf( "\tea  = %8.3g\n", eas );

    fclose( fp );

    ierr = ASL_dhnsnl(f, a, b, ers, eas, &q, &ae, &nev);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tIntegral Approximation\n" );
    printf( "\t  q   = %8.3g\n", q );
    printf( "\n\tEstimate of Absolute Error\n" );
    printf( "\t  ae  = %8.3g\n", ae );
    printf( "\n\tNumber of Function Evaluations\n" );
    printf( "\t  nev = %6d\n", nev );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dhnsnl ***

 ** Input **

a   =        0
b   =        1
er  =        0
ea  =        0

 ** Output **

ierr =      0

Integral Approximation
  q   =    0.333

Estimate of Absolute Error
  ae  = 1.48e-16

Number of Function Evaluations
  nev =     21
```

4.2.3 ASL_dhnofl, ASL_rhnofl
$f(x) \cdot (\sin \omega x$ **or** $\cos \omega x)$ 型の関数

(1) **機 能**

振動型関数のうち, $f(x) \cdot (\sin \omega x$ or $\cos \omega x)$ の形に因数分解できる関数を有限区間積分する.

(2) **使用法**

倍精度関数:

ierr = ASL_dhnofl (f, a, b, w, itype, er, ea, idv, & q, & ae, & nev, iwk, wk);

単精度関数:

ierr = ASL_rhnofl (f, a, b, w, itype, er, ea, idv, & q, & ae, & 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	f	{D / R}	–	入 力	被積分関数の因数 $f(x)$ を定義する関数名
2	a	{D / R}	1	入 力	積分の下端
3	b	{D / R}	1	入 力	積分の上端
4	w	{D / R}	1	入 力	重みの関数 $(\sin \omega x, \cos \omega x)$ での ω
5	itype	I	1	入 力	重みの関数の区別 $1 \cdots \int f(x) \cos(\omega x) dx$ $2 \cdots \int f(x) \sin(\omega x) dx$
6	er	{D / R}	1	入 力	要求相対精度 (既定値 : 誤差判定のための単位 $\times 64$) (4.1.1 参照)
7	ea	{D / R}	1	入 力	要求絶対精度 (既定値 : 絶対値最小値 $\times 2^{24}$) (4.1.1 参照)
8	idv	I	1	入 力	正常処理を行う細分区間数の最大値 (既定値 : 500)
9	q	{D∗ / R∗}	1	出 力	積分値
10	ae	{D∗ / R∗}	1	出 力	絶対誤差推定値 (4.1.1 参照)
11	nev	I*	1	出 力	被積分関数評価回数
12	iwk	I*	2×idv	ワーク	作業領域 (idv に 0 を入力した場合は 1000 の大きさで確保する)
13	wk	{D∗ / R∗}	4×idv	ワーク	作業領域 (idv に 0 を入力した場合は 2000 の大きさで確保する)
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) a < b

(b) er ≥ 誤差判定のための単位 ×64(既定値にするため, 0.0 を入力する場合は除く)

(c) ea ≥ 絶対値最小値 $\times 2^{24}$(既定値にするため, 0.0 を入力する場合は除く)

(d) idv>1(既定値にするため, 0 を入力する場合は除く)

(e) itype=1, 2

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1200	制限条件 (a) を満足しなかった.	区間 (b, a) の積分値に -1 を乗じる or 積分値=0.0
1500	制限条件 (b), (c) または (d) を満足しなかった.	既定値にセットして処理を続ける.
2000	細分区間数が idv 回に達した.	要求された精度の解が得られないままで処理が終了する.
2400	ある細分区間の細分がこれ以上不可能となった.	
2500	解の精度が要求精度に達しない.	
2700	$\varepsilon-$ アルゴリズムによる解の収束が得られない.	
3000	制限条件 (e) を満足しなかった.	処理を打ち切る.
3500	結果の信用性がない (誤差が結果より大きい).	

(6) **注意事項**

(a) 引数の内容の欄に既定値が記されている場合は, 0.0 を入力すれば既定値がセットされる.

(b) この関数は振動の激しい区間では修正 25 点クレンショーカーチス則を利用して積分し, その誤差を 13 点則と考え合わせて計算する. 振動が激しくない所では, 7-15 点ガウス–クロンロッド則により積分値とその誤差が計算される.

(7) **使用例**

(a) 問 題

$\displaystyle\int_0^{\frac{\pi}{2}} \sin x \cdot \frac{1}{\sqrt{1-0.25\sin^2 x}} dx$ を求める.

(b) 入力データ

被積分関数の因数 $f(x)$ に対応する関数名: f

a=0.0, b=$\frac{\pi}{2}$, w=1.0, itype=2, er=0.0, ea=0.0, idv=0

(c) 主プログラム

```
/*      C interface example for ASL_dhnofl */

#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 1.0/(sqrt(1.0-0.25*sin(*x)*sin(*x)));
}
#ifdef __cplusplus
}
#endif


int main()
```

```
{
    double a;
    double b;
    double w;
    int itype;
    double er;
    double ea;
    int idv;
    int idv0=500;
    double q;
    double ae;
    int nev;
    int *iwk;
    double *wk;
    int ierr;
    FILE *fp;

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

    printf( "    *** ASL_dhnofl ***\n" );
    printf( "\n     ** Input **\n\n" );
    idv=0;
    fscanf( fp, "%lf", &a );
    fscanf( fp, "%lf", &b );
    fscanf( fp, "%lf", &w );
    fscanf( fp, "%d", &itype);
    fscanf( fp, "%lf", &er );
    fscanf( fp, "%lf", &ea );

    iwk = ( int * )malloc((size_t)( sizeof(int) * (2*idv0) ));
    if( iwk == NULL )
    {
        printf( "no enough memory for array iwk\n" );
        return -1;
    }

    wk = ( double * )malloc((size_t)( sizeof(double) * (4*idv0) ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

    printf( "\ta     = %8.3g\n", a );
    printf( "\tb     = %8.3g\n", b );
    printf( "\tw     = %8.3g\n", w );
    printf( "\titype =   %6d\n", itype );
    printf( "\ter    = %8.3g\n", er );
    printf( "\tea    = %8.3g\n", ea );
    printf( "\tidv   =   %6d\n", idv );

    fclose( fp );

    ierr = ASL_dhnofl(f, a, b, w, itype, er, ea, idv, &q, &ae, &nev, iwk, wk);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tIntegral Approximation\n" );
    printf( "\t  q   = %8.3g\n", q );
    printf( "\n\tEstimate of Absolute Error\n" );
    printf( "\t  ae  = %8.3g\n", ae );
    printf( "\n\tNumber of Function Evaluations\n" );
    printf( "\t  nev = %6d\n", nev );

    free( iwk );
    free(  wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dhnofl ***

 ** Input **

a     =        0
b     =     1.57
w     =        1
itype =        2
er    =        0
ea    =        0
idv   =        0

 ** Output **

ierr =      0

Integral Approximation
  q   =      1.1
```

```
Estimate of Absolute Error
  ae  = 1.05e-14

Number of Function Evaluations
  nev =     75
```

4.2.4 ASL_dhnefl, ASL_rhnefl

$f(x)\cdot((x-a)^{\alpha}(b-x)^{\beta}\{\log(x-a)\}^{\gamma}\{\log(b-x)\}^{\delta})(a<x<b;\gamma,\delta=0,1)$ 型の関数

(1) **機 能**

端点特異型関数のうち, $f(x)\cdot((x-a)^{\alpha}(b-x)^{\beta}\{\log(x-a)\}^{\gamma}\{\log(b-x)\}^{\delta})(a<x<b;\gamma,\delta=0,1)$ の形に因数分解できる関数を有限区間積分する.

(2) **使用法**

倍精度関数:

ierr = ASL_dhnefl (f, a, b, alfa, beta, itype, er, ea, idv, & q, & ae, & nev, iwk, wk);

単精度関数:

ierr = ASL_rhnefl (f, a, b, alfa, beta, itype, er, ea, idv, & q, & ae, & 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　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	被積分関数の因数 $f(x)$ を定義する関数名
2	a	{D / R}	1	入 力	積分の下端
3	b	{D / R}	1	入 力	積分の上端
4	alfa	{D / R}	1	入 力	重みの関数 $(x-a)^{\alpha}$ での $\alpha(\alpha>-1)$
5	beta	{D / R}	1	入 力	重みの関数 $(b-x)^{\beta}$ での $\beta(\beta>-1)$
6	itype	I	1	入 力	重みの関数の対数因子の区別 $=1:(x-a)^{\alpha}\cdot(b-x)^{\beta}$ $=2:(x-a)^{\alpha}\cdot(b-x)^{\beta}\cdot\log(x-a)$ $=3:(x-a)^{\alpha}\cdot(b-x)^{\beta}\cdot\log(b-x)$ $=4:(x-a)^{\alpha}\cdot(b-x)^{\beta}\cdot\log(x-a)\cdot\log(b-x)$
7	er	{D / R}	1	入 力	要求相対精度 (既定値 : 誤差判定のための単位 $\times 64$) (4.1.1 参照)
8	ea	{D / R}	1	入 力	要求絶対精度 (既定値 : 絶対値最小値 $\times 2^{24}$) (4.1.1 参照)
9	idv	I	1	入 力	正常処理を行う細分区間数の最大値 (既定値 : 500)
10	q	{D∗ / R∗}	1	出 力	積分値

項番	引数と戻り値	型	大きさ	入出力	内　　容
11	ae	$\left\{\begin{matrix}\mathrm{D}*\\ \mathrm{R}*\end{matrix}\right\}$	1	出 力	絶対誤差推定値 (4.1.1 参照)
12	nev	I*	1	出 力	被積分関数評価回数
13	iwk	I*	idv	ワーク	作業領域 (idv に 0 を入力した場合は 500 の大きさで確保する)
14	wk	$\left\{\begin{matrix}\mathrm{D}*\\ \mathrm{R}*\end{matrix}\right\}$	4 × idv	ワーク	作業領域 (idv に 0 を入力した場合は 2000 の大きさで確保する)
15	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) a<b

(b) er≥ 誤差判定のための単位 ×64 (既定値にするため, 0.0 を入力する場合は除く)

(c) ea≥ 絶対値最小値 $\times 2^{24}$ (既定値にするため, 0.0 を入力する場合は除く)

(d) idv>1(既定値にするため, 0 を入力する場合は除く)

(e) itype=1, 2, 3, 4

(f) alfa > -1.0, beta > -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.	
1200	制限条件 (a) を満足しなかった.	区間 (b, a) の積分値に -1 を乗じる or 積分値 $= 0.0$
1500	制限条件 (b), (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.
2000	細分区間数が idv に達した.	要求された精度の解が得られないままで, 処理が終了する.
2400	ある細分区間の細分がこれ以上不可能となった.	
2500	解の精度が要求精度に達しない.	
3000	制限条件 (e) または (f) を満足しなかった.	処理を打ち切る.
3500	結果の信用性がない (誤差が結果より大きい).	

(6) **注意事項**

(a) 積分区間内では, 関数値 f のオーバフローが生じないよう対策をとる必要がある. (たとえば, 特異点での関数値を 0.0 とする).

(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) この関数は, 端点を含む細分区間において 13 点と 25 点の修正クレンショーカーチス則を利用し, 他の細分区間では 7-15 点ガウス–クロンロッド則により積分値とその誤差を計算する.

(7) 使用例

(a) 問 題

$\int_0^1 \frac{\log(\frac{1}{x})}{\sqrt{x}}dx$ を求める.

(b) 入力データ

被積分関数の因数 $f(x)$ に対応する関数名: f

($x=0.0$ のとき f=0.0 とする).

a=0.0, b=1.0, alfa=−0.5, beta=0.0, itype=1, er=0.0, ea=0.0, idv=0

(c) 主プログラム

```
/*      C interface example for ASL_dhnefl */

#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
{
    if( *x == 0.0 )
        return 0.0;
    else
        return log(1.0/(*x));
}
#ifdef __cplusplus
}
#endif

int main()
{
    double a;
    double b;
    double alfa;
    double beta;
    int itype;
    double er;
    double ea;
    int idv;
    int idv0=500;
    double q;
    double ae;
    int nev;
    int *iwk;
    double *wk;
    int ierr;
    FILE *fp;

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

    printf( "    *** ASL_dhnefl ***\n" );
    printf( "\n     ** Input **\n\n" );
    idv=0;
    fscanf( fp, "%lf", &a );
    fscanf( fp, "%lf", &b );
    fscanf( fp, "%lf", &alfa );
    fscanf( fp, "%lf", &beta );
    fscanf( fp, "%d", &itype);
    fscanf( fp, "%lf", &er );
    fscanf( fp, "%lf", &ea );

    iwk = ( int * )malloc((size_t)( sizeof(int) * idv0 ));
    if( iwk == NULL )
    {
        printf( "no enough memory for array iwk\n" );
        return -1;
    }

    wk = ( double * )malloc((size_t)( sizeof(double) * (4*idv0) ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }
```

```
    printf( "\ta     = %8.3g\n", a );
    printf( "\tb     = %8.3g\n", b );
    printf( "\talfa  = %8.3g\n", alfa );
    printf( "\tbeta  = %8.3g\n", beta );
    printf( "\titype =   %6d\n", itype );
    printf( "\ter    = %8.3g\n", er );
    printf( "\tea    = %8.3g\n", ea );
    printf( "\tidv   =   %6d\n", idv );

    fclose( fp );

    ierr = ASL_dhnefl(f, a, b, alfa, beta, itype, er, ea, idv, &q, &ae, &nev, iwk, wk);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tIntegral Approximation\n" );
    printf( "\t  q   = %8.3g\n", q );
    printf( "\n\tEstimate of Absolute Error\n" );
    printf( "\t  ae  = %8.3g\n", ae );
    printf( "\n\tNumber of Function Evaluations\n" );
    printf( "\t  nev = %6d\n", nev );

    free( iwk );
    free(  wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dhnefl ***

 ** Input **

a     =        0
b     =        1
alfa  =     -0.5
beta  =        0
itype =        1
er    =        0
ea    =        0
idv   =        0

 ** Output **

ierr =      0

Integral Approximation
  q   =        4

Estimate of Absolute Error
  ae  =  1.1e-13

Number of Function Evaluations
  nev =   4490
```

4.2.5 ASL_dhnifl, ASL_rhnifl
$f(x)\cdot(1/(x-c))$ 型の関数

(1) **機 能**

内点特異型関数のうち, $f(x)\cdot(1/(x-c))$ の形に因数分解できる関数を有限区間積分する.

(2) **使用法**

倍精度関数:

ierr = ASL_dhnifl (f, a, b, c, er, ea, idv, & q, & ae, & nev, iwk, wk);

単精度関数:

ierr = ASL_rhnifl (f, a, b, c, er, ea, idv, & q, & ae, & 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　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	被積分関数の因数 $f(x)$ を定義する関数名
2	a	{D / R}	1	入 力	積分の下端
3	b	{D / R}	1	入 力	積分の上端
4	c	{D / R}	1	入 力	重みの関数 $1/(x-c)$ での c
5	er	{D / R}	1	入 力	要求相対精度 (既定値 : 誤差判定のための単位 $\times 64$) (4.1.1 参照)
6	ea	{D / R}	1	入 力	要求絶対精度 (既定値 : 絶対値最小値 $\times 2^{24}$) (4.1.1 参照)
7	idv	I	1	入 力	正常処理を行う細分区間数の最大値 (既定値 : 500)
8	q	{D∗ / R∗}	1	出 力	積分値
9	ae	{D∗ / R∗}	1	出 力	絶対誤差推定値 (4.1.1 参照)
10	nev	I*	1	出 力	被積分関数評価回数
11	iwk	I*	idv	ワーク	作業領域 (idv に 0 を入力した場合は 500 の大きさで確保する)
12	wk	{D∗ / R∗}	4 × idv	ワーク	作業領域 (idv に 0 を入力した場合は 2000 の大きさで確保する)
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) a<b

(b) er $\geq$ 誤差判定のための単位 $\times 64$ (既定値にするため, 0.0 を入力する場合は除く)

(c) ea $\geq$ 絶対値最小値 $\times 2^{24}$ (既定値にするため, 0.0 を入力する場合は除く)

(d) idv>1(既定値にするため, 0 を入力する場合は除く)

(e) c≠a, b

(5) **エラーインディケータ (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.	区間 (b, a) の積分値に -1 を乗じる or 積分値=0.0
1500	制限条件 (b), (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.
2000	細分区間数が idv に達した.	要求された精度の解が得られないままで, 処理が終了する.
2400	ある細分区間の細分がこれ以上不可能となった.	
2500	解の精度が要求精度に達しない.	
3000	制限条件 (e) を満足しなかった.	処理を打ち切る.
3500	結果の信用性がない (誤差が結果より大きい).	

(6) **注意事項**

(a) 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.

(b) この関数は, c 点を含む細分区間において, 13 点と 25 点の修正クレンショーカーチス則を利用し, 他の細分区間では 7-15 点ガウス–クロンロッド則により積分値とその誤差を計算する.

(7) **使用例**

(a) 問 題
$\displaystyle\int_{-1}^{5}\frac{1}{x(5x^3+6)}dx$ を求める.

(b) 入力データ
被積分関数の因数 $f(x)$ に対応する関数名: f
a=$-$1.0, b=5.0, c=0.0, er=1.0e$-$8, ea=0.0, idv=0

(c) 主プログラム

```
/*      C interface example for ASL_dhnifl */

#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 1.0/(5.0*(*x)*(*x)*(*x)+6.0);
}
#ifdef __cplusplus
}
#endif

int main()
{
    double a;
    double b;
    double c;
    double er;
    double ea;
    int idv;
    int idv0=500;
    double q;
    double ae;
    int nev;
    int *iwk;
    double *wk;
    int ierr;
    FILE *fp;

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

    printf( "    *** ASL_dhnifl ***\n" );
    printf( "\n     ** Input **\n\n" );
    idv=0;
    fscanf( fp, "%lf", &a );
    fscanf( fp, "%lf", &b );
    fscanf( fp, "%lf", &c );
    fscanf( fp, "%lf", &er );
    fscanf( fp, "%lf", &ea );

    iwk = ( int * )malloc((size_t)( sizeof(int) * idv0 ));
    if( iwk == NULL )
    {
        printf( "no enough memory for array iwk\n" );
        return -1;
    }

    wk = ( double * )malloc((size_t)( sizeof(double) * (4*idv0) ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

    printf( "\ta     = %8.3g\n", a );
    printf( "\tb     = %8.3g\n", b );
    printf( "\tc     = %8.3g\n", c );
    printf( "\ter    = %8.3g\n", er );
    printf( "\tea    = %8.3g\n", ea );
    printf( "\tidv   =   %6d\n", idv );

    fclose( fp );

    ierr = ASL_dhnifl(f, a, b, c, er, ea, idv, &q, &ae, &nev, iwk, wk);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tIntegral Approximation\n" );
    printf( "\t  q   = %8.3g\n", q );
    printf( "\n\tEstimate of Absolute Error\n" );
    printf( "\t  ae  = %8.3g\n", ae );
    printf( "\n\tNumber of Function Evaluations\n" );
    printf( "\t  nev = %6d\n", nev );

    free( iwk );
    free(  wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dhnifl ***

 ** Input **

a     =       -1
b     =        5
c     =        0
er    =    1e-08
ea    =        0
idv   =        0
```

```
 ** Output **

ierr =      0

Integral Approximation
  q    =  -0.0899

Estimate of Absolute Error
  ae   = 3.51e-11

Number of Function Evaluations
  nev =    355
```

4.2.6 ASL_dhnpnl, ASL_rhnpnl
一般の振動型, ピーク型関数

(1) **機 能**

弱い特異性がある関数を有限区間積分する. 特異性の種類に対応して isw の値を決める必要がある.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dhnpnl  (f, a, b, er, ea, idv, & q, & ae, & nev, isw, iwk, wk);
```

単精度関数:

```
ierr = ASL_rhnpnl  (f, a, b, er, ea, idv, & q, & ae, & nev, 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	f	{D, R}	−	入 力	被積分関数 $f(x)$ を定義する関数名
2	a	{D, R}	1	入 力	積分の下端
3	b	{D, R}	1	入 力	積分の上端
4	er	{D, R}	1	入 力	要求相対精度 (既定値 : 誤差判定のための単位 ×64) (4.1.1 参照)
5	ea	{D, R}	1	入 力	要求絶対精度 (既定値 : 絶対値最小値 $\times 2^{24}$) (4.1.1 参照)
6	idv	I	1	入 力	正常処理を行う細分区間数の最大値 (既定値 : 500)
7	q	{D∗, R∗}	1	出 力	積分値
8	ae	{D∗, R∗}	1	出 力	絶対誤差推定値 (4.1.1 参照)
9	nev	I*	1	出 力	被積分関数評価回数
10	isw	I	1	入 力	1∼6 の整数で isw=1 のときはピーク型, isw=6 のときは振動型に適する.
11	iwk	I*	idv	ワーク	作業領域 (idv に 0 を入力した場合は 500 の大きさで確保する)
12	wk	{D∗, R∗}	4 × idv	ワーク	作業領域 (idv に 0 を入力した場合は 2000 の大きさで確保する)
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) a<b

(b) er≥ 誤差判定のための単位 ×64 (既定値にするため, 0.0 を入力する場合は除く)

(c) ea≥ 絶対値最小値 $\times 2^{24}$ (既定値にするため, 0.0 を入力する場合は除く)

(d) idv > 1(既定値にするため, 0 を入力する場合は除く)

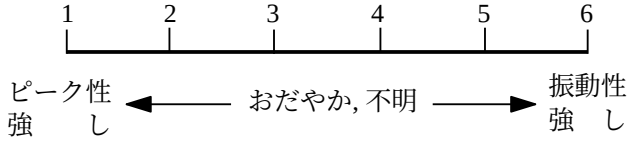
(e) $1 \leq \mathrm{isw} \leq 6$

(5) **エラーインディケータ (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.	区間 (b, a) の積分値に −1 を乗じる or 積分値 = 0.0
1500	制限条件 (b), (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.
2000	細分区間数が idv に達した.	要求された精度の解が得られないままで, 処理が終了する.
2400	ある細分区間の細分がこれ以上不可能となった.	
2500	解の精度が要求精度に達しない.	
3000	制限条件 (e) を満足しなかった.	処理を打ち切る.
3500	結果の信用性がない (誤差が結果より大きい).	

(6) **注意事項**

(a) 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. 引数 isw は, 下図のような規準で利用することが望まれる.



(b) この関数は適応型ガウス–クロンロッド則を利用する.

(7) **使用例**

(a) 問 題

$\int_0^{0.9} \sin(10\pi x)dx$ を求める.

(b) 入力データ

被積分関数 $f(x)$ に対応する関数名: f

a=0.0, b=0.9, er=1.0e−10, ea=0.0, idv=0, isw=6

(c) 主プログラム

```
/*      C interface example for ASL_dhnpnl */

#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 sin(10.0*M_PI*(*x));
}
#ifdef __cplusplus
}
#endif

int main()
{
    double a;
    double b;
    double ers;
    double eas;
    int idvs;
    int idv0=500;
    double q;
    double ae;
    int nev;
    int isw;
    int *iwk;
    double *wk;
    int ierr;
    FILE *fp;

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

    printf( "    *** ASL_dhnpnl ***\n" );
    printf( "\n     ** Input **\n\n" );
    fscanf( fp, "%lf", &a );
    fscanf( fp, "%lf", &b );
    fscanf( fp, "%lf", &ers );
    fscanf( fp, "%lf", &eas );
    fscanf( fp, "%d", &idvs );
    fscanf( fp, "%d", &isw );

    iwk = ( int * )malloc((size_t)( sizeof(int) * idv0 ));
    if( iwk == NULL )
    {
        printf( "no enough memory for array iwk\n" );
        return -1;
    }

    wk = ( double * )malloc((size_t)( sizeof(double) * (4*idv0) ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

    printf( "\ta     = %8.3g\n", a );
    printf( "\tb     = %8.3g\n", b );
    printf( "\ters   = %8.3g\n", ers );
    printf( "\teas   = %8.3g\n", eas );
    printf( "\tidv   =   %6d\n", idvs );
    printf( "\tisw   =   %6d\n", isw );

    fclose( fp );

    ierr = ASL_dhnpnl(f, a, b, ers, eas, idvs, &q, &ae, &nev, isw, iwk, wk);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tIntegral Approximation\n" );
    printf( "\t  q   = %8.3g\n", q );
    printf( "\n\tEstimate of Absolute Error\n" );
    printf( "\t  ae  = %8.3g\n", ae );
    printf( "\n\tNumber of Function Evaluations\n" );
    printf( "\t  nev = %6d\n", nev );

    free( iwk );
    free(  wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dhnpnl ***

 ** Input **
```

```
a     =        0
b     =      0.9
ers   =    1e-10
eas   =        0
idv   =        0
isw   =        6

 ** Output **

ierr =      0

Integral Approximation
  q   =   0.0637

Estimate of Absolute Error
  ae  = 2.56e-16

Number of Function Evaluations
  nev =     61
```

4.2.7 ASL_dhnenl, ASL_rhnenl
一般の端点特異型関数

(1) **機 能**

端点に特異性のある一般の関数を有限区間積分する.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dhnenl  (f, a, b, er, ea, itmx, & q, & ae, & nev, isw);
```

単精度関数:

```
ierr = ASL_rhnenl  (f, a, b, er, ea, itmx, & q, & ae, & nev, 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	f	{D / R}	–	入 力	被積分関数 $f(x)$ を定義する関数名
2	a	{D / R}	1	入 力	積分の下端
3	b	{D / R}	1	入 力	積分の上端
4	er	{D / R}	1	入 力	要求相対精度 (既定値 : 誤差判定のための単位 $\times 64$) (4.1.1 参照)
5	ea	{D / R}	1	入 力	要求絶対精度 (既定値 : 絶対値最小値 $\times 2^{24}$) (4.1.1 参照)
6	itmx	I	1	入 力	特異点間の最大反復回数 (DE 変換後の最小分割幅は, $0.25 \times 2^{-\mathrm{itmx}}$, 既定値 8)
7	q	{D∗ / R∗}	1	出 力	積分値
8	ae	{D∗ / R∗}	1	出 力	絶対誤差推定値 (4.1.1 参照)
9	nev	I*	1	出 力	被積分関数評価回数
10	isw	I	1	入 力	isw ≤ 0 : $f(x)$ をそのまま積分する isw ≥ 1 : $f(x)$ に桁落ち防止変換をしている
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) a<b

(b) er≥ 誤差判定のための単位 ×64 (既定値にするため, 0.0 を入力する場合は除く)

(c) ea≥ 絶対値最小値 $\times 2^{24}$ (既定値にするため, 0.0 を入力する場合は除く)

(d) itmx>1(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.	区間 (b, a) の積分値に −1 を乗じる or 積分値=0.0
1500	制限条件 (b), (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.
2000	反復回数が itmx 回に達した.	要求された精度の解が得られないままで, 処理が終了する.
2500	解の精度が要求精度に達しない.	
3100	DE 変換後, +− の両側で関数値が十分小さくならない.	処理を打ち切る.
3500	結果の信用性がない (誤差が結果より大きい).	

(6) **注意事項**

(a) 積分区間内では, 関数値のオーバフローが生じないよう対策をとる必要がある (たとえば, 特異点での関数値を 0.0 とする).

(b) 被積分関数 $f(x)$ に $(x-a)^{\alpha}(b-x)^{\beta}(-1<\alpha,\beta<0)$ の因子が含まれているときは, 桁落ち防止のため次のような変換を行う必要がある (詳細は参考文献 (5) 参照).

$a<x<\frac{a+b}{2}$ のところでは, $x=a-y(-\frac{b-a}{2}<y<0)$
$b>x\geq\frac{a+b}{2}$ のところでは, $x=b-y(0<y\leq\frac{b-a}{2})$

とし, y の関数として書きなおす.
そして, isw=1 としてもとの積分区間 (a, b) を用いる.
例えば,

$$\int_0^1 g(x)/\sqrt{x\cdot(1-x)}dx$$

であれば,

$$\int_0^1 g(x)/\sqrt{x\cdot(1-x)}dx=\int_{-\frac{1}{2}}^0 g(-y)/\sqrt{(-y)(1+y)}dy+\int_0^{\frac{1}{2}} g(1-y)/\sqrt{(1-y)y}dy$$

であるから次のようにする. ただし, y は x でおきかえてある.
なおこの場合 isw=1 として積分範囲 (0.0, 1.0) を用いている.

```
/* C interface example for ASL_dhnenl */

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

・関数 f

```
double FORTRAN f(double *x)··· 主プログラムにおける第 1 引数 f と同じ名前とする.
{
   if(*x>=0.0)                                                    ┐
      return  g(1.0-(*x))/sqrt((*x)*(1.0-(*x)));                  ├ 桁落ち防止変換
   else                                                           ┘
      return  g(-(*x)/sqrt(-(*x)*(1.0+(*x)));
}
```

・関数 g··· 利用者が任意に与える.

```
double FORTRAN g(double  *x)
{
    ≀
}
```

・主プログラム

```
int  main()
{
    ≀
    ierr  =  ASL_dhnenl(f, a, b, epsr, epsa, ···);
    ≀
}
```

(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) この関数は, 2 重指数関数型公式 (DE 変換公式) を利用する.

(7) 使用例

(a) 問 題

$\displaystyle\int_{-1}^{1} \frac{1}{\sqrt{1-x^2}}dx$ を求める.

(b) 入力データ

被積分関数 $f(x)$ に対応する関数名: f

$$\left(\mathrm{f}=\begin{cases}1/\sqrt{(2-\mathrm{x})\mathrm{x}} & (0<\mathrm{x}\leq 1)\\ 1/\sqrt{-\mathrm{x}(2+\mathrm{x})} & (-1<\mathrm{x}<0)\end{cases}\right)$$

a=−1.0, b=1.0, er=0.0, ea=0.0, itmx=0, isw=1

(c) 主プログラム

```
/*      C interface example for ASL_dhnenl */

#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
{
    if( *x >= 0.0 )
```

```
        return 1.0/sqrt((2.0-(*x))*(*x));
    else
        return 1.0/sqrt(-(*x)*(2.0+(*x)));
}
#ifdef __cplusplus
}
#endif

int main()
{
    double a;
    double b;
    double epsr;
    double epsa;
    int lim;
    double result;
    double abserr;
    int neval;
    int isw;
    int ierr;
    FILE *fp;

    fp = fopen( "dhnenl.dat", "r" );
    if( fp == NULL )
    {
        printf( "    *** ASL_dhnenl ***\n" );
        return -1;
    }

    printf( "    *** ASL_dhnenl ***\n" );
    printf( "\n     ** Input **\n\n" );
    fscanf( fp, "%lf", &a );
    fscanf( fp, "%lf", &b );
    fscanf( fp, "%lf", &epsr );
    fscanf( fp, "%lf", &epsa );
    fscanf( fp, "%d", &lim );
    fscanf( fp, "%d", &isw );

    printf( "\ta     = %8.3g\n", a );
    printf( "\tb     = %8.3g\n", b );
    printf( "\ter    = %8.3g\n", epsr );
    printf( "\tea    = %8.3g\n", epsa );
    printf( "\titmx  =   %6d\n", lim );
    printf( "\tisw   =   %6d\n", isw );

    fclose( fp );

    ierr = ASL_dhnenl(f, a, b, epsr, epsa, lim, &result, &abserr, &neval, isw);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tIntegral Approximation\n" );
    printf( "\t  q   = %8.3g\n", result );
    printf( "\n\tEstimate of Absolute Error\n" );
    printf( "\t  ae  = %8.3g\n", abserr );
    printf( "\n\tNumber of Function Evaluations\n" );
    printf( "\t  nev = %6d\n", neval );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dhnenl ***

 ** Input **

a     =       -1
b     =        1
er    =        0
ea    =        0
itmx  =        0
isw   =        1

 ** Output **

ierr =      0

Integral Approximation
  q   =     3.14

Estimate of Absolute Error
  ae  = 5.58e-15

Number of Function Evaluations
  nev =     65
```

4.2.8 ASL_dhninl, ASL_rhninl
一般の内点特異型関数

(1) **機 能**

積分区間内点に特異性のある一般の関数を有限区間積分する.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dhninl  (f, a, b, sp, nsp, er, ea, itmx, & q, & ae, & nev);
```

単精度関数:

```
ierr = ASL_rhninl  (f, a, b, sp, nsp, er, ea, itmx, & q, & ae, & nev);
```

(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	被積分関数 $f(x)$ を定義する関数名
2	a	{D / R}	1	入 力	積分の下端
3	b	{D / R}	1	入 力	積分の上端
4	sp	{D∗ / R∗}	nsp	入 力	特異点の X 座標値
				出 力	昇順にソートされた座標値
5	nsp	I	1	入 力	特異点の数
6	er	{D / R}	1	入 力	要求相対精度 (既定値 : 誤差判定のための単位 $\times 64$) (4.1.1 参照)
7	ea	{D / R}	1	入 力	要求絶対精度 (既定値 : 絶対値最小値 $\times 2^{24}$) (4.1.1 参照)
8	itmx	I	1	入 力	特異点間の最大反復回数 (DE 変換後の最小分割幅は, $0.25 \times 2^{-\mathrm{itmx}}$, 既定値 8)
9	q	{D∗ / R∗}	1	出 力	積分値
10	ae	{D∗ / R∗}	1	出 力	絶対誤差推定値 (4.1.1 参照)
11	nev	I*	1	出 力	被積分関数評価回数
12	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) a<b

(b) er≥ 誤差判定のための単位 ×64 (既定値にするため, 0.0 を入力する場合は除く)

(c) ea≥ 絶対値最小値 $\times 2^{24}$ (既定値にするため, 0.0 を入力する場合は除く)

(d) itmx>1(既定値にするため, 0 を入力する場合は除く)

(e) nsp>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.	
1200	制限条件 (a) を満足しなかった.	区間 (b, a) の積分値に −1 を乗じる or 積分値=0.0
1500	制限条件 (b), (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.
2000	一区間の反復回数が itmx 回に達した.	要求された精度の解が得られないままで, 処理が終了する.
2300	ある区間での積分精度が悪い.	
2500	解の精度が要求精度に達しない.	
3000	制限条件 (e) を満足しなかった.	処理を打ち切る.
3100	DE 変換後, +− の両側で関数値が十分小さくならなかった.	
3500	結果の信用性がない (誤差が結果より大きい).	

(6) **注意事項**

(a) 積分区間内では, 関数値 f のオーバフローが生じないよう対策をとる必要がある (たとえば, 特異点での関数値を 0.0 とする).

(b) 特異点での特異性が著しい場合は, 要求精度を満足せず, 単精度では 2 桁, 倍精度では 4 桁程度しか求まらないことがある. したがって, より高精度を要求する場合には, 特異点で分割し, 桁落ち防止変換し, 4.2.7 $\left\{\begin{matrix}\text{ASL_dhnenl}\\ \text{ASL_rhnenl}\end{matrix}\right\}$を利用して積分する必要がある.

(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) この関数は, 各特異点ごとに 2 重指数関数型公式 (DE 変換公式) を用いて積分値を求め, これらを加算することにより全体の積分値を計算する.

(7) 使用例

(a) 問 題

$\int_{-1}^{1} \frac{1}{\sqrt[3]{x^2}} dx$ を求める.

(b) 入力データ

被積分関数 $f(x)$ に対応する関数名: f

($x = 0.0$ のとき f=0.0 とする).

a=−1.0, b=1.0, sp [0] =0.0, nsp=1, er=1.0e−4, ea=0.0, itmx=0

(c) 主プログラム

```
/*      C interface example for ASL_dhninl */

#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
{
    if( *x == 0.0 )
        return 0.0;
    else if( *x > 0.0)
        return (pow((*x),(-2.0/3.0)));
    else
        return (pow(-(*x),(-2.0/3.0)));
}
#ifdef __cplusplus
}
#endif

int main()
{
    double a;
    double b;
    double *point;
    int npts;
    double epsrel;
    double epsabs;
    int limit;
    double result;
    double abserr;
    int neval;
    int ierr;
    int i;
    FILE *fp;

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

    printf( "    *** ASL_dhninl ***\n" );
    printf( "\n     ** Input **\n\n" );
    npts=1;

    point = ( double * )malloc((size_t)( sizeof(double) * npts ));
    if( point == NULL )
    {
        printf( "no enough memory for array point\n" );
        return -1;
    }
    fscanf( fp, "%lf", &a );
    fscanf( fp, "%lf", &b );
    for( i=0 ; i<npts ; i++ )
    {
           fscanf( fp, "%lf", &point[i] );
    }
    fscanf( fp, "%lf", &epsrel );
    fscanf( fp, "%lf", &epsabs );
    fscanf( fp, "%d", &limit );

    printf( "\ta         = %8.3g\n", a );
    printf( "\tb         = %8.3g\n", b );
    for( i=0 ; i<npts ; i++ )
    {
```

```
        printf( "\tsp[%6d]= %8.3g\n", i,point[i] );
    }
    printf( "\ter        = %8.3g\n", epsrel );
    printf( "\tea        = %8.3g\n", epsabs );
    printf( "\titmx      =   %6d\n", limit );

    fclose( fp );

    ierr = ASL_dhninl(f, a, b, point, npts, epsrel, epsabs, limit, &result, &abserr, &neval);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tSorted X-Coordinate Value of The Singular Point\n" );
    for( i=0 ; i<npts ; i++ )
    {
        printf( "\t  sp[%6d]= %8.3g\n", i,point[i] );
    }
    printf( "\n\tIntegral Approximation\n" );
    printf( "\t  q   = %8.3g\n", result );
    printf( "\n\tEstimate of Absolute Error\n" );
    printf( "\t  ae  = %8.3g\n", abserr );
    printf( "\n\tNumber of Function Evaluations\n" );
    printf( "\t  nev = %6d\n", neval );

    free( point );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dhninl ***

 ** Input **

a         =       -1
b         =        1
sp[     0]=        0
er        =   0.0001
ea        =        0
itmx      =        0

 ** Output **

ierr =      0

Sorted X-Coordinate Value of The Singular Point
  sp[     0]=        0

Integral Approximation
  q   =        6

Estimate of Absolute Error
  ae  = 3.16e-06

Number of Function Evaluations
  nev =     90
```

4.2.9 ASL_dhnanl, ASL_rhnanl
特異型であるがその情報が不明な関数

(1) **機 能**

端点または内点に特異性があるが, その情報が不明な場合の有限区間積分をする. やや計算時間が多くかかる.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dhnanl  (f, a, b, er, ea, idv, & q, & ae, & nev, iwk, wk);
```

単精度関数:

```
ierr = ASL_rhnanl  (f, a, b, er, ea, idv, & q, & ae, & 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　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	被積分関数 $f(x)$ を定義する関数名
2	a	{D / R}	1	入 力	積分の下端
3	b	{D / R}	1	入 力	積分の上端
4	er	{D / R}	1	入 力	要求相対精度 (既定値 : 誤差判定のための単位 ×64) (4.1.1 参照)
5	ea	{D / R}	1	入 力	要求絶対精度 (既定値 : 絶対値最小値 $\times 2^{24}$) (4.1.1 参照)
6	idv	I	1	入 力	正常処理を行う細分区間数の最大値 (既定値 : 500)
7	q	{D∗ / R∗}	1	出 力	積分値
8	ae	{D∗ / R∗}	1	出 力	絶対誤差推定値 (4.1.1 参照)
9	nev	I*	1	出 力	被積分関数評価回数
10	iwk	I*	idv	ワーク	作業領域 (idv に 0 を入力した場合は 500 の大きさで確保する)
11	wk	{D∗ / R∗}	4 × idv	ワーク	作業領域 (idv に 0 を入力した場合は 2000 の大きさで確保する)
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) a<b

(b) er≥ 誤差判定のための単位 ×64 (既定値にするため, 0.0 を入力する場合は除く)

(c) ea≥ 絶対値最小値 $\times 2^{24}$ (既定値にするため, 0.0 を入力する場合は除く)

(d) idv>1(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.	区間 (b, a) の積分値に −1 を乗じる or 積分値=0.0
1500	制限条件 (b), (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.
2000	細分区間数が idv に達した.	要求された精度の解が得られないままで処理が終了する.
2400	ある細分区間の細分がこれ以上不可能となった.	
2500	解の精度が要求精度に達しない.	
2700	$\varepsilon-$ アルゴリズムによる解の収束が得られない.	
3500	結果の信用性がない (誤差が結果より大きい).	処理を打ち切る.

(6) **注意事項**

(a) 積分区間内では, 関数値 f のオーバフローが生じないよう対策をとる必要がある (たとえば, 特異点での関数値を 0.0 とする).

(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) この関数は 10-21 点の適応型ガウス–クロンロッド則を基本とし, 収束し難い特異点付近では $\varepsilon-$ アルゴリズムによる補外により収束の加束を行って解を得る.

(7) **使用例**

(a) 問 題
$\int_0^1 \frac{\log x}{\sqrt{x}} dx$ を求める.

(b) 入力データ
被積分関数 $f(x)$ に対応する関数名: f
($x = 0.0$ のとき f=0.0 とする).
a=0.0, b=1.0, er=1.0e−8, ea=0.0, idv=0

(c) 主プログラム

```
/*      C interface example for ASL_dhnanl */

#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
{
    if( *x == 0.0 )
        return 0.0;
    else
    {
        return  (log(*x)/sqrt(*x));
    }
}
#ifdef __cplusplus
}
#endif

int main()
{
    double a;
    double b;
    double ers;
    double eas;
    int idvs;
    int idv0=500;
    double q;
    double ae;
    int nev;
    int *iwk;
    double *wk;
    int ierr;
    FILE *fp;

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

    printf( "    *** ASL_dhnanl ***\n" );
    printf( "\n     ** Input **\n\n" );
    idvs=0;
    fscanf( fp, "%lf", &a );
    fscanf( fp, "%lf", &b );
    fscanf( fp, "%lf", &ers );
    fscanf( fp, "%lf", &eas );

    iwk = ( int * )malloc((size_t)( sizeof(int) * idv0 ));
    if( iwk == NULL )
    {
        printf( "no enough memory for array iwk\n" );
        return -1;
    }

    wk = ( double * )malloc((size_t)( sizeof(double) * (4*idv0) ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

    printf( "\ta     = %8.3g\n", a );
    printf( "\tb     = %8.3g\n", b );
    printf( "\ters   = %8.3g\n", ers );
    printf( "\teas   = %8.3g\n", eas );
    printf( "\tidv   =   %6d\n", idvs );

    fclose( fp );

    ierr = ASL_dhnanl(f, a, b, ers, eas, idvs, &q, &ae, &nev, iwk, wk);

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

    printf( "\n\tIntegral Approximation\n" );
    printf( "\t  q   = %8.3g\n", q );
    printf( "\n\tEstimate of Absolute Error\n" );
    printf( "\t  ae  = %8.3g\n", ae );
    printf( "\n\tNumber of Function Evaluations\n" );
    printf( "\t  nev =   %6d\n", nev );
```

```
    free( iwk );
    free(  wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dhnanl ***

 ** Input **

a     =        0
b     =        1
ers   =    1e-08
eas   =        0
idv   =        0

 ** Output **

ierr =      0

Integral Approximation
  q   =       -4

Estimate of Absolute Error
  ae  = 4.25e-13

Number of Function Evaluations
  nev =      357
```

4.2.10 ASL_dhbdfs, ASL_rhbdfs
任意の関数 $f(x)$ と第 1 種 0 次ベッセル関数の積の定積分

(1) **機 能**

m 個の正パラメータ α_i $(i = 1, 2, \cdots, \mathrm{m})$ を与えて, 第 1 種 0 次ベッセル関数 $J_0(\alpha_i x)$ と, 有限区間 [0,1] で連続な関数 $f(x)$ の, 積の定積分 $\displaystyle\int_0^1 J_0(\alpha_i x) f(x) x dx$ を, ガウス積分公式または 2 次精度補間を用いて求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dhbdfs (m, ng, z, isw, f, b, work);

単精度関数:

ierr = ASL_rhbdfs (m, ng, z, isw, f, b, 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	m	I	1	入 力	正パラメータ α_i の個数
2	ng	I	1	入 力	ガウス積分の次数 n
3	z	{ D∗ / R∗ }	m	入 力	正パラメータ α_i (重み関数は $J_0(\alpha_i x)x$ となる.)
4	isw	I	1	入 力	積分法の選択 (注意事項 (a) 参照)
5	f	−	−	入 力	関数 $f(x)$ を定義する関数名 (注意事項 (b),(c) 参照)
6	b	{ D∗ / R∗ }	m	出 力	定積分 $\displaystyle\int_0^1 J_0(\alpha_i x) f(x) x dx$ $(i = 1, \cdots, \mathrm{m})$
7	work	{ D∗ / R∗ }	4×ng	ワーク	作業領域
8	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $\mathrm{m} \geq 1$

(b) $\mathrm{ng} \geq 2$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.
4000	ガウス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) isw=0 のときは, ガウスの ng 点則が用いられ, isw≠0 のときは, 関数 $f(x)$ を $[0,1]$ で 2×ng−2 等分した分点において評価し, 2 次精度で近似し, 解析的に計算する. この方法は z の要素が大きくても問題なく使用できる. しかし, ガウスの積分則がしばしばもたらすような極端な高精度は得られない. すなわち, 2 次精度補間の誤差項 (ng の 3 乗に反比例する) 程度の結果の誤差は避けられない. また, isw $\neq 0$ のとき, 関数 $f(x)$ は, x^2 の滑らかな関数となっているものが望ましい.

(b) f の作り方は次のようにする.
倍精度版
第一引数を変更しないこと

```
void FORTRAN sfun(double *x, double *y)
{
      *y=f(*x);
}
```

単精度版
第一引数を変更しないこと

```
void FORTRAN sfun(float *x, float *y)
{
      *y=f(*x);
}
```

(c) $f(x)$ は, $0 \leq x \leq 1$ において定義されていればよい.

(d) z の要素 (α_i) に 15.0 以上の値が存在する場合は, この関数を isw=0 として使う (ガウス積分を用いる) よりは, 4.2.11 $\left\{\begin{matrix}\text{ASL\_dhbsfc}\\ \text{ASL\_rhbsfc}\end{matrix}\right\}$ を使用した方がよい.

(e) 次数 ng を大きめにとることが精度上望ましい. ただし, 次数 ng を大きくすると計算量が増える.

(f) α_i の値が不明な時は, isw=1 として使用すれば実用的な精度が得られる.

(7) **使用例**

(a) 問 題

α_1=3.8352, α_2=4.1954, ng=50, isw=0 (ガウスの積分公式を用いる),

$$f(r) = 0.854 - 0.483r(2.845 + 1.726r(1.429 + 0.396r(0.472 - 3.172r(1.741 - 2.621r$$

$$(2.459 - 1.637r(1.28 + 3.284r))))))$$

として, $\int_0^1 J_0(\alpha_i x) f(x) x dx$ を求める. isw=1 のときと比較する.

(b) 主プログラム

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

#ifdef __cplusplus
```

```
extern "C"
{
#endif
#ifdef __STDC__
void xtoy(double *x, double *y)
#else
void xtoy(x, y)
double *x;
double *y;
#endif
{
    *y=1.28 +3.284*(*x);
    *y=2.459-1.637*(*x)*(*y);
    *y=1.741-2.621*(*x)*(*y);
    *y=0.472-3.172*(*x)*(*y);
    *y=1.429+0.396*(*x)*(*y);
    *y=2.845+1.726*(*x)*(*y);
    *y=0.854-0.483*(*x)*(*y);
}
#ifdef __cplusplus
}
#endif
int main()
{
    double *aa, *bb, *work;
    int    m,ng,isw,ierr;
    m=2;ng=50;isw=0;
    aa=(double * )malloc((size_t)( sizeof(double)* m ));
    if( aa == NULL )
    {
        printf( "no enough memory for array aa \n" );
        return -1;
    }
    bb=(double * )malloc((size_t)( sizeof(double)* (m*2) ));
    if( bb == NULL )
    {
        printf( "no enough memory for array bb  \n" );
        return -1;
    }
    work=(double * )malloc((size_t)( sizeof(double)* (4*ng) ));
    if( work == NULL )
    {
        printf( "no enough memory for array work  \n" );
        return -1;
    }
    aa[0]=3.8352;
    aa[1]=4.1954;
    printf( "\n\t *** ASL_dhbdfs   \n\n" );
    printf( "\n\t *** input    aa \n\n" );
    printf( "\n%13.8g , %13.8g\n", aa[0],aa[1]);
    ierr=ASL_dhbdfs(m, ng, aa, isw, xtoy, bb, work);
    printf( "\n\t *** OUTPUT ***\n\n" );
    printf( "\n\tierr = %6d\n", ierr );
    printf( "\n\t  bb[0],bb[1]  \n\n" );
    printf( "\n%13.8g , %13.8g\n",bb[0],bb[1]);
    isw=1;
    ierr=ASL_dhbdfs(m, ng, aa, isw, xtoy, &bb[2], work);
    printf( "\n\tierr = %6d\n", ierr );
    printf( "\n%13.8g , %13.8g\n",bb[2],bb[3]);
    printf( "\n%13.8g , %13.8g\n",bb[2]-bb[0],bb[3]-bb[1]);
    free(aa);
    free(bb);
    free(work);
    return 0;
}
```

(c) 出力結果

```
 *** ASL_dhbdfs

 *** input    aa

   3.8352 ,        4.1954
 *** OUTPUT ***

ierr =      0

  bb[0],bb[1]

-0.45758033 ,   -0.48041803
  ierr =      0

-0.45758082 ,    -0.4804185

-4.9076906e-07 , -4.7077319e-07
```

4.2.11 ASL_dhbsfc, ASL_rhbsfc
チェビシェフ多項式と第 1 種 0 次ベッセル関数の積の定積分

(1) **機 能**

0 次から n 次までのチェビシェフ多項式 $T_k(2x-1)$ と, m 個の パラメータ α_i に対する第 1 種 0 次ベッセル関数 $J_0(\alpha_i x)$ の積の定積分 $\int_0^1 J_0(\alpha_i x)T_k(2x-1)xdx(k=0,1,\cdots,n;\ i=1,2,\cdots,\mathrm{m})$ を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dhbsfc (n, m, z, c, nc, work);

単精度関数:

ierr = ASL_rhbsfc (n, m, z, c, nc, 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	チェビシェフ多項式 $T_k(2x-1)$ の次数 k の上限 n
2	m	I	1	入 力	パラメータ α_i の個数
3	z	{D∗ / R∗}	m	入 力	パラメータ α_i
4	c	{D∗ / R∗}	(nc + 1)×m	出 力	定積分 $\int_0^1 J_0(\alpha_i x)T_k(2x-1)xdx(k=0,\cdots,n\ ;\ i=1,2,\cdots,\mathrm{m})$
5	nc	I	1	入 力	配列 c の整合寸法
6	work	{D∗ / R∗}	(n+1)×3	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) $1 \le \mathrm{n} \le \mathrm{nc}$

(b) $\mathrm{m} \ge 1$

(c) z の各要素が正であること.

(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.	
3000	制限条件 (a),(b) または (c) が満足しなかった.	処理を打ち切る.

(6) **注意事項**

(a) z の要素に 15.0 以下の値が存在するときは, その値については 4.2.10 $\left\{\begin{matrix}\text{ASL_dhbdfs}\\ \text{ASL_rhbdfs}\end{matrix}\right\}$を使用した方が安定な計算が行える.

(b) n は 35 程度を限度とする.

(c) 定積分 $\int_0^1 J_0(\alpha x)T_l(2x-1)xdx(l=0,\cdots,\text{n})$ は,c[l+(n+1)*j](α=z[j]) と代入される.

(7) **使用例**

(a) 問 題

α_1=3.8352,α_2=4.1954,n=7 として,$\int_0^1 J_0(\alpha_i x)T_l(2x-1)xdx$ の値を求め, これを利用して,

$$f(r)=0.854-0.483r(2.845+1.726r(1.429+0.396r(0.472-3.172r(1.741-2.621r$$

$$(2.459-1.637r(1.28+3.284r))))))$$

のフーリエ展開係数から $\int_0^1 J_0(\alpha_i x)f(x)xdx$ を求める.

(b) 主プログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <asl.h>
#include <math.h>
int main()
{
    int n,m,ng;
    int i,j,k,ierr;
    double *aa, *cf, *work, *fr, *b, *fn;
    double x,y;
    n=7;m=2;ng=50;
    aa=(double * )malloc((size_t)( sizeof(double)* m ));
    if( aa == NULL )
    {
        printf( "no enough memory for array aa\n" );
        return -1;
    }
    aa[0]=3.8352;
    aa[1]=4.1954;
    cf=(double * )malloc((size_t)( sizeof(double)* (m*(1+n)) ));
    if( cf == NULL )
    {
        printf( "no enough memory for array cf\n" );
        return -1;
    }
    work=(double * )malloc((size_t)( sizeof(double)* (3*(1+n)) ));
    if( work == NULL )
    {
        printf( "no enough memory for array work\n" );
        return -1;
    }
    fr=(double * )malloc((size_t)( sizeof(double)* (1+n) ));
    if( fr == NULL )
    {
        printf( "no enough memory for array fr \n" );
        return -1;
    }
    b=(double * )malloc((size_t)( sizeof(double)* m ));
    if(b  == NULL )
    {
        printf( "no enough memory for array b\n" );
        return -1;
    }
    fn=(double * )malloc((size_t)( sizeof(double)* ng ));
    if( fn == NULL )
    {
        printf( "no enough memory for array fn\n" );
        return -1;
    }
    printf( "\n\t *** ASL_dhbsfc \n\n" );
    printf( "\n\t input \n\n" );
    printf( "\n %13.8g,%13.8g\n",aa[0],aa[1]);
    ierr=ASL_dhbsfc(n, m, aa, cf, n, work);
    printf( "\n\t *** OUTPUT ***\n\n" );
    printf( "\n\tierr = %6d\n", ierr );
    printf( "\n\ti          cf1          cf2  \n\n" );
    for ( i=0 ; i<n+1 ; i++ )
    {
        printf( "\n%9d,%13.8g,%13.8g\n",i,cf[i],cf[i+n+1]);
    }
/*    fourier transform */
    for ( k=0 ; k<ng ; k++ )
```

```
    {
        x=(cos((M_PI*(2*k+1))/(2*ng))+1.0)/2.0;
        y=1.28 +3.284*x;
        y=2.459-1.637*x*y;
        y=1.741-2.621*x*y;
        y=0.472-3.172*x*y;
        y=1.429+0.396*x*y;
        y=2.845+1.726*x*y;
        y=0.854-0.483*x*y;
        fn[k]=y;
    }
    for ( i=0 ; i<n+1 ; i++ )
    {
        fr[i]=0.0;
        for ( k=0 ; k<ng ; k++ )
        {
            fr[i]=fr[i]+fn[k]*cos((i*M_PI*(2*k+1))/(2*ng));
        }
        fr[i]=fr[i]*2.0/ng;
        if(i == 0)
        {
            fr[i]=fr[i]/2.0;
        }
    }
    for ( i=0 ; i<m ; i++ )
    {
        b[i]=0.0;
        for ( j=0 ; j<n+1 ; j++ )
        {
            b[i]=b[i]+fr[j]*cf[j+(n+1)*i];
        }
    }
    printf( "\n\t coefficient \n\n" );
    printf( "\n %13.8g,%13.8g\n",b[0],b[1]);
    free(cf);
    free(work);
    free(fr);
    free(b);
    free(fn);
    free(aa);
    return 0;
}
```

(c) 出力結果

```
 *** ASL_dhbsfc

 input

    3.8352,       4.1954
 *** OUTPUT ***

ierr =      0
i          cf1          cf2

    0,-0.00036676299, -0.032670159
    1, -0.093804025,  -0.10122805
    2, -0.063933641, -0.041917162
    3,  0.075850369,  0.089379895
    4,  0.044986007,  0.041434474
    5,  0.001469729,-0.0034078357
    6, 0.0042640297, 0.0030696751
    7, 0.0040831492, 0.0038106815
 coefficient

-0.45758033,  -0.48041803
```

4.3 半無限区間積分

4.3.1 ASL_dhemnh, ASL_rhemnh
任意の関数

(1) **機 能**

関数の半無限区間積分を自動的に行う. 被積分関数に特異性があってもその性質を判定し, 自動的に処理を行う. 入出力引数は必要最小限とし, 使用しやすくなっている.

(2) **使用法**

倍精度関数:

ierr = ASL_dhemnh (f, a, er, & q, & ae);

単精度関数:

ierr = ASL_rhemnh (f, a, er, & q, & ae);

(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	被積分関数 $f(x)$ を定義する関数名
2	a	{D / R}	1	入 力	積分の下端
3	er	{D / R}	1	入 力	要求相対精度 (既定値 : 誤差判定のための単位 ×64) (4.1.1 参照)
4	q	{D∗ / R∗}	1	出 力	積分値
5	ae	{D∗ / R∗}	1	出 力	絶対誤差推定値 (4.1.1 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) er≥ 誤差判定のための単位 ×64 (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.	
1000	特異点が 5 個以上ある.	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.
2000	細分区間数が 500 に達した.	徐々に最小分割幅を広げ, 近似解を得るようにする.
2400	ある細分区間の細分がこれ以上不可能となった.	要求された精度の解が得られないままで処理が終了する.
2500	解の精度が要求精度に達しない.	
3500	結果の信用性がない (誤差が結果より大きい).	処理を打ち切る.
4000	一つの細分区間で 2 回以上オーバフローが発生した.	

(6) **注意事項**

(a) 被積分関数が特異型のときは, 要求精度を既定値より緩くしないと解の精度が悪くなったり, 特異点の数が実際より多く出力されることがある. 被積分関数が著しいピーク型のときは, 倍精度により要求精度を高くして解くことが望まれる. 被積分関数が振動型のときは, 4.3.2 $\left\{\begin{matrix}\text{ASL_dhnofh}\\ \text{ASL_rhnofh}\end{matrix}\right\}$を使った方が良い. その他のときや分からないときは, 必要とする精度を要求精度として解けばよい.

(b) 変数 er は, 0.0 を入力すれば既定値がセットされる.

(c) この関数は適応型ニュートン・コーツ 9 点則に特異点処理能力を加えたアルゴリズムを基本とする.

(7) **使用例**

(a) 問 題
$\int_2^\infty \frac{1}{x^2}dx$ を求める.

(b) 入力データ
被積分関数 $f(x)$ に対応する関数名: f
a=2.0, er=0.0

(c) 主プログラム

```
/*      C interface example for ASL_dhemnh */

#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 1.0/((*x) * (*x));
}
#ifdef __cplusplus
}
#endif

int main()
{
    double a;
```

```
        double epsrel;
        double result;
        double abserr;
        int ierr;
        FILE *fp;

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

        printf( "    *** ASL_dhemnh ***\n" );
        printf( "\n     ** Input **\n\n" );
        fscanf( fp, "%lf", &a );
        fscanf( fp, "%lf", &epsrel );

        printf( "\ta   = %8.3g\n", a );
        printf( "\ter  = %8.3g\n", epsrel );

        fclose( fp );

        ierr = ASL_dhemnh(f, a, epsrel, &result, &abserr);

        printf( "\n     ** Output **\n\n" );
        printf( "\tierr = %6d\n", ierr );
        printf( "\n\tIntegral Approximation\n" );
        printf( "\t  q   = %8.3g\n", result );
        printf( "\n\tEstimate of Absolute Error\n" );
        printf( "\t  ae  = %8.3g\n", abserr );

        return 0;
    }
```

(d) 出力結果

```
    *** ASL_dhemnh ***

     ** Input **

    a   =        2
    er  =        0

     ** Output **

    ierr =      0

    Integral Approximation
      q   =      0.5

    Estimate of Absolute Error
      ae  = 8.88e-16
```

4.3.2 ASL_dhnofh, ASL_rhnofh

$f(x)\cdot(\sin\omega x$ **or** $\cos\omega x)$ 型の関数

(1) **機 能**

振動型関数のうち, $f(x)\cdot(\sin\omega x$ or $\cos\omega x)$ の形に因数分解できる関数を半無限区間積分する.

(2) **使用法**

倍精度関数:

ierr = ASL_dhnofh (f, a, w, itype, ea, isy, idv, & q, & ae, & nev, & iwk, & wk);

単精度関数:

ierr = ASL_rhnofh (f, a, w, itype, ea, isy, idv, & q, & ae, & 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: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 }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	f	{D / R}	–	入 力	被積分関数 $f(x)$ を定義する関数名
2	a	{D / R}	1	入 力	積分の下端
3	w	{D / R}	1	入 力	重みの関数 $(\sin\omega x, \cos\omega x)$ での ω
4	itype	I	1	入 力	重みの関数の区別 $1\cdots\int f(x)\cos\omega x dx$ $2\cdots\int f(x)\sin\omega x dx$
5	ea	{D / R}	1	入 力	要求絶対精度 (既定値 : 絶対値最小値 $\times 2^{24}$) (4.1.1 参照)
6	isy	I	1	入 力	最大反復回数 (変換後の最小分割幅は, $0.25\times 2^{-\mathrm{isy}}$, 既定値 8)
7	idv	I	1	–	未使用
8	q	{D∗ / R∗}	1	出 力	積分値
9	ae	{D∗ / R∗}	1	出 力	絶対誤差推定値 (4.1.1 参照)
10	nev	I*	1	出 力	被積分関数評価回数
11	iwk	I*	1	ワーク	作業領域 (未使用) , ダミーポインタを渡す.
12	wk	{D∗ / R∗}	1	ワーク	作業領域 (未使用), ダミーポインタを渡す.
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) ea≥ 絶対値最小値 $\times 2^{24}$ (既定値にするため, 0.0 を 入力する場合は除く)

(b) 2<isy<51(既定値にするため, 0 を入力する場合は除く)

(c) itype=1, 2

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
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.
2100	反復回数が isy 回に達した.	要求された精度の解が得られないままで, 処理が終了する.
2500	解の精度が要求精度に達しない.	
3000	制限条件 (c) を満足しなかった.	処理を打ち切る.
3100	DE 変換後, +− の両側で関数値が十分小さくならない.	
3500	結果の信用性がない (誤差が結果より大きい).	

(6) **注意事項**

(a) 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.

(b) この関数は, 振動型半無限積分に対する変数変換型公式をもとに計算する.

(c) 要求絶対精度に達しないときは, 相対精度として誤差判定のための単位 $\times 64$ をもって収束判定を行い解を返却する.

(7) **使用例**

(a) 問 題

$\displaystyle\int_0^\infty \frac{\sin x}{x}dx$ を求める.

(b) 入力データ

被積分関数 $f(x)$ に対応する関数名: f

($x = 0.0$ のとき f=0.0 とする).

a=0.0, w=1.0, itype=2, ea=1.0e−8, isy=0

(c) 主プログラム

```
/*      C interface example for ASL_dhnofh */

#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
{
    if( *x == 0.0 )
        return 0.0;
    else
```

```
        return 1.0/(*x);
}
#ifdef __cplusplus
}
#endif

int main()
{
    double a;
    double om;
    int ity;
    double epsab;
    int lims;
    int lit=0; /* dummy */
    double s;
    double abser;
    int neval;
    int ierr;
    FILE *fp;

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

    printf( "    *** ASL_dhnofh ***\n" );
    printf( "\n     ** Input **\n\n" );
    fscanf( fp, "%lf", &a );
    fscanf( fp, "%lf", &om );
    fscanf( fp, "%d", &ity );
    fscanf( fp, "%lf", &epsab );
    fscanf( fp, "%d", &lims );

    printf( "\ta     = %8.3g\n", a );
    printf( "\tw     = %8.3g\n", om );
    printf( "\titype =   %6d\n", ity );
    printf( "\tea    = %8.3g\n", epsab );
    printf( "\tisy   =   %6d\n", lims );

    fclose( fp );

    ierr = ASL_dhnofh
    (f, a, om, ity, epsab, lims, lit, &s, &abser, &neval, NULL, NULL);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tIntegral Approximation\n" );
    printf( "\t  q   = %8.3g\n", s );
    printf( "\n\tEstimate of Absolute Error\n" );
    printf( "\t  ae  = %8.3g\n", abser );
    printf( "\n\tNumber of Function Evaluations\n" );
    printf( "\t  nev = %6d\n", neval );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dhnofh ***

 ** Input **

a     =        0
w     =        1
itype =        2
ea    =    1e-08
isy   =        0

 ** Output **

ierr =      0

Integral Approximation
  q   =     1.57

Estimate of Absolute Error
  ae  = 5.69e-10

Number of Function Evaluations
  nev =     73
```

4.3.3 ASL_dhnenh, ASL_rhnenh
端点特異型関数

(1) **機 能**

端点 a に特異性のある関数の半無限区間積分をする.

(2) **使用法**

倍精度関数:

ierr = ASL_dhnenh (f, a, er, ea, itmx, & q, & ae, & nev, isw);

単精度関数:

ierr = ASL_rhnenh (f, a, er, ea, itmx, & q, & ae, & nev, 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	f	{D / R}	–	入 力	被積分関数 $f(x)$ を定義する関数名
2	a	{D / R}	1	入 力	積分の下端
3	er	{D / R}	1	入 力	要求相対精度 (既定値 : 誤差判定のための単位 × 64) (4.1.1 参照)
4	ea	{D / R}	1	入 力	要求絶対精度 (既定値 : 絶対値最小値 $\times 2^{24}$) (4.1.1 参照)
5	itmx	I	1	入 力	特異点間の最大反復回数 (DE 変換後の最小分割幅は, $0.25 \times 2^{-\mathrm{itmx}}$, 既定値 8)
6	q	{D∗ / R∗}	1	出 力	積分値
7	ae	{D∗ / R∗}	1	出 力	絶対誤差推定値 (4.1.1 参照)
8	nev	I*	1	出 力	被積分関数評価回数
9	isw	I	1	入 力	≤ 0 : 被積分関数内に e^{-x} の因子を持たない, または不明のとき ≥ 1 : 被積分関数内に e^{-x} の因子をもつとき
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) er≥ 誤差判定のための単位 ×64 (既定値にするため, 0.0 を入力する場合は除く)

(b) ea≥ 絶対値最小値 $\times 2^{24}$ (既定値にするため, 0.0 を入力する場合は除く)

(c) itmx>1(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), (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.
2000	反復回数が itmx 回に達した.	要求された精度の解が得られないままで, 処理が終了する.
2500	解の精度が要求精度に達しない.	
3100	DE 変換後, +− の両側で関数値が十分小さくならない.	処理を打ち切る.
3500	結果の信用性がない (誤差が結果より大きい).	

(6) **注意事項**

(a) 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.

(b) この関数は, 2 重指数関数型公式 (DE 変換公式) を利用する.

(7) **使用例**

(a) 問 題
$\int_0^\infty e^{-x}\log(x)dx$ を求める.

(b) 入力データ
被積分関数 $f(x)$ に対応する関数名: f
($x = 0.0$ のとき f=0.0 とする).
a=0.0, er=1.0e−8, ea=0.0, itmx=0, isw=1

(c) 主プログラム

```
/*      C interface example for ASL_dhnenh */

#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
{
    if( *x == 0.0 )
        return 0.0;
    else
        return exp(-(*x))*log(*x);
}
#ifdef __cplusplus
}
#endif

int main()
{
    double a;
    double epsr;
    double epsa;
    int lim;
    double result;
    double abserr;
    int neval;
    int isw;
    int ierr;
```

```
    FILE *fp;

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

    printf( "    *** ASL_dhnenh ***\n" );
    printf( "\n     ** Input **\n\n" );
    fscanf( fp, "%lf", &a );
    fscanf( fp, "%lf", &epsr );
    fscanf( fp, "%lf", &epsa );
    fscanf( fp, "%d", &lim );
    fscanf( fp, "%d", &isw );

    printf( "\ta     = %8.3g\n", a );
    printf( "\ter    = %8.3g\n", epsr );
    printf( "\tea    = %8.3g\n", epsa );
    printf( "\titmx  =   %6d\n", lim );
    printf( "\tisw   =   %6d\n", isw );

    fclose( fp );

    ierr = ASL_dhnenh(f, a, epsr, epsa, lim, &result, &abserr, &neval, isw);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tIntegral Approximation\n" );
    printf( "\t  q   = %8.3g\n", result );
    printf( "\n\tEstimate of Absolute Error\n" );
    printf( "\t  ae  = %8.3g\n", abserr );
    printf( "\n\tNumber of Function Evaluations\n" );
    printf( "\t  nev = %6d\n", neval );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dhnenh ***

 ** Input **

a     =        0
er    =    1e-08
ea    =        0
itmx  =        0
isw   =        1

 ** Output **

ierr =      0

Integral Approximation
  q   =   -0.577

Estimate of Absolute Error
  ae  = 2.49e-12

Number of Function Evaluations
  nev =     81
```

4.3.4 ASL_dhninh, ASL_rhninh
内点特異型関数

(1) **機 能**

積分区間内点に特異性のある関数を半無限区間積分する.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dhninh  (f, a, sp, nsp, er, ea, itmx, & q, & ae, & nev);
```

単精度関数:

```
ierr = ASL_rhninh  (f, a, sp, nsp, er, ea, itmx, & q, & ae, & 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	被積分関数 $f(x)$ を定義する関数名
2	a	{D / R}	1	入 力	積分の下端
3	sp	{D∗ / R∗}	nsp	入 力	特異点の X 座標値
				出 力	昇順にソートされた座標値
4	nsp	I	1	入 力	特異点の数
5	er	{D / R}	1	入 力	要求相対精度 (既定値 : 誤差判定のための単位 × 64) (4.1.1 参照)
6	ea	{D / R}	1	入 力	要求絶対精度 (既定値 : 絶対値最小値 $\times 2^{24}$) (4.1.1 参照)
7	itmx	I	1	入 力	特異点間の最大反復回数 (DE 変換後の最小分割幅は, $0.25 \times 2^{-\mathrm{itmx}}$, 既定値 8)
8	q	{D∗ / R∗}	1	出 力	積分値
9	ae	{D∗ / R∗}	1	出 力	絶対誤差推定値 (4.1.1 参照)
10	nev	I*	1	出 力	被積分関数評価回数
11	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) er≥ 誤差判定のための単位 ×64 (既定値にするため, 0.0 を入力する場合は除く)

(b) ea≥ 絶対値最小値 $\times 2^{24}$ (既定値にするため, 0.0 を入力する場合は除く)

(c) itmx>1(既定値にするため, 0 を入力する場合は除く)

(d) nsp> 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), (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.
2000	一区間の反復回数が itmx 回に達した.	要求された精度の解が得られないままで, 処理が終了する.
2300	ある区間での積分精度が悪い.	
2500	解の精度が要求精度に達しない.	
3000	制限条件 (d) を満足しなかった.	処理を打ち切る.
3100	DE 変換後, +− の両側で関数値が十分小さくならない.	
3500	結果の信用性がない (誤差が結果より大きい).	

(6) **注意事項**

(a) 積分区間内では, 関数値のオーバフローが生じないよう対策をとる必要がある (たとえば, 特異点での関数値を 0.0 とする).

(b) 特異点での特異性が著しい場合は, 要求精度を満足せず, 単精度では 2 桁, 倍精度では 4 桁しか求まらないことがある. したがって, より高精度を要求する場合には, 特異点で分割し, 有限区間は桁落ち防止変換し, 4.2.7 $\left\{\begin{matrix}\text{ASL_dhnenl}\\ \text{ASL_rhnenl}\end{matrix}\right\}$を利用し, さらに半無限区間は 4.3.3 $\left\{\begin{matrix}\text{ASL_dhnenh}\\ \text{ASL_rhnenh}\end{matrix}\right\}$により積分する必要がある.

(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) この関数は, 半無限区間と有限区間の 2 重指数関数型公式 (DE 変換公式) を用いて各特異点間ごとの積分値を求め, これらを加算し全体の積分値を計算する.

(7) **使用例**

(a) 問 題
$\displaystyle\int_{-1}^{\infty}\frac{x}{e^x-1}dx$ を求める.

(b) 入力データ
被積分関数 $f(x)$ に対応する関数名: f
($x=0.0$ のとき f=0.0 とする).
a=−1.0, sp [0] =0.0, nsp=1, er=1.0e−3, ea=0.0, itmx=0

(c) 主プログラム

```
/*      C interface example for ASL_dhninh */

#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
```

```
{
    if( *x == 0.0 )
        return 0.0;
    else
        return (*x)/( expm1(*x) );
}
#ifdef __cplusplus
}
#endif

int main()
{
    double a;
    double *point;
    int npts;
    double epsrel;
    double epsabs;
    int limit;
    double result;
    double abserr;
    int neval;
    int ierr;
    int i;
    FILE *fp;

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

    printf( "    *** ASL_dhninh ***\n" );
    printf( "\n     ** Input **\n\n" );
    npts=1;

    point = ( double * )malloc((size_t)( sizeof(double) * npts ));
    if( point == NULL )
    {
        printf( "no enough memory for array point\n" );
        return -1;
    }
    fscanf( fp, "%lf", &a );
    for( i=0 ; i<npts ; i++ )
    {
            fscanf( fp, "%lf", &point[i] );
    }
    fscanf( fp, "%lf", &epsrel );
    fscanf( fp, "%lf", &epsabs );
    fscanf( fp, "%d", &limit );

    printf( "\ta         = %8.3g\n", a );
    for( i=0 ; i<npts ; i++ )
    {
            printf( "\tsp[%6d]= %8.3g\n", i,point[i] );
    }
    printf( "\ter        = %8.3g\n", epsrel );
    printf( "\tea        = %8.3g\n", epsabs );
    printf( "\titmx      =   %6d\n", limit );

    fclose( fp );

    ierr = ASL_dhninh(f, a, point, npts, epsrel, epsabs, limit, &result, &abserr, &neval);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tSorted X-Coordinate Value of The Singular Point\n" );
    for( i=0 ; i<npts ; i++ )
    {
            printf( "\t  sp[%6d]= %8.3g\n", i,point[i] );
    }
    printf( "\n\tIntegral Approximation\n" );
    printf( "\t  q   = %8.3g\n", result );
    printf( "\n\tEstimate of Absolute Error\n" );
    printf( "\t  ae  = %8.3g\n", abserr );
    printf( "\n\tNumber of Function Evaluations\n" );
    printf( "\t  nev = %6d\n", neval );

    free( point );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dhninh ***

 ** Input **

a         =       -1
sp[     0]=        0
er        =   0.0001
ea        =        0
itmx      =        0
```

```
 ** Output **

ierr =      0

Sorted X-Coordinate Value of The Singular Point
  sp[     0]=        0

Integral Approximation
  q    =     2.92

Estimate of Absolute Error
  ae   = 3.31e-05

Number of Function Evaluations
  nev =     78
```

4.4 全無限区間積分

4.4.1 ASL_dhemni, ASL_rhemni
任意の関数

(1) **機 能**

$-\infty \sim \infty$ の全無限区間積分を自動的に行う. 内点に特異性があってもよい. 入出力引数は必要最小限とし, 使いやすくしている.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dhemni  (f, er, & q, & ae);
```

単精度関数:

```
ierr = ASL_rhemni  (f, er, & q, & ae);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	f	{D / R}	−	入 力	被積分関数 $f(x)$ を定義する関数名
2	er	{D / R}	1	入 力	要求相対精度 (既定値 : 誤差判定のための単位 $\times 64$) (4.1.1 参照)
3	q	{D∗ / R∗}	1	出 力	積分値
4	ae	{D∗ / R∗}	1	出 力	絶対誤差推定値 (4.1.1 参照)
5	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) er$\geq$ 誤差判定のための単位 $\times 64$ (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) を満足しなかった.	既定値にセットして処理する.
2000	細分区間数が 500 に達した.	徐々に最小分割幅を広げ, 近似解を得るようにする.
2400	ある細分区間の細分がこれ以上不可能となった.	要求された精度の解が得られないままで処理が終了する.
2500	解の精度が要求精度に達しない.	
3500	結果の信用性がない (誤差が結果より大きい).	処理を打ち切る.
4000	一つの細分区間で 2 回以上オーバフローが発生した.	

(6) **注意事項**

(a) 被積分関数が特異型のときは, 要求精度を既定値より緩くしないと解の精度が悪くなったり, 特異点の数が実際より多く出力されることがある.
被積分関数が著しいピーク型のときは, 倍精度により要求精度を高くして解くことが望まれる. 被積分関数が振動型のときは, 4.4.2 $\left\{\begin{matrix}\text{ASL_dhnofi}\\ \text{ASL_rhnofi}\end{matrix}\right\}$を使った方が良い. その他のときや分からないときは, 必要とする精度を要求精度として解けばよい.

(b) 変数 er は, 0.0 を入力すれば既定値がセットされる.

(c) この関数は適応型ニュートン・コーツ 9 点則に特異点処理能力を加えたアルゴリズムを基本とし, 変数変換により全無限区間積分を行う.

(7) **使用例**

(a) 問 題
$\displaystyle\int_{-\infty}^{\infty}\frac{1}{1+x^2}dx$ を求める.

(b) 入力データ
被積分関数 $f(x)$ に対応する関数名: f
er=0.0

(c) 主プログラム

```
/*      C interface example for ASL_dhemni */

#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 1.0/(1.0 + (*x) * (*x));
}
#ifdef __cplusplus
}
#endif
```

```
int main()
{
    double epsr;
    double result;
    double abserr;
    int ierr;
    FILE *fp;

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

    printf( "    *** ASL_dhemni ***\n" );
    printf( "\n     ** Input **\n\n" );
    fscanf( fp, "%lf", &epsr );

    printf( "\ter  = %8.3g\n", epsr );

    fclose( fp );

    ierr = ASL_dhemni(f, epsr, &result, &abserr);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tIntegral Approximation\n" );
    printf( "\t  q   = %8.3g\n", result );
    printf( "\n\tEstimate of Absolute Error\n" );
    printf( "\t  ae  = %8.3g\n", abserr );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dhemni ***

 ** Input **

er  =        0

 ** Output **

ierr =      0

Integral Approximation
  q   =     3.14

Estimate of Absolute Error
  ae  = 5.58e-15
```

4.4.2 ASL_dhnofi, ASL_rhnofi

$f(x)\cdot(\sin\omega x$ **or** $\cos\omega x)$ 型の関数

(1) **機 能**

振動型関数で $f(x)\cdot(\sin\omega x$ or $\cos\omega x)$ の形に因数分解できる関数の全無限積分をする.

(2) **使用法**

倍精度関数:

ierr = ASL_dhnofi (f, w, itype, ea, isy, idv, & q, & ae, & nev, iwk, & wk);

単精度関数:

ierr = ASL_rhnofi (f, w, itype, ea, isy, idv, & q, & ae, & 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	f	{D / R}	–	入 力	被積分関数の因数 $f(x)$ を定義する関数名
2	w	{D / R}	1	入 力	重みの関数 $(\sin\omega x, \cos\omega x)$ での ω
3	itype	I	1	入 力	重みの関数の区別 $1\cdots\int f(x)\cos\omega x dx$ $2\cdots\int f(x)\sin\omega x dx$
4	ea	{D / R}	1	入 力	要求絶対精度 (既定値：絶対値最小値 $\times 2^{24}$) (4.1.1 参照)
5	isy	I	1	入 力	最大反復回数 (変換後の最小分割幅は, $0.25\times 2^{-\mathrm{isy}}$, 既定値 8)
6	idv	I	1	–	未使用
7	q	{D∗ / R∗}	1	出 力	積分値
8	ae	{D∗ / R∗}	1	出 力	絶対誤差推定値 (4.1.1 参照)
9	nev	I*	1	出 力	被積分関数評価回数
10	iwk	I*	1	ワーク	作業領域 (未使用) , ダミーポインタを渡す.
11	wk	{D∗ / R∗}	1	ワーク	作業領域 (未使用), ダミーポインタを渡す.
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) ea≥ 絶対値最小値 $\times 2^{24}$ (既定値にするため, 0.0 を入力する場合は除く)

(b) 2<isy<51(既定値にするため, 0 を入力する場合は除く)

(c) itype=1, 2

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
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.
2100	反復回数が isy に達した.	要求された精度の解が得られないままで, 処理が終了する.
2500	解の精度が要求精度に達しない.	
3000	制限条件 (c) を満足しなかった.	処理を打ち切る.
3100	DE 変換後 $+-$ の両側で関数が十分小さくならない.	
3500	結果の信用性がない (誤差が結果より大きい).	

(6) **注意事項**

(a) 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.

(b) この関数は, 振動型半無限積分に対する変数変換型公式をもとに $(-\infty, 0]$ 区間と $[0, \infty)$ 区間の積分値を求め, これを加算して全区間の積分値を計算する.

(c) 要求絶対精度に達しないときは, 相対精度として誤差判定のための単位 $\times 64$ をもって収束判定を行い解を返却する.

(7) **使用例**

(a) 問 題
$\int_{-\infty}^{\infty} \frac{\sin x}{x} dx$ を求める.

(b) 入力データ
被積分関数 $f(x)$ に対応する関数名: f
($x = 0.0$ のとき f=0.0 とする).
w=1.0, itype=2, ea=1.0e−8, isy=0

(c) 主プログラム

```
/*      C interface example for ASL_dhnofi */

#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
{
    if( *x == 0.0 )
```

```
        return 0.0;
    else
        return 1.0/(*x);
}
#ifdef __cplusplus
}
#endif

int main()
{
    double omega;
    int integr;
    double epsabs;
    int limst;
    int limit=0;  /* dummy */
    double result;
    double abserr;
    int neval;
    int ierr;
    FILE *fp;

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

    printf( "    *** ASL_dhnofi ***\n" );
    printf( "\n     ** Input **\n\n" );
    fscanf( fp, "%lf", &omega );
    fscanf( fp, "%d", &integr );
    fscanf( fp, "%lf", &epsabs );
    fscanf( fp, "%d", &limst );

    printf( "\tw     = %8.3g\n", omega );
    printf( "\titype =   %6d\n", integr );
    printf( "\tea    = %8.3g\n", epsabs );
    printf( "\tisy   =   %6d\n", limst );

    fclose( fp );

    ierr = ASL_dhnofi
    (f, omega, integr, epsabs, limst, limit,
        &result, &abserr, &neval, NULL, NULL);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tIntegral Approximation\n" );
    printf( "\t  q   = %8.3g\n", result );
    printf( "\n\tEstimate of Absolute Error\n" );
    printf( "\t  ae  = %8.3g\n", abserr );
    printf( "\n\tNumber of Function Evaluations\n" );
    printf( "\t  nev = %6d\n", neval );


    return 0;
}
```

(d) 出力結果

```
*** ASL_dhnofi ***

 ** Input **

w     =        1
itype =        2
ea    =    1e-08
isy   =        0

 ** Output **

ierr =      0

Integral Approximation
  q   =     3.14

Estimate of Absolute Error
  ae  = 3.29e-10

Number of Function Evaluations
  nev =    156
```

4.4.3 ASL_dhnini, ASL_rhnini
内点特異型関数

(1) **機 能**

特異点のある関数の全無限区間積分をする.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dhnini  (f, sp, nsp, er, ea, itmx, & q, & ae, & nev);
```

単精度関数:

```
ierr = ASL_rhnini  (f, sp, nsp, er, ea, itmx, & q, & ae, & 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	被積分関数 $f(x)$ を定義する関数名
2	sp	{D∗ / R∗}	nsp	入 力	特異点の X 座標値
				出 力	昇順にソートされた座標値
3	nsp	I	1	入 力	特異点の数
4	er	{D / R}	1	入 力	要求相対精度 (既定値 : 誤差判定のための単位 × 64) (4.1.1 参照)
5	ea	{D / R}	1	入 力	要求絶対精度 (既定値 : 絶対値最小値 $\times 2^{24}$) (4.1.1 参照)
6	itmx	I	1	入 力	一区間での特異点間の最大反復回数 (DE 変換後の最小分割幅は, $0.25\times 2^{-\mathrm{itmx}}$, 既定値 8)
7	q	{D∗ / R∗}	1	出 力	積分値
8	ae	{D∗ / R∗}	1	出 力	絶対誤差推定値 (4.1.1 参照)
9	nev	I*	1	出 力	被積分関数評価回数
10	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) er≥ 誤差判定のための単位 ×64 (既定値にするため, 0.0 を入力する場合は除く)

(b) ea≥ 絶対値最小値 $\times 2^{24}$ (既定値にするため, 0.0 を入力する場合は除く)

(c) itmx>1(既定値にするため, 0 を入力する場合は除く)

(d) nsp>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), (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.
2000	一区間の細分区間数が itmx 回に達した.	要求された精度の解が得られないままで, 処理が終了する.
2300	ある区間での積分精度が悪い.	
2400	ある細分区間の細分がこれ以上不可能となった.	
2500	解の精度が要求精度に達しない.	
3000	制限条件 (d) を満足しなかった.	処理を打ち切る.
3100	DE 変換後, $+-$ の両側で関数値が十分小さくならない.	
3500	結果の信用性がない (誤差が結果より大きい).	

(6) **注意事項**

(a) 積分区間内では, 関数値 f のオーバフローが生じないよう対策をとる必要がある (たとえば, 特異点での関数値を 0.0 とする).

(b) 特異点での特異性が著しい場合は, 要求精度を満足せず, 単精度では 2 桁, 倍精度では 4 桁程度しか求まらないことがある. したがって, より高精度を要求する場合には, 特異点で分割し, 有限区間については桁落ち防止変換し, 4.2.7 $\left\{\begin{array}{l}\text{ASL_dhnenl}\\ \text{ASL_rhnenl}\end{array}\right\}$を利用して積分する必要がある.

(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) この関数は, 半無限区間と有限区間の 2 重指数関数型公式 (DE 変換公式) を用いて各特異点間ごとの積分値を求め, これらを加算し全体の積分値を計算する.

(7) **使用例**

(a) 問 題

$\displaystyle\int_{-\infty}^{\infty} f(x)dx$ を求める. $f(x) = \left\{\begin{array}{ll}\frac{1}{x^2} & (|x|>2)\\ x+2 & (|x|\le 2)\end{array}\right.$

(b) 入力データ

被積分関数 $f(x)$ に対応する関数名: f

sp [0] $=-2.0$, sp [1] $=2.0$, nsp=2, er=1.0e$-$8, ea=0.0, itmx=0

(c) 主プログラム

```
/*      C interface example for ASL_dhnini */

#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
{
    if( fabs(*x) > 2.0 )
        return (1.0/( (*x) * (*x) ));
    else
        return ((*x)+2.0) ;
}
#ifdef __cplusplus
}
#endif

int main()
{
    double *point;
    int npts;
    double epsrel;
    double epsabs;
    int limit;
    double result;
    double abserr;
    int neval;
    int ierr;
    int i;
    FILE *fp;

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

    printf( "    *** ASL_dhnini ***\n" );
    printf( "\n     ** Input **\n\n" );
    npts=2;

    point = ( double * )malloc((size_t)( sizeof(double) * npts ));
    if( point == NULL )
    {
        printf( "no enough memory for array point\n" );
        return -1;
    }
    for( i=0 ; i<npts ; i++ )
    {
            fscanf( fp, "%lf", &point[i] );
    }
    fscanf( fp, "%lf", &epsrel );
    fscanf( fp, "%lf", &epsabs );
    fscanf( fp, "%d", &limit );

    for( i=0 ; i<npts ; i++ )
    {
            printf( "\tsp[%6d]= %8.3g\n", i,point[i] );
    }
    printf( "\ter        = %8.3g\n", epsrel );
    printf( "\tea        = %8.3g\n", epsabs );
    printf( "\titmx      =   %6d\n", limit );

    fclose( fp );

    ierr = ASL_dhnini(f, point, npts, epsrel, epsabs, limit, &result, &abserr, &neval);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tSorted X-Coordinate Value of The Singular Point\n\n" );
    for( i=0 ; i<npts ; i++ )
    {
            printf( "\t  sp[%6d]= %8.3g\n", i,point[i] );
    }
    printf( "\n\tIntegral Approximation\n\n" );
    printf( "\t  q   = %8.3g\n", result );
    printf( "\n\tEstimate of Absolute Error\n\n" );
    printf( "\t  ae  = %8.3g\n", abserr );
    printf( "\n\tNumber of Function Evaluations\n\n" );
    printf( "\t  nev = %6d\n", neval );

    free( point );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dhnini ***

 ** Input **

sp[     0]=       -2
sp[     1]=        2
er        =        0
ea        =        0
itmx      =        0
```

```
 ** Output **
ierr =      0
Sorted X-Coordinate Value of The Singular Point
  sp[     0]=       -2
  sp[     1]=        2
Integral Approximation
  q   =        9
Estimate of Absolute Error
  ae  = 3.02e-14
Number of Function Evaluations
  nev =    350
```

4.4.4 ASL_dh2int, ASL_rh2int
$e^{-x^2} \cdot f(x)$ 型の関数

(1) **機 能**

重み e^{-x^2} の付いた全無限区間積分

$$\int_{-\infty}^{\infty} e^{-x^2} f(x)dx$$

を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dh2int  (n, f, & w, work);
```

単精度関数:

```
ierr = ASL_rh2int  (n, f, & 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	ガウス積分の次数
2	f	–	–	入 力	関数 $f(x)$ を定義する関数名
3	w	{ D∗ / R∗ }	1	出 力	定積分 $\int_{-\infty}^{\infty} e^{-x^2} f(x)dx$
4	work	{ D∗ / R∗ }	3×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) $n \geq 2$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) が満足しなかった.	処理を打ち切る.
4000	ガウス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) f の作り方は次のようにする.
倍精度版

```
void FORTRAN sfun(double *x, double *y)
{
/***  使用例では *y=1.0/(1.0+(*x)*(*x)); ***/
      *y=f(*x);
}
```

単精度版

```
void FORTRAN sfun(float *x, float *y)
{
/***  使用例では *y=1.0/(1.0+(*x)*(*x)); ***/
      *y=f(*x);
}
```

(b) $f(x)$ は,$-7 < x < 7$ において定義されていればよい. すなわち,

$$\int_{-\infty}^{-7} e^{-x^2} dx + \int_{7}^{\infty} e^{-x^2} dx < 7.415 \cdot 10^{-23}$$

であるので, $x \leq -7$ および $x \geq 7$ に対する関数値 $f(x)$ は無視される. $f(x)$ が平均絶対値の 10^7 倍程度の絶対値にまで達し得る場合, 数値積分を適用して積分値を求めると結果が保証されない. この様な場合であっても $7.4 \cdot 10^{-16}$ 程度の相対精度を保証するには, $x \leq -7$ および $x \geq 7$ の関数値は無視できる.

(7) **使用例**

(a) 問 題

無限積分

$$\int_{-\infty}^{\infty} \frac{e^{-x^2}}{x^2+1} dx$$

(真の値は $\pi e \operatorname{Erfc}(1)$) を求める.

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
void fn(double *x, double *y)
#else
void fn(x, y)
double *x;
double *y;
#endif
{
    *y=1.0/((*x)*(*x)+1.0);
}
#ifdef __cplusplus
}
#endif
#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
void fn1(double *x1, double *y1)
#else
void fn1(x1, y1)
double *x1;
double *y1;
#endif
{
    double x,y,z,z2;
    x=*x1;
    z=x;
    if(z<0.0)
    {  z=-z;}
    z2=z*z;
    y=z2*z2;
    y=y*y*z2;
    *y1=y*z;
```

```
}
#ifdef __cplusplus
}
#endif
int main()
{
    int n,ierr;
    double w, *work;
    double v1,v2,one,verfc;
    n=24;
    one=1.0;
    work=(double * )malloc((size_t)( sizeof(double)* (3*n) ));
    if( work == NULL )
    {
        printf( "no enough memory for array work  \n" );
        return -1;
    }
    printf( "\n\t *** ASL_dh2int   \n\n" );
    ierr=ASL_dh2int(n, fn, &w, work);
    printf( "\n\t *** OUTPUT ***\n\n" );
    printf( "\n\tierr = %6d\n", ierr );
    printf( "\n\t *** value 1   true1  \n\n" );
    ierr=ASL_wierfc(1,&one, &verfc);
    v1=M_PI*exp(one)*verfc;
    printf( "\n%13.8g,%13.8g\n",w,v1);
    ierr=ASL_dh2int(n, fn1, &w, work);
    printf( "\n\t *** OUTPUT ***\n\n" );
    printf( "\n\tierr = %6d\n", ierr );
    v2=120;
    printf( "\n\t *** value 2   true2  \n\n" );
    printf( "\n%13.8g,%13.8g\n",w,v2);
    free(work);
    return 0;
}
```

(b) 出力結果

```
 *** ASL_dh2int

 *** OUTPUT ***

ierr =      0

 *** value 1   true1

1.3432934,    1.3432934

 *** OUTPUT ***

ierr =      0

 *** value 2   true2

      120,          120
```

4.5 2 次元有限区間積分

4.5.1 ASL_dhnrnm, ASL_rhnrnm
矩形領域の 2 次元積分

(1) **機 能**

矩形領域の 2 次元積分を自動的に行う.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dhnrnm  (f, a, b, c, d, er, ea, idv, & q, & ae, & nev);
```

単精度関数:

```
ierr = ASL_rhnrnm  (f, a, b, c, d, er, ea, idv, & q, & ae, & nev);
```

(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	被積分関数 $f(x,y)$ を定義する関数名
2	a	{D / R}	1	入 力	X 軸方向積分下端
3	b	{D / R}	1	入 力	X 軸方向積分上端
4	c	{D / R}	1	入 力	Y 軸方向積分下端
5	d	{D / R}	1	入 力	Y 軸方向積分上端
6	er	{D / R}	1	入 力	要求相対精度 (既定値 : 誤差判定のための単位 $\times 64$) (4.1.1 参照)
7	ea	{D / R}	1	入 力	要求絶対精度 (既定値 : 絶対値最小値 $\times 2^{24}$) (4.1.1 参照)
8	idv	I	1	入 力	正常処理を行う細分区間数の最大値 (既定値:5000)
9	q	{D∗ / R∗}	1	出 力	積分値
10	ae	{D∗ / R∗}	1	出 力	絶対誤差推定値 (4.1.1 参照)
11	nev	I*	1	出 力	被積分関数評価回数
12	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) a<b, c<d

(b) er≥ 誤差判定のための単位 ×64 (既定値にするため, 0.0 を入力する場合は除く)

(c) ea≥ 絶対値最小値 $\times 2^{24}$ (既定値にするため, 0.0 を入力する場合は除く)

(d) idv>1(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.	積分値に −1 を乗じる or 積分値=0 (d<c かつ b<a のときは, そのまま).
1500	制限条件 (b), (c) または (d) を満足しなかった.	既定値にセットして処理を続ける.
2000	細分区間数が idv に達した.	徐々に最小分割幅を広げ近似解を得るようにする.
2400	ある細分区間の細分がこれ以上不可能となった.	要求された精度の解が得られないままで処理が終了する.
2500	解の精度が要求度に達しない.	
3500	結果の信用性がない (誤差が結果より大きい).	処理を打ち切る.
4000	一つの細分区間で 2 回以上オーバフローが発生した.	

(6) **注意事項**

(a) 非常に狭い範囲にピークがある場合は, 倍精度により要求精度を高くして解くことが望まれる. なお, 要求相対精度は $\sqrt{\text{誤差判定のための単位}}$ までとするのが妥当である.

(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) この関数は, 適応型ニュートン・コーツ 9 点則に特異点処理能力を加えたアルゴリズムを 2 次元に拡張して利用している.

(7) **使用例**

(a) 問 題

$$\int_0^2 \int_0^2 (x+y)dxdy$$

を求める.

(b) 入力データ

被積分関数 $f(x,y)$ に対応する関数名: f

a=0.0, b=2.0, c=0.0, d=2.0, er=0.0, ea=0.0, idv=0

(c) 主プログラム

```
/*      C interface example for ASL_dhnrnm */

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

```
#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
double f(double *x,double *y)
#else
double f(x,y)
double *x;
double *y;
#endif
{
    return (*x)+(*y);
}
#ifdef __cplusplus
}
#endif

int main()
{
    double a;
    double b;
    double c;
    double d;
    double epsr;
    double epsa;
    int limi;
    double result;
    double abserr;
    int neval;
    int ierr;
    FILE *fp;

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

    printf( "    *** ASL_dhnrnm ***\n" );
    printf( "\n     ** Input **\n\n" );
    fscanf( fp, "%lf", &a );
    fscanf( fp, "%lf", &b );
    fscanf( fp, "%lf", &c );
    fscanf( fp, "%lf", &d );
    fscanf( fp, "%lf", &epsr );
    fscanf( fp, "%lf", &epsa );
    fscanf( fp, "%d", &limi );

    printf( "\ta     = %8.3g\n", a );
    printf( "\tb     = %8.3g\n", b );
    printf( "\tc     = %8.3g\n", c );
    printf( "\td     = %8.3g\n", d );
    printf( "\ters   = %8.3g\n", epsr );
    printf( "\teas   = %8.3g\n", epsa );
    printf( "\tidv   =   %6d\n", limi );

    fclose( fp );

    ierr = ASL_dhnrnm(f, a, b, c, d, epsr, epsa, limi, &result, &abserr, &neval);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tIntegral Approximation\n" );
    printf( "\t  q   = %8.3g\n", result );
    printf( "\n\tEstimate of Absolute Error\n" );
    printf( "\t  ae  = %8.3g\n", abserr );
    printf( "\n\tNumber of Function Evaluations\n" );
    printf( "\t  nev = %6d\n", neval );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dhnrnm ***

 ** Input **

a     =        0
b     =        2
c     =        0
d     =        2
ers   =        0
eas   =        0
idv   =        0

 ** Output **

ierr =      0

Integral Approximation
  q   =        8
```

```
Estimate of Absolute Error
  ae  = 1.78e-14

Number of Function Evaluations
  nev =    441
```

4.5.2 ASL_dhnfnm, ASL_rhnfnm 関数で示す領域の 2 次元積分

(1) **機 能**

X 軸方向の積分範囲を y に対する関数として与えて, 任意領域の 2 次元積分

$$\int_c^d \int_{a(y)}^{b(y)} f(x,y)dxdy$$

を自動的に行う.

(2) **使用法**

倍精度関数:

ierr = ASL_dhnfnm (f, a, b, c, d, er, ea, idv, & q, & ae, & nev);

単精度関数:

ierr = ASL_rhnfnm (f, a, b, c, d, er, ea, idv, & q, & ae, & nev);

(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	被積分関数 $f(x,y)$ を定義する関数名
2	a	{D / R}	–	入 力	X 軸方向積分下端を与える関数 $a(y)$ を定義する関数名
3	b	{D / R}	–	入 力	X 軸方向積分上端を与える関数 $b(y)$ を定義する関数名
4	c	{D / R}	1	入 力	Y 軸方向積分下端
5	d	{D / R}	1	入 力	Y 軸方向積分上端
6	er	{D / R}	1	入 力	要求相対精度 (既定値 : 誤差判定のための単位 $\times 64$) (4.1.1 参照)
7	ea	{D / R}	1	入 力	要求絶対精度 (既定値 : 絶対値最小値 $\times 2^{24}$) (4.1.1 参照)
8	idv	I	1	入 力	正常処理を行う細分区間数の最大値 (既定値:5000)
9	q	{D* / R*}	1	出 力	積分値
10	ae	{D* / R*}	1	出 力	絶対誤差推定値 (4.1.1 参照)
11	nev	I*	1	出 力	被積分関数評価回数
12	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) c<d

(b) er≥ 誤差判定のための単位 ×64 (既定値にするため, 0.0 を入力する場合は除く)

(c) ea≥ 絶対値最小値 $\times 2^{24}$ (既定値にするため, 0.0 を入力する場合は除く)

(d) idv>1(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.	積分値に −1 を乗じる or 積分値=0
1500	制限条件 (b), (c) または (d) を満足しなかった.	既定値にセットして処理を続ける.
2000	細分区間数が idv に達した.	徐々に最小分割幅を広げ近似解を得るようにする.
2400	ある細分区間の細分がこれ以上不可能となった.	要求された精度の解が得られないままで処理が終了する.
2500	解の精度が要求度に達しない.	
3500	結果の信用性がない (誤差が結果より大きい).	処理を打ち切る.
4000	一つの細分区間で 2 回以上オーバフローが発生した.	

(6) **注意事項**

(a) 非常に狭い範囲にピークがある場合は, 倍精度により要求精度を高くして解くことが望まれる. なお, 要求相対精度は $\sqrt{\text{誤差判定のための単位}}$ までとするのが妥当である.

(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) この関数は, 適応型ニュートン・コーツ 9 点則に特異点処理能力を加えたアルゴリズムを 2 次元に拡張して利用している.

(7) **使用例**

(a) 問 題

$$\int_0^2 \int_0^{\frac{\sqrt{4-y^2}}{2}} (x+y)dxdy$$

を求める.

(b) 入力データ

被積分関数 $f(x)$ に対応する関数名: f

X 軸方向積分下端ならびに上端を与える関数に対応する関数名: a, b

c=0.0, d=2.0, er=1.0e−8, ea=0.0, idv=0

(c) 主プログラム

```
/*      C interface example for ASL_dhnfnm */

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

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
double f(double *x,double *y)
#else
double f(x,y)
double *x;
double *y;
#endif
{
    return (*x)+(*y) ;
}
#ifdef __cplusplus
}
#endif

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
double a(double *y)
#else
double a(y)
double *y;
#endif
{
    return 0.0*(*y);
}
#ifdef __cplusplus
}
#endif

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
double b(double *y)
#else
double b(y)
double *y;
#endif
{
    return 0.5*sqrt(4.0-(*y)*(*y));
}
#ifdef __cplusplus
}
#endif


int main()
{
    double c;
    double d;
    double epsr;
    double epsa;
    int limi;
    double result;
    double abserr;
    int neval;
    int ierr;
    FILE *fp;

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

    printf( "    *** ASL_dhnfnm ***\n" );
    printf( "\n     ** Input **\n\n" );
    fscanf( fp, "%lf", &c );
    fscanf( fp, "%lf", &d );
    fscanf( fp, "%lf", &epsr );
    fscanf( fp, "%lf", &epsa );
    fscanf( fp, "%d", &limi );

    printf( "\tc     = %8.3g\n", c );
    printf( "\td     = %8.3g\n", d );
    printf( "\ter    = %8.3g\n", epsr );
    printf( "\tea    = %8.3g\n", epsa );
    printf( "\tidv   =   %6d\n", limi );

    fclose( fp );

    ierr = ASL_dhnfnm(f, a, b, c, d, epsr, epsa, limi, &result, &abserr, &neval);

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

```
    printf( "\n\tIntegral Approximation\n" );
    printf( "\t  q   = %8.3g\n", result );
    printf( "\n\tEstimate of Absolute Error\n" );
    printf( "\t  ae  = %8.3g\n", abserr );
    printf( "\n\tNumber of Function Evaluations\n" );
    printf( "\t  nev = %6d\n", neval );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dhnfnm ***

 ** Input **

c     =        0
d     =        2
er    =    1e-08
ea    =        0
idv   =        0

 ** Output **

ierr =      0

Integral Approximation
  q   =        2

Estimate of Absolute Error
  ae  =  1.4e-10

Number of Function Evaluations
  nev =   3990
```

4.6 多次元有限区間積分

4.6.1 ASL_dhnrml, ASL_rhnrml
超立方体領域の多次元積分

(1) **機 能**

2 次元以上の超立方体領域に対する多重積分をする (2 次元での特異性のあるときは, 2 次元用の関数を用いることが望まれる).

(2) **使用法**

倍精度関数:

ierr = ASL_dhnrml (f, a, b, m, er, ea, itmx, & q, & ae, & nev, iwk, wk);

単精度関数:

ierr = ASL_rhnrml (f, a, b, m, er, ea, itmx, & q, & ae, & 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　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	被積分関数 $f(x_1, \cdots, x_{\mathrm{m}})$ を定義する関数名
2	a	{D∗ / R∗}	m	入 力	x_i 軸方向積分下端, $i = 1, 2, \cdots, \mathrm{m}$
3	b	{D∗ / R∗}	m	入 力	x_i 軸方向積分上端, $i = 1, 2, \cdots, \mathrm{m}$
4	m	I	1	入 力	積分の多重度
5	er	{D / R}	1	入 力	要求相対精度 (既定値:誤差判定のための単位 ×64×m) (4.1.1 参照)
6	ea	{D / R}	1	入 力	要求絶対精度 (既定値:絶対値最小値 $\times 2^{24} \times$ m) (4.1.1 参照)
7	itmx	I	1	入 力	最大反復回数 (既定値:60/m)
8	q	{D∗ / R∗}	1	出 力	積分値
9	ae	{D∗ / R∗}	1	出 力	絶対誤差推定値 (4.1.1 参照)
10	nev	I*	1	出 力	被積分関数評価回数
11	iwk	I*	m	ワーク	作業領域
12	wk	{D∗ / R∗}	3×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) $2 \leq \mathrm{m} \leq 9$

(b) er≥ 誤差判定のための単位 $\times 64 \times \mathrm{m}$ (既定値にするため, 0.0 を入力する場合は除く)

(c) ea≥ 絶対値最小値 $\times 2^{24} \times \mathrm{m}$ (既定値にするため, 0.0 を入力する場合は除く)

(d) $4 \leq \mathrm{itmx} \leq 30$(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	$a_i > b_i$, $a_i = b_i$	$\int_{a_i}^{b_i} f dx_i = -\int_{b_i}^{a_i} f dx_i$ として積分する. または, 積分値 0.
1500	制限条件 (b), (c) または (d) を満足しなかった.	既定値にセットして処理を続ける.
2500	解の精度が要求精度に達せずに反復を終わった.	要求された精度の解が得られないままで, 処理が終了する. (反復終了時点で最も精度が良い解が返される)
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3500	結果の信用性がない (誤差が結果より大きい).	得られた結果を返す.

(6) **注意事項**

(a) 関数 f の作り方は次のようにする.

```
double FORTRAN f(double * x, int * m)
{
        return(被積分関数 f(x[0], ···, x[*m-1]));
}
```

(b) 要求相対精度は積分範囲内に特異点 (微分値が不連続であったり ∞ であったり微分不可能となる点) がある場合は $\sqrt[m]{10^{-6}}$ 程度, 積分境界上に著しい特異点 (関数値が ∞ になる点など) があるときは $\max(\sqrt[m]{10^{-12}}, 10^{-4} \times \mathrm{m})$ 程度, 上記以外は $\sqrt{\text{誤差判定のための単位} \times \mathrm{m}^2/20}$ 程度とするのが妥当である.

(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) この関数はガウス・ロンバーグ n 点則の n を各次元方向に増しながら得られる解の数列を, $\theta-$ アルゴリズムの改良の方法で加速して積分値を得る.

(7) **使用例**

(a) 問 題

$$\int_0^1 \int_0^1 \int_0^1 1/(3 - \cos(\pi x) - \cos(\pi y) - \cos(\pi z)) dx dy dz$$

を求める.

(b) 入力データ

被積分関数 $f(x_1, \cdots, x_m)$ に対応する関数名: f

a [0] =a [1] =a [2] =0.0, b [0] =b [1] =b [2] =1.0, m=3, er=1.0e−4, ea=0.0, itmx=15

(c) 主プログラム

```
/*      C interface example for ASL_dhnrml */

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
double f(double *x,int *m)
#else
double f(x,m)
double *x;
int   *m;
#endif
{
    return 1.0/(3.0-cos(M_PI*x[0])-cos(M_PI*x[1])-cos(M_PI*x[2]));
}
#ifdef __cplusplus
}
#endif

int main()
{
    double *a;
    double *b;
    int m;
    double epsr;
    double epsa;
    int lim;
    double result;
    double abserr;
    int neval;
    int *iwk;
    double *wk;
    int ierr;
    int i;
    FILE *fp;

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

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

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

    a = ( double * )malloc((size_t)( sizeof(double) * m ));
    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;
    }

    wk = ( double * )malloc((size_t)( sizeof(double) * (3*m) ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

    iwk = ( int * )malloc((size_t)( sizeof(int) * m ));
    if( iwk == NULL )
    {
        printf( "no enough memory for array iwk\n" );
        return -1;
    }
    for( i=0 ; i<m ; i++ )
    {
        fscanf( fp, "%lf", &a[i] );
        printf( "\ta[%6d]=%8.3g\n", i,a[i] );
    }
    for( i=0 ; i<m ; i++ )
    {
        fscanf( fp, "%lf", &b[i] );
        printf( "\tb[%6d]=%8.3g\n", i,b[i] );
```

```
    }
    fscanf( fp, "%lf", &epsr );
    fscanf( fp, "%lf", &epsa );
    fscanf( fp, "%d", &lim );

    printf( "\tm        =  %6d\n", m );
    printf( "\ter       =%8.3g\n", epsr );
    printf( "\tea       =%8.3g\n", epsa );
    printf( "\titmx     =  %6d\n", lim );

    fclose( fp );

    ierr = ASL_dhnrml(f, a, b, m, epsr, epsa, lim, &result, &abserr, &neval, iwk, wk);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tIntegral Approximation\n" );
    printf( "\t  q   = %8.3g\n", result );
    printf( "\n\tEstimate of Absolute Error\n" );
    printf( "\t  ae  = %8.3g\n", abserr );

    free( a );
    free( b );
    free( wk );
    free( iwk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dhnrml ***

 ** Input **

a[     0]=       0
a[     1]=       0
a[     2]=       0
b[     0]=       1
b[     1]=       1
b[     2]=       1
m        =       3
er       =  0.0001
ea       =       0
itmx     =      15

 ** Output **

ierr =      0

Integral Approximation
  q   =    0.505

Estimate of Absolute Error
  ae  = 1.24e-05
```

4.6.2 ASL_dhnfml, ASL_rhnfml
関数で示す領域の多次元積分

(1) **機 能**

2 次元以上の関数で示す領域に対する多重積分

$$\int_{a_m}^{b_m}\int_{a_{m-1}}^{b_{m-1}}\cdots\int_{a_1}^{b_1} f(x_1,\cdots,x_m)dx_1\cdots dx_m$$

を計算する. ただし,

$$\begin{aligned} a_i &= f_i(x_{i+1},\cdots,x_m),\ b_i = g_i(x_{i+1},\cdots,x_m);\ i=1,2,\cdots,m-1 \\ a_m &= f_m,\ b_m = g_m \end{aligned}$$

(2 次元での特異性のあるときは, 2 次元用の関数を用いることが望まれる).

(2) **使用法**

倍精度関数:

ierr = ASL_dhnfml (f, r, m, er, ea, itmx, & q, & ae, & nev, iwk, wk);

単精度関数:

ierr = ASL_rhnfml (f, r, m, er, ea, itmx, & q, & ae, & 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　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	被積分関数 $f(x_1,\cdots,x_m)$ を定義する関数名
2	r	−	−	入 力	積分の下端 a_i および上端 b_i $(i=1,\cdots,m)$ を定義する関数名
3	m	I	1	入 力	積分の多重度 m
4	er	{D / R}	1	入 力	要求相対精度 (既定値:誤差判定のための単位 $\times 64\times$m) (4.1.1 参照)
5	ea	{D / R}	1	入 力	要求絶対精度 (既定値:絶対値最小値 $\times 2^{24}\times$ m) (4.1.1 参照)
6	itmx	I	1	入 力	最大反復回数 (既定値:60/m)
7	q	{D∗ / R∗}	1	出 力	積分値
8	ae	{D∗ / R∗}	1	出 力	絶対誤差推定値 (4.1.1 参照)
9	nev	I*	1	出 力	被積分関数評価回数
10	iwk	I*	m	ワーク	作業領域

項番	引数と戻り値	型	大きさ	入出力	内　　容
11	wk	$\left\{\begin{matrix}\mathrm{D}* \\ \mathrm{R}*\end{matrix}\right\}$	3×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) $2 \leq \mathrm{m} \leq 9$

(b) er $\geq$ 誤差判定のための単位 $\times 64 \times$m (既定値にするため, 0.0 を入力する場合は除く)

(c) ea$\geq$ 絶対値最小値 $\times 2^{24} \times$m (既定値にするため, 0.0 を入力する場合は除く)

(d) $4 \leq$ itmx ≤ 30(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	$a_i > b_i$, $a_i = b_i$	$\int_{a_i}^{b_i} f dx_i = -\int_{b_i}^{a_i} f dx_i$ として積分する. または, 積分値 0.
1500	制限条件 (b), (c) または (d) を満足しなかった.	既定値にセットして処理を続ける.
2500	解の精度が要求精度に達せずに反復を終わった.	要求された精度の解が得られないままで, 処理が終了する.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3500	結果の信用性がない (誤差が結果より大きい).	得られた結果を返す.

(6) **注意事項**

(a) 関数 f の作り方は次のようにする.

```
double FORTRAN f(double *x, int *m)
{
        return (被積分関数 f(x[0], ..., x[*m - 1]));
}
```

(b) 関数 r の作り方は次のようにする.

```
void FORTRAN r(int *i, double *x, double *a, double *b, int *m)
{
        if(*i==1)
        {
                a[0]=f1(x[1], ..., x[m - 1]) に対応する式;
                b[0]=g1(x[1], ..., x[m - 1]) に対応する式;
        }
        else if(*i==2)
        {
                a[1]=f2(x[2], ..., x[m - 1]) に対応する式;
                b[1]=g2(x[2], ..., x[m - 1]) に対応する式;
```

}
⋮
⋮
else if(∗i==(∗m−1))
{
a[∗m−2] = f_{*m-2}(x[m − 1]) に対応する式;
b[∗m−2] = g_{*m-2}(x[m − 1]) に対応する式;
}
else if(∗i==(∗m))
{
a[∗m−1] = f_m;
b[∗m−1] = g_m;
}
}

(c) 要求相対精度は, 積分範囲内に特異点 (微分値が不連続であったり ∞ であったり微分不可能となる点) がある場合は $\sqrt[m]{10^{-6}}$ 程度, 積分境界上に著しい特異点 (関数値が ∞ になる点など) があるときは $\max(\sqrt[m]{10^{-12}}, 10^{-4} \times m)$ 程度, 上記以外は $\sqrt{\text{誤差判定のための単位}} \times m^2/20$ 程度とするのが妥当である.

(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) この関数はガウス・ロンバーグ N 点則の N を各次元方向に増しながら得られる解の数列を, $\theta-$ アルゴリズムの改良の方法で加速して積分近似値を得る.

(7) 使用例

(a) 問 題

$$\int_0^1 \int_0^{\sqrt{1-z^2}} \int_0^{\sqrt{1-y^2-z^2}} \sqrt{1-x^2-y^2-z^2}\,dxdydz$$

を求める.

(b) 入力データ
被積分関数 $f(x_1, \cdots, x_m)$ に対応する関数名: f
積分の下端および上端を与える関数名:r
m=3, er=1.0e−8, ea=0.0, itmx=15

(c) 主プログラム

```
/*      C interface example for ASL_dhnfml */

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
double f(double *x,int *m)
#else
double f(x,m)
double *x;
int  *m;
#endif
{
    return sqrt(1.0-x[0]*x[0]-x[1]*x[1]-x[2]*x[2]);
}
#ifdef __cplusplus
}
#endif
```

```
#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
void    r(int *i,double *x,double *a,double *b,int *m)
#else
void    r(i,x,a,b,m)
double *x,*a,*b;
int   *i,*m;
#endif
{
    if( *i == 1 )
    {
            a[0]=0.0;
            b[0]=sqrt(1.0-x[1]*x[1]-x[2]*x[2]);
    }
    else if( *i == 2 )
    {
            a[1]=0.0;
            b[1]=sqrt(1.0-x[2]*x[2]);
    }
    else
    {
            a[2]=0.0;
            b[2]=1.0;
    }
}
#ifdef __cplusplus
}
#endif

int main()
{
    int m;
    double epsr;
    double epsa;
    int lim;
    double result;
    double abserr;
    int neval;
    int *iwk;
    double *wk;
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%d", &m );
    fscanf( fp, "%lf", &epsr );
    fscanf( fp, "%lf", &epsa );
    fscanf( fp, "%d", &lim );

    printf( "\tm   =   %6d\n", m );
    printf( "\ter  = %8.3g\n", epsr );
    printf( "\tea  = %8.3g\n", epsa );
    printf( "\titmx=   %6d\n", lim );

    fclose( fp );

    wk = ( double * )malloc((size_t)( sizeof(double) * (3*m) ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

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

    ierr = ASL_dhnfml(f, r, m, epsr, epsa, lim, &result, &abserr, &neval, iwk, wk);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tIntegral Approximation\n" );
    printf( "\t  q   = %8.3g\n", result );
    printf( "\n\tEstimate of Absolute Error\n" );
    printf( "\t  ae  = %8.3g\n", abserr );

    free( wk );
    free( iwk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dhnfml ***

 ** Input **

m    =         3
er   =     1e-08
ea   =         0
itmx=        15

 ** Output **

ierr =       0

Integral Approximation
  q    =     0.308

Estimate of Absolute Error
  ae   = 1.64e-09
```

第 5 章　近似・補間

5.1　概 要

本章では, 与えられたデータ点に対して関数の当てはめを行う関数と与えられた関数に対して多項式近似を行う関数について説明する.

最初に, 最小二乗近似に関連しては, 利用者が与えたデータ点を最小二乗の意味で近似する関数の最適係数を求めるための以下の関数が用意されている.

(1) 最小二乗近似直交多項式

(2) 最小二乗近似非線形関数

(3) 2 次元任意データ最小二乗近似多項式

(4) 2 次元格子データ最小二乗近似多項式

最小二乗近似直交多項式の関数では, n 個のデータ点 x_i $(i = 1, 2, \cdots, n)$ における関数値 y_i $(i = 1, 2, \cdots, n)$ が与えられた場合に $x = x_i$ での x についての直交多項式の値と関数値 y_i との差の二乗和が極小になるよう直交多項式の係数を決定する. また, 自動次数最小二乗近似直交多項式関数では最適な近似多項式の次数も求める.

最小二乗近似非線形関数の関数では, n 個のデータ点 x_i $(i = 1, 2, \cdots, n)$ における関数値 y_i $(i = 1, 2, \cdots, n)$ が与えられた場合に $x = x_i$ での利用者定義関数の値と関数値 y_i との差の二乗和が極小になるように利用者定義関数を決定する.

2 次元任意データ最小二乗近似多項式の関数では, n 個の 2 次元座標点 (x_i, y_i) $(i = 1, 2, \cdots, n)$ と, その点における関数値 z_i が与えられた場合に $(x, y) = (x_i, y_i)$ での x と y についての多項式の係数を決定する.

2 次元格子データ最小二乗近似多項式の関数では, $nx \times ny$ 個の 2 次元格子点 (x_i, y_i) $(i = 1, 2, \cdots, nx;\ j = 1, 2, \cdots, ny)$ 上のすべての関数値 z_{ij} が与えられた場合に $(x, y) = (x_i, y_i)$ での x と y についての多項式の値と関数値 z_{ij} との差の二乗和を極小するように x と y についての多項式の係数を決定する.

補間に関連しては, 利用者が与えたデータ点についての補間値または補間多項式の係数を求めるために以下の関数が用意されている.

(1) 不等間隔離散点補間値

(2) 不等間隔離散点補間値, 補間係数

(3) 2 次元断面線上離散点補間値

(4) 2 次元格子上離散点補間値

不等間隔離散点補間値の関数では, エイトケン法を用いて n 個の与えられたデータ点 (x_i, y_i) $(i = 1, 2, \cdots, n)$ を与えられた補間点 x について補間し, 補間点での y 座標値を求める.

不等間隔離散点補間値, 補間係数の関数では, ニュートン法を用いて n 個の与えられたデータ点 (x_i, y_i) $(i = 1, 2, \cdots, n)$ を与えられた補間点 x について補間し, 補間点での y 座標値および, 補間多項式の係数を求める.

2 次元断面線上離散点補間値の関数では, xy 平面上に y 軸に平行な nx 個の直線 (ここでは断面線とよぶ) $x = x_i$ $(i = 1, 2, \cdots, nx)$ を設定し, おのおのの断面上に ny_i 個のデータ点とその点での関数値を与え, 任意の点での補間値を 3 次元スプライン関数によって求める. また, 与えられた断面線上のデータを補間するスプライン係数も求める.

2 次元格子上離散点補間値の関数では, 2 次元格子点 (x_i, y_i) $(i = 1, 2, \cdots, nx;\ j = 1, 2, \cdots, ny)$ 上のすべての関数値 z_{ij} が与えられた場合に一点 (x, y) 上での関数の補間値を求める.

チェビシェフ近似に関連しては, 利用者が与えた関数を最良近似の意味で近似する関数のチェビシェフ係数を求めるための以下の関数が用意されている.

(1) チェビシェフ近似

チェビシェフ近似の関数では, 有限区間 $[a, b]$ において関数 $f(x)$ が与えられた場合に, $x = x_i$ $(i = 0, 2, \cdots, n)$ での x についてのチェビシェフ多項式の値と関数値 y_i との差が極小になるようにチェビシェフ多項式の係数を決定する. 続いて, 求められたチェビシェフ係数 c_k $(k = 0, 1, \cdots, m)$ により T_k が チェビシェフ多項式であるとき,

$$\sum_{k=0}^{m}(d_k y^k) = \sum_{k=0}^{m}(c_k T_k(y))$$

となるような多項式の係数 d_k $(k = 0, 1, \cdots, m)$ を求める. また, この関数では最適な近似多項式の次数も求める.

5.1.1 使用上の注意

(1) 最小二乗近似非線形関数に対しては, 係数の初期値は, できるかぎり最適の係数に近いところにとるのが望ましい.

(2) 最小二乗近似非線形関数, 不等間隔離散点補間値, 2 次元格子上離散点補間値に対しては, 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) チェビシェフ近似に対しては, 要求最大誤差は, (チェビシェフ係数の打切り誤差)$\times 10^2$ 程度にとるのが適当である. (“打切り” については 5.6.1 参照)

5.1.2 使用しているアルゴリズム

5.1.2.1 最小二乗近似直交多項式

データ点 (x_i, y_i), 重み関数値 $w(x_i)$ $(i = 1, 2, \cdots, n)$ が与えられたとき y_i を m 次の多項式

$$f(x) = a_0 x^m + a_1 x^{m-1} + \cdots + a_{m-1} x + a_m \tag{5.1}$$

で近似する.
このとき, (x_i, y_i) が原点付近に分布しないときは誤差を少なくするため, x_i の分布の中点を 0.0 にし, y_i は最小値が 0.0 になるよう座標変換して計算する.
いま, 直交多項式 $P_j(x)$ $(j = 0, 1, \cdots, m)$ を

$$\sum_{i=1}^{n} w(x_i) P_u(x_i) P_v(x_i) = \sum_{i=1}^{n} w(x_i) \left\{P_u(x_i)\right\}^2 \delta_{uv} \tag{5.2}$$

を満たす多項式と定義し, $f(x)$ を直交多項式 $P_j(x)$ $(j = 0, 1, \cdots, m)$ の一次結合

$$f(x) = \sum_{j=0}^{m} b_j P_j(x) \tag{5.3}$$

で表わす.
ただし, δ_{uv} はクロネッカーのデルタで

$$\delta_{uv} = \begin{cases} 1 & (u = v) \\ 0 & (u \neq v) \end{cases}$$

(1) b_j, $P_j(x_i)$ $(i = 1, 2, \cdots, n;\ j = 0, 1, \cdots, m)$ の決定
係数 b_j を最小二乗法によって決定する. すなわち

$$H(b_0, \cdots, b_m) \equiv \sum_{i=1}^{n} w(x_i) \left\{y_i - f(x_i)\right\}^2$$

が最小になるよう, つまり

$$\frac{\partial H}{\partial b_k} = 0 \tag{5.4}$$

となるよう決められる.
(5.4) 式に (5.2) 式の直交条件を用いることによって, 係数 b_j は

$$b_j = \frac{\displaystyle\sum_{i=1}^{n} w(x_i) y_i P_j(x_i)}{\displaystyle\sum_{i=1}^{n} w(x_i) \{P_j(x_i)\}^2} \ (j = 0, 1, \cdots, m) \tag{5.5}$$

となる.
直交多項式 $P_j(x)$ は, x についての j 次の多項式として, 次の漸化式によって構成できる.

$$\begin{aligned} P_{-1}(x) &= 0 \\ P_0(x) &= 1 \\ P_{j+1}(x) &= (x - \alpha_{j+1}) P_j(x) - \beta_j P_{j-1}(x) \ (j = 0, \cdots, m-1) \end{aligned} \tag{5.6}$$

ここで, 係数 α_{j+1}, β_j は (5.2) 式で与えられる直交条件より

$$\begin{aligned} \alpha_{j+1} &= \frac{\sum_{i=1}^{n} w(x_i)x_i\{P_j(x_i)\}^2}{\sum_{i=1}^{n} w(x_i)\{P_j(x_i)\}^2} \\ \beta_j &= \frac{\sum_{i=1}^{n} w(x_i)\{P_j(x_i)\}^2}{\sum_{i=1}^{n} w(x_i)\{P_{j-1}(x_i)\}^2} \end{aligned} \tag{5.7}$$

と表せる.
(5.6), (5.7) 式より

$$P_{-1}(x_i), P_0(x_i) \to \alpha_1, \beta_0 \to P_1(x_i) \to \alpha_2, \beta_1 \to \cdots$$

と順次求まる.
これと (5.5) 式より b_j $(j = 0, 1, \cdots, m)$ が求まる.

(2) 係数 a_k $(k = 0, 1, \cdots, m)$ の決定
$P_j(x)$ を

$$P_j(x) = \sum_{k=0}^{j} c_{j,k}x^k = c_{j,j}x^j + c_{j,j-1}x^{j-1} + \cdots + c_{j,1}x + c_{j,0}$$

と表すと,

$$c_{j+1,k+1} = c_{j,k} - \alpha_{j+1}c_{j,k+1} - \beta_j c_{j-1,k+1} \ (j = 0, 1, \cdots, m-1;\ k = 0, 1, \cdots, j)$$

(ただし, $c_{0,0} = 1$, $c_{-1,0} = 0$)
と書ける. これより $c_{j,k}$ が決まる. また,

$$f(x) = \sum_{j=0}^{m} b_j \sum_{k=0}^{j} c_{j,k}x^k = \sum_{k=0}^{m} \left(\sum_{j=k}^{m} b_j c_{j,k} \right) x^k$$

と書けるから, 求められた b_j と $c_{j,k}$ から係数 a_k は

$$a_{m-k+1} = \sum_{j=k}^{m} b_j c_{j,k}$$

と求められる.

5.1.2.2 最小二乗近似非線形関数

n 個の座標値 (x_i, y_i) $(i = 1, \cdots, n)$ と m 個のパラメータ $\boldsymbol{a} = \{a_i\}$ $(i = 1, \cdots, m)$ をもつ近似関数 $f(x, \boldsymbol{a})$ が与えられたとき, 残差二乗和

$$S(\boldsymbol{a}) = \sum_{i=1}^{n} (y_i - f(x_i, \boldsymbol{a}))^2$$

を極小にする $\boldsymbol{a}$ を求める. ここでベクトル関数 $\boldsymbol{h}(\boldsymbol{a}) = \{h_i(\boldsymbol{a})\}$ を以下のように定義する.

$$h_i(\boldsymbol{a}) = y_i - f(x_i, \boldsymbol{a}) \ (i = 1, \cdots, n)$$

このとき, $S(\boldsymbol{a})$ は

$$S(\boldsymbol{a}) = \|\boldsymbol{h}(\boldsymbol{a})\|_2^2$$

と書ける. ただし, $\|\boldsymbol{a}\|_2 = \sqrt{\boldsymbol{a}^T\boldsymbol{a}}$ である (^{T} は転置を意味する).
$S(\boldsymbol{a})$ を極小にする解を求めるために, パウエルのハイブリッド法を用いて $\boldsymbol{a} = \boldsymbol{a}_0$ から探索を開始し, $\boldsymbol{a} = \boldsymbol{a}_1, \boldsymbol{a}_2, \cdots$ と解を逐次修正していく.
ハイブリッド法では, 非線形関数を係数 $\boldsymbol{a}$について線形化して求めたガウス・ニュートン法と最急降下法による修正ベクトル線形結合として各ステップの修正ベクトルを決定する. 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, 各ステップでのヤコビ行列の値は直接求めずに, 前のステップの値と関数情報から決定する.

(1) 修正ベクトル $\Delta\boldsymbol{a}$ の計算
ベクトル関数 $\boldsymbol{h}(\boldsymbol{a} + \Delta\boldsymbol{a})$ を $\Delta\boldsymbol{a}$ について線形の範囲で近似して

$$\boldsymbol{h}_L(\boldsymbol{a} + \Delta\boldsymbol{a}) = \boldsymbol{h}(\boldsymbol{a}) + A\Delta\boldsymbol{a}$$

とおき

$$S_L(\boldsymbol{a} + \Delta\boldsymbol{a}) = \|\boldsymbol{h}(\boldsymbol{a}) + A\Delta\boldsymbol{a}\|_2^2$$

を極小にする問題を考える. ここで A は $\boldsymbol{h}$ のヤコビ行列 $\dfrac{\partial \boldsymbol{h}}{\partial \boldsymbol{a}}$ である.
このとき最急降下法による各ステップの修正ベクトル $\Delta\boldsymbol{a}_S$ は $\boldsymbol{b} = -A^T\boldsymbol{h}(\boldsymbol{a})$ として,

$$\Delta\boldsymbol{a}_S = \frac{\|\boldsymbol{b}\|_2^2}{\|A\boldsymbol{b}\|_2^2}\boldsymbol{b}$$

によって与えられ, 一方, ガウス・ニュートン法による各ステップの修正ベクトル $\Delta\boldsymbol{a}_G$ は正規方程式

$$A^TA\Delta\boldsymbol{a}_G = \boldsymbol{b}$$

を解いて得られる. 本ライブラリでは, これを解くために QR 分解法を用いる.
一般に,

$$\|\Delta\boldsymbol{a}_S\|_2 \leq \|\Delta\boldsymbol{a}_G\|_2$$

である.
各ステップの修正ベクトルは次に述べるステップサイズ d によって $\Delta\boldsymbol{a}_S$ と $\Delta\boldsymbol{a}_G$ の線形結合として以下のように決定する.

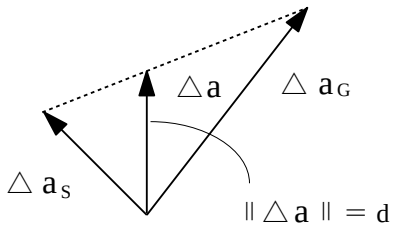
(a) $d \leq \|\Delta\boldsymbol{a}_S\|_2$ の場合

$$\Delta\boldsymbol{a} = d\frac{\Delta\boldsymbol{a}_S}{\|\Delta\boldsymbol{a}_S\|_2}$$

(b) $\|\Delta\boldsymbol{a}_S\|_2 < d < \|\Delta\boldsymbol{a}_G\|_2$ の場合 (図 5−1 参照)

$$\Delta\boldsymbol{a} = \alpha\Delta\boldsymbol{a}_S + \beta\Delta\boldsymbol{a}_G \ (\alpha > 0, \beta > 0, \|\Delta\boldsymbol{a}\|_2 = d)$$

図 5−1



(c) $\|\Delta \boldsymbol{a}_G\|_2 \leq d$ の場合

$$\Delta \boldsymbol{a} = \Delta \boldsymbol{a}_G$$

(2) ステップサイズ d の決定

ステップサイズは, 初期値 $d = \|\Delta \boldsymbol{a}_S\|_2$ 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近似での S の変化量

$$\Delta S_L = S_L(\boldsymbol{a} + \Delta \boldsymbol{a}) - S_L(\boldsymbol{a})$$

と実際の変化量

$$\Delta S = S(\boldsymbol{a} + \Delta \boldsymbol{a}) - S(\boldsymbol{a})$$

との比 $r = \dfrac{\Delta S}{\Delta S_L}$ を用いる.

(a) $r < 0.1$ のとき非線決性が強いと判断し, 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.

$$\lambda^2 = 1.0 - (r - 0.1)\frac{\Delta S_L}{(S_P + \sqrt{(S_P^2 - S_S(r - 0.1)\Delta S_L)}}$$

ただし,

$$\delta \boldsymbol{h} = \boldsymbol{h}(\boldsymbol{a} + \Delta \boldsymbol{a}) - (\boldsymbol{h}(\boldsymbol{a}) + A\Delta \boldsymbol{a})$$

として

$$\begin{aligned} S_P &= \sum_{i=1}^{n} |h_i(\boldsymbol{a} + \Delta \boldsymbol{a})\delta h_i| \\ S_S &= \|\delta \boldsymbol{h}\|_2^2 \end{aligned}$$

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.

$$\begin{aligned} \mu &= \min(2, \lambda, \tau) \\ \tau &= \frac{\lambda}{\mu} \end{aligned}$$

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. ハイブリッド法でベクトル $\boldsymbol{p}$ 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 $2m$ 個のヤコビ行列の修正で用いられた修正ベクトルのうち互いに独立な m 個のベクトルを直交化して, $\Omega = (\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \cdots, \boldsymbol{\omega}_m)$ に保持しておく, 大きさ m の配列$\boldsymbol{j}$を用いて, $\boldsymbol{\omega}_i$ 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, $\boldsymbol{\omega}_i$ 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 $j_i = m - i + 1$ $(i = 1, \cdots, m)$ 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) $\Delta\boldsymbol{a} = \Delta\boldsymbol{a}_G$ の場合

独立性のいかんにかかわらず, $\Delta\boldsymbol{a}$ を修正ベクトルとして採用する.

(b) $\Delta\boldsymbol{a} = \Delta\boldsymbol{a}_G$ でない場合

$j_1 < 2m$ であるか, または, $\Delta\boldsymbol{a}$ が $(\boldsymbol{\omega}_2, \boldsymbol{\omega}_3, \cdots, \boldsymbol{\omega}_m)$ と独立なら, $\Delta\boldsymbol{a}$ 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.

$$A' = A + \delta\boldsymbol{h}\frac{\Delta\boldsymbol{a}^T}{\|\Delta\boldsymbol{a}\|_2^2}$$

ただし, $\|\Delta\boldsymbol{a}\|_2 < d_{min}$ であるか, または, $j_1 = 2m$ で $\Delta\boldsymbol{a}$ が $(\boldsymbol{\omega}_2, \boldsymbol{\omega}_3, \cdots, \boldsymbol{\omega}_m)$ と独立でないなら, $\Delta\boldsymbol{a} = d_{min}\boldsymbol{\omega}_1$ 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.

$\Delta\boldsymbol{a} = d_{min}\boldsymbol{\omega}_1$ としたときには,

$$\begin{aligned}
\boldsymbol{\omega}_i &= \boldsymbol{\omega}_{i+1} \qquad (i = 1, \cdots, m-1) \\
\boldsymbol{\omega}_m &= \boldsymbol{\omega}_1 \\
j_i &= j_{i+1} + 1 \quad (i = 1, \cdots, m-1) \\
j_m &= 1
\end{aligned}$$

とすればよい. そうでないときには, 以下のようにする.

$(\boldsymbol{\omega}_{k+1}, \boldsymbol{\omega}_{k+2}, \cdots, \boldsymbol{\omega}_m, \Delta\boldsymbol{a})$ が互いに独立になる最小の k を求める.

$(\boldsymbol{\omega}_1, \cdots, \boldsymbol{\omega}_{k-1}, \boldsymbol{\omega}_{k+1}, \boldsymbol{\omega}_{k+2}, \cdots, \boldsymbol{\omega}_m, \Delta\boldsymbol{a})$ を直交化して, それを改めて

$(\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \cdots, \boldsymbol{\omega}_m)$ とする.

$$\begin{aligned}
j_i &= j_i + 1 \qquad (i = 1, \cdots, k-1) \\
j_i &= j_{i+1} + 1 \quad (i = k, \cdots, m-1) \\
j_m &= 1
\end{aligned}$$

このようにして, 常に, $j_1 \leq 2m$ 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, $\boldsymbol{a}+\Delta\boldsymbol{a}$ を解とする.

$$\|\Delta\boldsymbol{a}\|_\infty \le e_r \max(1, \|\boldsymbol{a}+\Delta\boldsymbol{a}\|_\infty)$$

ここで, e_r は要求精度であり,

$$\|\boldsymbol{a}\|_\infty = \max_i |a_i|$$

である.

5.1.2.3 2 次元任意データ最小二乗近似多項式

空間上の与えられた離散点 (x_k, y_k, z_k) $(k=1,2,\cdots,n)$ に対する最小二乗近似多項式

$$f(x,y) = \sum_{j=1}^{m+1}\sum_{i=1}^{m+2-j} a_{i,j}x^{i-1}y^{j-1}$$

を求める.
ここで $a_{i,j}$ は多項式の係数であり, x, y は独立変数, m は x, y に関する多項式の最大次数である.
いま, z_k と $f(x_k, y_k)$ の残差平方和を χ^2 とすれば

$$\chi^2 = \sum_{k=1}^{n}\left(z_k - \sum_{j=1}^{m+1}\sum_{i=1}^{m+2-j} a_{i,j}x_k^{i-1}y_k^{j-1}\right)^2$$

で与えられる. $a_{i,j}$ は χ^2 を最小にする条件

$$\frac{\partial\chi^2}{\partial a_{i,j}} = 0$$

により定められる.
これにより次の正規方程式系を得る.

$$\sum_{jc=1}^{m+1}\sum_{ic=1}^{m+2-jc}\sum_{k=1}^{n} a_{ic,jc}x_k^{ic+ir-2}y_k^{jc+jr-2} = \sum_{k=1}^{n} x_k^{ir-1}y_k^{jr-1}z_k$$
$$(jr=1,\cdots,m+1;\ ir=1,\cdots,m+2-jr)$$

いま,

$$\begin{aligned}
G_k &= (g_{k,(jr,ir),(jc,ic)}) \\
&= (x_k^{ic+ir-2}y_k^{jc+jr-2}) && \left(\begin{array}{ll} jr=1,\cdots,m+1; & jc=1,\cdots,m+1 \\ ir=1,\cdots,m+2-jr; & ic=1,\cdots,m+2-jc \end{array}\right) \\
\boldsymbol{a} &= (a_{(jc,ic)}) && \left(\begin{array}{l} jc=1,\cdots,m+1 \\ ic=1,\cdots,m+2-jc \end{array}\right) \\
\boldsymbol{b}_k &= (b_{k,(jr,ir)}) = (x_k^{ir-1}y_k^{jr-1}z_k) && \left(\begin{array}{l} jr=1,\cdots,m+1 \\ ir=1,\cdots,m+2-jr \end{array}\right)
\end{aligned}$$

と定義し,

$$\begin{aligned}
G &= \sum_{k=1}^{n} G_k \\
\boldsymbol{b} &= \sum_{k=1}^{n} \boldsymbol{b}_k
\end{aligned}$$

とおけば正規方程式系は

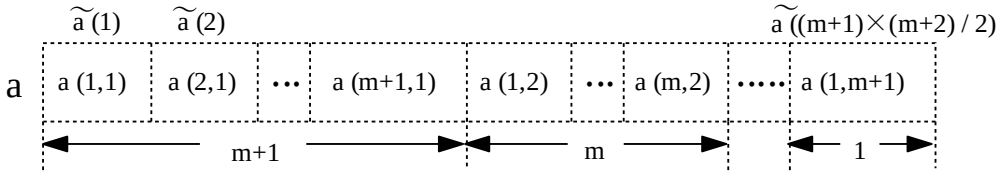
$$G\boldsymbol{a} = \boldsymbol{b}$$

と表される. ここで $\tilde{G}$ を $\{\frac{(m+1)(m+2)}{2}\}$ 次正方行列, $\tilde{\boldsymbol{a}}$, $\tilde{\boldsymbol{b}}$ を $\{\frac{(m+1)(m+2)}{2}\}$ 次列ベクトルに変換し $\tilde{G}\tilde{\boldsymbol{a}} = \tilde{\boldsymbol{b}}$ とし, これを $\tilde{\boldsymbol{a}}$ について解けば求める多項式係数が得られる.
なお, 多項式の係数 $a_{i,j}$ は計算の便宜上, 図 5−2 のように 1 次元配列に格納している.
添字の対応関係は,

$$a(i,j) = \tilde{a}(i + ((j-1) \times (2 \times m - j + 4))/2)$$

と表される.

図 5−2



5.1.2.4 2 次元格子データ最小二乗近似多項式

2 次元格子点 (x_i, y_j) $(i = 1, 2, \cdots, nx;\ j = 1, 2, \cdots, ny)$ 上のすべての Z 座標値 $Z_{i,j}$ が与えられた場合, この関数を近似する最小二乗近似多項式

$$f(x,y) = \sum_{i=1}^{ix+1} \sum_{j=1}^{iy+1} a_{i,j} x^{i-1} y^{j-1} \tag{5.8}$$

を求める. 係数 $a_{i,j}$ を決定するために $f(x,y)$ が以下に定義する直交多項式 $\Phi_{r-1}(x)$, $\Psi_{s-1}(y)$ の積の線形結合として

$$f(x,y) = \sum_{r=1}^{ix+1} \sum_{s=1}^{iy+1} \Gamma_{rs} \Phi_{r-1}(x) \Psi_{s-1}(y) \tag{5.9}$$

と表せると仮定する.
$\Phi_{r-1}(x)$, $\Psi_{s-1}(y)$ はそれぞれ x および y に関する $r-1$, $s-1$ 次の多項式であるから, $C_{r,k}$, $B_{s,l}$ を x および y に関する直交多項式の係数とすれば, $\Phi_{r-1}(x)$, $\Psi_{s-1}(y)$ は次式のように書くことができる.

$$\Phi_{r-1}(x) = \sum_{k=1}^{r} C_{r,k} x^{k-1} \tag{5.10}$$

$$\Psi_{s-1}(y) = \sum_{l=1}^{s} B_{s,l} y^{l-1} \tag{5.11}$$

(5.9) 式に (5.10) および (5.11) 式を代入して

$$\begin{aligned} f(x,y) &= \sum_{r=1}^{ix+1} \sum_{s=1}^{iy+1} \Gamma_{rs} \Phi_{r-1}(x) \Psi_{s-1}(y) \\ &= \sum_{r=1}^{ix+1} \sum_{s=1}^{iy+1} \Gamma_{rs} \sum_{k=1}^{r} C_{r,k} x^{k-1} \sum_{l=1}^{s} B_{s,l} y^{l-1} \\ &= \sum_{r=1}^{ix+1} \sum_{s=1}^{iy+1} \Gamma_{rs} \sum_{k=r}^{ix+1} C_{r,k} x^{r-1} \sum_{l=s}^{iy+1} B_{s,l} y^{l-1} \quad \text{(図 5−3 参照)} \end{aligned} \tag{5.12}$$

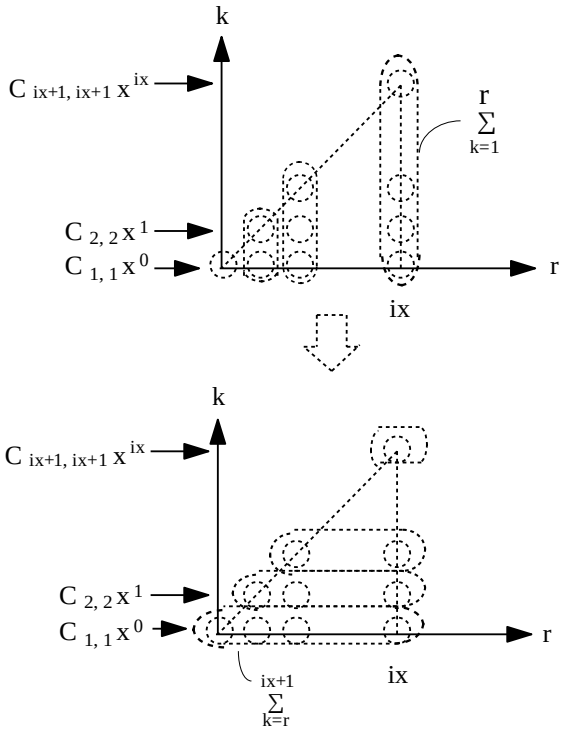
$$f(x,y) \quad = \quad \sum_{i=1}^{ix+1}\sum_{j=1}^{iy+1}\left(\Gamma_{ij}\sum_{k=1}^{ix+1}\sum_{l=j}^{iy+1}C_{i,k}B_{j,l}\right)x^{i-1}y^{j-1}$$

これを式 (5.8) を比べることによって係数 $a_{i,j}$ は

$$a_{i,j} = \Gamma_{ij}\sum_{k=1}^{ix+1}\sum_{l=j}^{iy+1}C_{i,k}B_{j,l}$$

と求まる.

図 5−3



(1) 直交多項式

x または y に関する直交多項式をここでは与えられた, データ点 x_i $(i = 1, 2, \cdots, nx)$ または y_i $(j = 1, 2, \cdots, ny)$ 上で

$$\sum_{i=1}^{nx}\Phi_r(x_i)\Phi_s(x_i) = D_r\delta_{rs} \tag{5.13}$$

$$\sum_{j=1}^{ny}\Psi_r(y_j)\Psi_s(y_j) = d_r\delta_{rs} \tag{5.14}$$

を満たす多項式と定義する.

ただし, δ_{rs} はクロネッカーのデルタで

$$\delta_{rs} = \begin{cases} 1 & (r = s) \\ 0 & (r \neq s) \end{cases}$$

前述の直交条件を満たす x または y の直交多項式 $\Phi_r(x)$, $\Psi_s(y)$ は次の漸化式から求めることができる. x の直交多項式

$$\Phi_r(x) = (x - \alpha_{r-1})\Phi_{r-1}(x) - \beta_{r-1}\Phi_{r-2}(x) \ (r = 2, 3, \cdots) \tag{5.15}$$

ただし,

$$\begin{aligned}
\alpha_{r-1} &= \sum_{i=1}^{nx} x_i \frac{\Phi_{r-1}^2(x_i)}{D_{r-1}} \\
\beta_{r-1} &= \frac{D_{r-1}}{D_{r-2}} \\
\Phi_0(x) &= 1 \\
\Phi_1(x) &= x - \bar{x} \\
\bar{x} &= \sum_{i=1}^{nx} \frac{x_i}{nx}
\end{aligned}$$

y の直交多項式

$$\Psi_s(y) = (y - \alpha'_{s-1})\Psi_{s-1}(y) - \beta'_{s-1}\Psi_{s-2}(y) \ (s = 2, 3, \cdots) \tag{5.16}$$

ただし,

$$\begin{aligned}
\alpha'_{s-1} &= \sum_{j=1}^{ny} y_j \frac{\Psi_{s-1}^2(y_j)}{d_{s-1}} \\
\beta'_{s-1} &= \frac{d_{s-1}}{d_{s-2}} \\
\Psi_0(y) &= 1 \\
\Psi_1(y) &= y - \bar{y} \\
\bar{y} &= \sum_{j=1}^{ny} \frac{y_j}{ny}
\end{aligned}$$

(2) Γ_{rs} の計算

まずはじめに $y = y_j$ と固定して x について最小二乗曲線を求める.

求める最小二乗曲線を $\lambda_{r,j}$ を係数として

$$q_j(x) \equiv f(x, y_j) = \sum_{r=1}^{ix+1} \lambda_{r,j}\Phi_{r-1}(x) \ (j = 1, 2, \cdots, ny) \tag{5.17}$$

と書き, 次式で与えられる格子点 $x = x_i$ 上での残差平方和

$$Q = \sum_{i=1}^{nx} \{q_j(x_i) - z_{i,j}\}^2 \tag{5.18}$$

を最小にする. すなわち, $\dfrac{\partial Q}{\partial \lambda_{r,j}} = 0$ とおき, (5.13) 式で与えられる直交条件を用いることによって次式を得る.

$$\lambda_{r,j} = \sum_{i=1}^{nx} z_{i,j} \frac{\Phi_{r-1}(x_i)}{D_{r-1}} \ (r = 1, 2, \cdots, ix + 1) \tag{5.19}$$

(5.9) 式, (5.17) 式を比較することによって

$$\lambda_{r,j} = \sum_{s=1}^{iy+1} \Gamma_{r,s} \Psi_{s-1}(y_j) \tag{5.20}$$

(5.20) の式の両辺に $\Psi_{s-1}(y_j)$ を掛けて $j = 1, \cdots, ny$ まで和をとり (5.14) 式の直交条件を用いることによって

$$\Gamma_{rs} = \sum_{j=1}^{ny} \lambda_{r,j} \frac{\Psi_{s-1}(y_j)}{d_{s-1}} \ (s = 1, 2, \cdots, iy + 1) \tag{5.21}$$

が得られる. (5.21) 式によって Γ_{rs} が決定される.

(3) $C_{r,k}$, $B_{s,l}$ の計算
(5.10) 式を (5.15) 式に代入してこれが任意の x について成立するように x のべきの係数を 0 とおくことによって次の $C_{r,k}$ に関する漸化式を得る.

$$C_{r,k} = C_{r-1,k-1} - \alpha_{r-1} C_{r-1,k} - \beta_{r-1} C_{r-2,k}$$

(ただし $r < k$ のときは $C_{r,k} = 0$ とおき $C_{0,0} = 1$ とする).
同様にして $B_{s,l}$ に関する漸化式は次式で与えられる.

$$B_{s,l} = B_{s-1,l-1} - \alpha'_{s-1} B_{s-1,l} - \beta'_{s-1} B_{s-2,l}$$

(ただし $s < l$ のときは $B_{s,l} = 0$ とおき $B_{0,0} = 1$ とする).
これらにより $C_{r,k}$, $B_{s,l}$ を計算できる.

5.1.2.5 不等間隔離散点補間値

データ点 (x_i, y_i) $(i = 1, \cdots, n)$, 補間点 $x = u$ が与えられたとき x_i を u に近い順に並べ換え, それに対応した y_i も同時に並べ換えて (u_i, v_i) とする.
2 点 (u_1, v_1) と (u_2, v_2) の間を補間し, 補間多項式を求めたいときは
基本線形補間公式

$$y(x) = \frac{v_1(x - u_2) - v_2(x - u_1)}{u_1 - u_2} \tag{5.22}$$

を用いれば, 1 次の補間多項式が求められる.
また, 3 点 $(u_1, v_1), (u_2, v_2), (u_3, v_3)$ の間を補間し, 補間多項式を求めたいときは, まず,
$(u_1, v_1), (u_2, v_2)$ の間で (5.22) を用いて線形補間を行い, 1 次の補間多項式

$$y_1^1(x) = \frac{v_1(x - u_2) - v_2(x - u_1)}{u_1 - u_2}$$

を求める.
次に, (u_2, v_2), (u_3, v_3) の間で (5.22) を用いて線形補間を行い, 1 次の補間多項式

$$y_1^2(x) = \frac{v_2(x - u_3) - v_3(x - u_2)}{u_2 - u_3}$$

を求める.
そして得られた $y_1^1(x)$, $y_1^2(x)$ の間で (5.22) を用いて線形補間を行えば,

$$y_2^1(x) = \frac{y_1^1(x)(x - u_3) - y_1^2(x)(x - u_1)}{u_1 - u_3}$$

という 2 次の補間多項式が求められる.
同様に (5.22) 式を繰り返し用いることによって, 3 次, 4 次, $\cdots$ の補間多項式をつくることができる.
一般に, j 次の補間多項式 (ただし, $j = 2, \cdots, n$) は,

$$y_1^k(x) = \frac{v_k(x - u_{k+1}) - v_{k+1}(x - u_k)}{u_k - u_{k+1}} \ (k = 1, \cdots, j) \tag{5.23}$$

$$y_m^k(x) = \frac{y_{m-1}^k(x)(x - u_{k+m}) - y_{m-1}^{k+1}(x)(x - u_k)}{u_k - u_{k+m}} \tag{5.24}$$
$$(m = 2, \cdots, j;\ k = 1, \cdots, j + 1 - m)$$

と表せる.
そこで, $x = u$ に対する n 次補間多項式の値 (補間値) は $x = u$ に対して (5.23) 式, (5.24) 式を計算することによって得られる.
次に, 補間多項式の次数の決定法について述べる.
まず, $x = u$ の近傍の j 個の点を用いて補間した補間値を Z_j $(= y_j^1(u))$ とする.
補間差 D_j を

$$D_j \equiv |Z_{j-1} - Z_j| \ (j = 2, \cdots, n)$$

で定義し $D_2, \cdots, D_n$ と計算していく過程で, 利用者が与えた要求絶対精度 ε に対して常に

$$D_j > \varepsilon \ (j = 2, \cdots, n)$$

であれば

$$D_l = \min_j(D_j)$$

となる l を補間多項式の次数とし, Z_l をそのときの補間値とする. また D_l を補間値の絶対誤差として出力する.
もし, ある数 j $(j \leq n)$ に対して

$$D_j \leq \varepsilon$$

となったならば, 上式を満たす最小の j を補間多項式の次数とし, Z_j をそのときの補間値とする. また D_j を補間値の絶対誤差として出力する.

5.1.2.6 不等間隔離散点補間値, 補間係数

データ点 (x_i, y_i) $(i = 1, \cdots, n)$, 補間点 $x = u$ が与えられたとき, x_i を u に近い順に並べ換え, それに対応した y_i も同時に並べ換えて (u_i, v_i) とする.
ここで m 次多項式を

$$f(x) = c_1 + c_2(x - u_1) + \cdots + c_{m+1}(x - u_1) \cdots (x - u_m) \ (1 \leq m \leq n - 1) \tag{5.25}$$

とおき, 係数 $c_1, \cdots, c_{m+1}$ を求める.
いま, 補間点 $x = u_1, \cdots, u_n$ と関数 $f(x)$ が与えられるとき $f(x)$ の差分商を

$$f[i] = f(u_i) \ (i = 1, \cdots, n)$$

$$f[i_1, \cdots, i_{k+1}] = \frac{f[i_2, \cdots, i_{k+1}] - f[i_1, \cdots, i_k]}{u_{i_{k+1}} - u_{i_1}} \ (k = 1, \cdots, n)$$

によって帰納的に定義する. $(i_1, \cdots, i_{k+1}$ は $n + 1$ 以下の相異なる正の整数)
(5.25) 式において $x = u_1$ とおくと

$$c_1 = f(u_1) = f[1]$$

となる.
これを (5.25) 式に代入し整理すると

$$\frac{f(x)-f[1]}{x-u_1} = c_2 + c_3(x-u_2) + \cdots + c_{m+1}(x-u_2)\cdots(x-u_m)$$

となる. $x = u_2$ とおくと

$$c_2 = \frac{f(u_2)-f[1]}{u_2-u_1} = \frac{f[2]-f[1]}{u_2-u_1} = f[1,2]$$

同様に上の操作を繰り返すと, 係数 c_i は

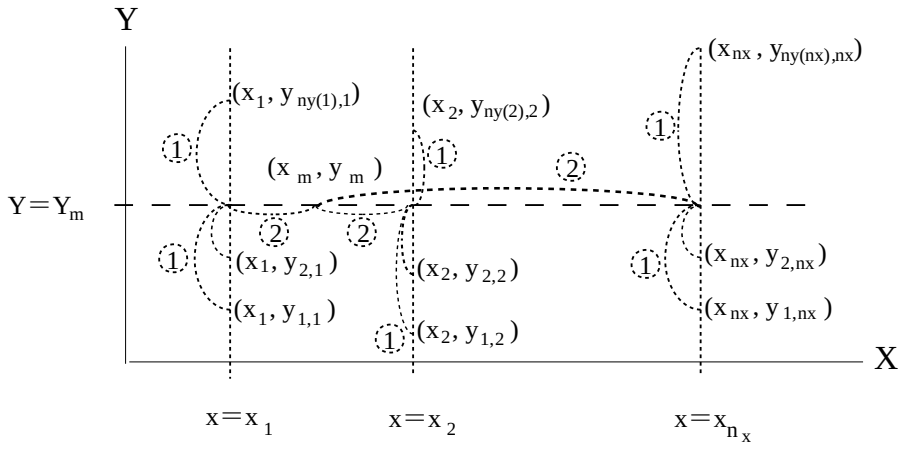
$$c_i = f[1,\cdots,i] \ (i = 1,\cdots,m+1)$$

と表すことができる. したがって, 差分商を逐次計算することによって c_i $(i = 1,2,\cdots,m+1)$ が求まり, これと (5.25) 式より与えられた補間点における補間値を計算することができる.

5.1.2.7　2 次元断面線上離散点補間値

xy 平面上で X 軸に垂直な nx 個の直線 $x = x_i$ $(i = 1,2,\cdots,nx)$ (ここではこの直線を断面線と呼ぶ) を考え各直線上の ny_i $(i = 1,\cdots,nx)$ 個の点 $(x_i, y_{j,i})$ $(i = 1,2,\cdots,nx;\ j = 1,2,\cdots,ny_i)$ と各点での Z 座標値 $z_{i,j}$ $(i = 1,\cdots,nx;\ j = 1,\cdots,ny_i)$ が与えられたとき, 点 (x_l, y_l) $(x_1 \le x_l \le x_{nx};\ \min(y_{j,i}) \le y_l \le \max(y_{j,i}))$ での補間値を次のようにして求める. (図 5−4 参照)

図 5−4



① ： step1で計算

② ： step2で計算

(Step 1)

まず, 各断面上にて, データ点の Y 座標値を横座標値, 関数値を縦座標値とみなして 3 次スプライン係数を求める. (6.1.2 参照) 続いて, 今求めたスプライン係数を用いて各断面線と直線 $y = y_l$ との交点における補間値を求める. (6.1.2 参照) この場合 y_l の値によっては補外値となることもある.

(Step 2)

次に全く同様にして断面線の X 座標値を横座標値, 今求めた y_l における各断面線上の補間値を縦座標値とみなして, スプライン係数を求め, さらに $x = x_l$ における補間値を求める. 以上によって, 点 (x_l, y_l) での補間値が決定できる.

5.1.2.8　2 次元格子離散点補間値

2 次元格子点 (x_i, y_j) $(i = 1, \cdots, nx;\ j = 1, \cdots, ny)$ 上のすべての Z 座標値 $z_{i,j}$ が与えられたとき, 格子内の任意の点 (x_l, y_l) に対する補間値をエイトケン法により求める (5.1.2 参照).
まず, X 軸に平行な各線上において X 座標値を横座標値, 関数値を縦座標値とみなして, x_l における補間値を求める. 次に Y 座標値を横座標値, 今求めた x_l における各補間値を縦座標値とみなして y_l における補間値を求める.

5.1.2.9　チェビシェフ近似

閉区間 $[a, b]$ 上で連続な関数 $f(x)$ のノルムを

$$\|f\| = \max_{x \in [a,b]} |f(x)|$$

によって定義する. これを最大値ノルムまたは一様ノルムという. n 次多項式 $P_n(x) = a_0 + a_1 x + \cdots + a_n x^n$ (または分子, 分母がそれぞれ m, n 次の多項式である有理式) を関数 $f(x)$ の近似式とみて, その近似度 $\|f - P_n\|$ を最小化する多項式 (有理式) をチェビシェフ近似, あるいは単に最良近似という.

さて最良近似は公式などとして近似式を表すには良いが, 近似式を求める手順が複雑なことや項の継ぎ足しをするときには新たに計算し直す必要があり, 近似式の作成には技巧と多大な労力が必要となる. 最良近似で使われる手法はいろいろあるが, 比較的容易に計算できるのが, 次に述べるチェビシェフの展開式を使った手法である.

最良の近似式であるための多項式の条件は, 誤差関数の極大値が等しくかつその符号が正負交互に表れることである. このような条件を満たす多項式として, チェビシェフ多項式がある. 以下で, 順次チェビシェフ多項式の求め方について述べる.

(1) チェビシェフ係数を求める

n 次のチェビシェフ多項式は, $T_n(x)$ と書き表し, 次の陽関数で与えられる.

$$T_n(x) = \cos(n \arccos x)\ (n \geq 0) \tag{5.26}$$

これは, 一見三角関数に見えるが (5.26) に三角関数の恒等式を用いると, 次のような $T_n(x)$ に対する式を導くことができる.

$$\begin{aligned}
T_0(x) &= 1 \\
T_1(x) &= x \\
T_2(x) &= 2x^2 - 1 \\
T_3(x) &= 4x^3 - 3x \\
T_4(x) &= 8x^4 - 8x^2 + 1 \\
&\vdots \\
T_{n+1}(x) &= 2xT_n(x) - T_{n-1}(x)\ (n \geq 1)
\end{aligned}$$

チェビシェフ多項式は, 区間 $[-1, 1]$ で $(1 - x^2)^{-\frac{1}{2}}$ の重みで直交しており, 特に,

$$\int_{-1}^{1} \frac{T_i(x) T_j(x)}{\sqrt{1 - x^2}} dx = \begin{cases} 0 & (i \neq j) \\ \dfrac{\pi}{2} & (i = j \neq 0) \\ \pi & (i = j = 0) \end{cases} \tag{5.27}$$

となる.
多項式 $T_n(x)$ は区間 $[-1,1]$ で n 個の零点をもつ. その零点の位置は,

$$x = \cos\left[\frac{\pi(k-\frac{1}{2})}{n}\right] \quad (k = 1, 2, \cdots, n) \tag{5.28}$$

である.
これと同じ区間で $T_n(x)$ は, $n+1$ 個の極値をとり, そのときの X 座標は,

$$x = \cos\left[\frac{\pi k}{n}\right] \quad (k = 0, 1, \cdots, n)$$

である. 極大値と極小値は交互に現れる. すべての極大値は $T_n(x) = 1$ であり, すべての極小値は $T_n(x) = -1$ となる. チェビシェフ多項式が関数の多項式近似で, 誤差を小さくするのに役に立つのは, この性質による.

チェビシェフ多項式は (5.28) の連続的な直交関係と同時に離散的な直交関係も満足する. すなわち, x_k $(k = 1, \cdots, m)$ が (5.27) によって与えられた $T_m(x)$ の m 個の零点で, $i, j < m$ ならば,

$$\sum_{k=1}^{m}(T_i(x_k)T_j(x_k)) = \begin{cases} 0 & (i \neq j) \\ \dfrac{m}{2} & (i = j \neq 0) \\ m & (i = j = 0) \end{cases} \tag{5.29}$$

となる.
関係式 (5.26), (5.28) および (5.29) から次の性質が得られる.

$f(x)$ を区間 $[-1,1]$ で定義された任意の関数とした時, 十分大きな N に対し, $N+1$ 個の係数 c_j $(j = 0, 1, \cdots, N)$ が

$$\begin{aligned} c_j &= \frac{2}{N}\sum_{k=1}^{N}(f(x_k)T_j(x_k)) \\ &= \frac{2}{N}\sum_{k=1}^{N}(f\left(\cos\left[\frac{\pi(k-\frac{1}{2})}{N}\right]\right)\cos\left[\frac{\pi j(k-\frac{1}{2})}{N}\right]) \end{aligned} \tag{5.30}$$

によって定義されていれば,

$$f(x) \sim \left[\sum_{j=0}^{N}(c_jT_j(x))\right] - \frac{1}{2}c_0 \tag{5.31}$$

であり, 特に, $T_N(x)$ の N 個の零点において (すなわち $x = x_k$ のとき) 左辺 = 右辺となっている.

固定された N に対して, (5.31) は x の多項式であり, その多項式は (すべての $T_N(x)$ の零点が含まれている) 区間 $[-1,1]$ で関数 $f(x)$ を近似している.

(5.31) は c_k $(k = m+1, \cdots, N)$ を無視することにより, 以下の式で表される, 同じ次数の多項式の中で最も正確な多項式の近似に短縮できる.

$$f(x) \sim \left[\sum_{k=0}^{m}(c_kT_k(x))\right] - \frac{1}{2}c_0 \tag{5.32}$$

このことは, $T_k(x)$ が, すべて ± 1 の間に値を取るので (5.32) と (5.31) の差は無視した $c_k(k = m+1, \cdots, N)$ の和よりも大きくならないためである. 特に, c_k が速く減衰する場合を考えると, 誤差の大部分は, 区間 $[-1,1]$ で

ほぼ均等に分布する $m+2$ 個の等しい極値を持つ振動関数 $c_{m+1}T_{m+1}(x)$ で占められる．この均等に誤差を引き延ばすという性質が関数を近似するのに重要である．

(5.30) の式を一般化するために変数変換

$$y \equiv \frac{x-\frac{1}{2}(b+a)}{\frac{1}{2}(b-a)} \tag{5.33}$$

を行い, 近似する $f(x)$ の定義域を $[a,b]$ とする．この変換により, 任意の区間の関数 $f(x)$ を y $(-1 \leq y \leq 1)$ のチェビシェフ多項式で近似できる．

(2) チェビシェフ係数による多項式近似

次に, 求めたチェビシェフ係数 c_k を元の変数 x の多項式係数に変換すると, 以下のような近似多項式を得る．

$$f(x) \sim \sum_{k=0}^{m}(g_k x^k) \qquad a \leq x \leq b \tag{5.34}$$

ただし, 式 (5.34) の係数 g_k はもとのチェビシェフ近似を反映しているが, この式の計算は, チェビシェフ和 (5.32) を計算するよりも高い演算精度が必要になる．というのは, 以下の理由による．

例えば, 30 次のチェビシェフ多項式を考えたとき,

$$T_{30}(x) = 2^{29}x^{30} + \cdots \tag{5.35}$$

であるが, 実は $T_{30}(x) = \cos(30\arccos(x))$ と表せる．ここで, $-1 \leq T_{30}(x) \leq 1$ であるので, 大きな係数同士の計算が生じた上で, T_{30} の値が ± 1 の間に収まる．そのときに, (5.35) における計算において, 大きい数どうしの計算で桁落ちが発生して精度が落ちてしまう．だから, チェビシェフ近似多項式の次数 m が 7, 8 を越えないときに限り, チェビシェフ近似を多項式として表すべきである．その場合でも, (5.35) における計算において, 2 桁ほど精度が低下することには注意されたい．

さて, 係数 g_k は次の手続きを順に行うことによって, 適当な m の値に変換したチェビシェフ係数 c_k $(k = 0, 1, \cdots, m)$ から導き出すことができる．

係数 c_k $(k = 0, 1, \cdots, m)$ が与えられたとき,

$$\sum_{k=0}^{m}(d_k y^k) = \sum_{k=0}^{m}(c_k T_k(y))$$

となるような係数 d_k $(k = 0, 1, \cdots, m)$ を求める．ここでは以下のクレンショウの漸化式を用いる．

$$\begin{aligned} d_{m+2} &\equiv d_{m+1} \equiv 0 \\ d_j &= 2xd_{j+1} - d_{j+2} + c_j \ (j = m, m-1, \cdots, 1) \\ f(x) &\equiv d_0 = xd_1 - d_2 + \frac{1}{2}c_0 \end{aligned}$$

次に, 係数 d_k $(k = 0, 1, \cdots, m)$ が与えられたとき,

$$\sum_{k=0}^{m}(d_k y^k) = \sum_{k=0}^{m}(g_k x^k)$$

となるような係数 g_k $(k = 0, 1, \cdots, m)$ を求める．

(ここで, x, y は (5.33) で関連づけられる．すなわち, 区間 $-1 \leq y \leq 1$ が区間 $a \leq x \leq b$ に写像される．)

まず, 以下の変換を行う.

$$g_k = d_k \left(\frac{b-a}{2} \right)^k$$

次に, 組み立て除法により g_k を求める.
ここで, $x = z$ における値 $g_k(z)$ の値を求めてみよう. まず求める多項式を

$$g_k(x) = \sum_{k=0}^{m} (g_k x^k) \tag{5.36}$$

とする. これを $(x - z)$ で割り算したとすると,

$$g_k(x) = (x-z) \sum_{k=0}^{m-1} (d_k x_{k-1}) \tag{5.37}$$

とかける.

ここで, 上の 2 式の x^k の係数を比べると,

$$\begin{aligned} g_0 &= d_0 \\ g_k &= z g_{k-1} + d_k \ (k = 1, 2, \cdots, m) \end{aligned}$$

となる.

一方, 明らかに

$$g_m(z) = d_m$$

であるから, (5.33) より, $z = \dfrac{b+a}{2}$ とおけば, 結局 $d_0 = g_0$ から出発して, 漸化式 (5.36) および (5.37) によって, g_k の値が求められる.

5.1.3 参考文献

(1) 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).

(2) 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).

(3) A. Ralston and P. Rabinowitz, “A First Course in Numerical Analysis”, McGraw-Hill, Inc. (1978)

(4) Stephen J. Balch and Garth T. Thompson, “An Efficient Algorithm For Polynomial Surface Fitting”, Computers & Geosciences Vol. 15, No. 1, pp. 107–119 (1989).

(5) 森 正武, “曲線と曲面”, 教育出版 (1974).

(6) 長田直樹, “数値微分積分法”, 現代数学社 (1987).

(7) 森 正武, “計算機による数値計算”, 岩波書店 (1986).

(8) 井阪秀高, “土木設計プログラムシリーズ 4 巻 地層の解析計算”, 山海堂 (1987).

(9) William H. Press 他, “NUMERICAL RECIPES IN FORTRAN”, CAMBRIDGE UNIVERSITY PRESS.

(10) 広中平祐他, “現代数理科学事典 第三版”, 大阪書籍.

(11) 大野豊, 磯田和男監修, “新版 数値計算ハンドブック”, オーム社.

5.2 補 間

5.2.1 ASL_dpdopl, ASL_rpdopl
不等間隔離散点補間値

(1) **機 能**

エイトケン法を用いて n 個の与えられた点 $(x_i, y_i)(i = 1, \cdots, n)$ を補間し補間点 x_l での補間値 f_l を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dpdopl  (x, y, n, xl, eps, & fl, & dl, wk);
```

単精度関数:

```
ierr = ASL_rpdopl  (x, y, n, xl, eps, & fl, & dl, 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	x	{D∗ / R∗}	n	入 力	データ点の X 座標値 x_i
2	y	{D∗ / R∗}	n	入 力	データ点の Y 座標値 y_i
3	n	I	1	入 力	データ点の個数 n
4	xl	{D / R}	1	入 力	補間点 x_l
5	eps	{D / R}	1	入 力	要求絶対精度 (既定値: 誤差判定のための単位 ×64) (5.1.2.5 参照)
6	fl	{D∗ / R∗}	1	出 力	x_l での補間値 f_l
7	dl	{D∗ / R∗}	1	出 力	補間値の絶対誤差 (5.1.2.5 参照)
8	wk	{D∗ / R∗}	n × 5	ワーク	作業領域
9	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) eps ≥ 誤差判定のための単位 ×64

(b) n ≥ 2

(c) x[i] ≠ x[j] (i ≠ j)

(d) $\min\limits_{i=1,\cdots,n}(x[i-1]) \le xl \le \max\limits_{i=1,\cdots,n}(x[i-1])$

(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.	
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.
2500	補間値の絶対誤差が要求精度以下にならないため精度は保証されない.	要求された精度の解が得られないままで処理が終了.
3000	制限条件 (b) を満足しなかった.	処理を打ち切る.
3010	制限条件 (c) を満足しなかった.	
3020	制限条件 (d) を満足しなかった.	
4000	補間値の計算の途中でオーバフローが発生した.	

(6) **注意事項**

(a) x_l 近辺の複数の補間値を求める場合は, 5.2.2 $\left\{\begin{matrix}\text{ASL_dpdapn}\\ \text{ASL_rpdapn}\end{matrix}\right\}$で補間多項式係数を求め, これより補間値を求める方が効率がよい. しかし, x_l から離れた複数の補間値を求める場合は, 5.2.2 $\left\{\begin{matrix}\text{ASL_dpdapn}\\ \text{ASL_rpdapn}\end{matrix}\right\}$ではルンゲの現象で真の値から著しくずれるため, この関数を使用した方が有効である.

(7) **使用例**

(a) 問 題

i	x_i	y_i
1	0.0	0.0
2	60.0	0.0824
3	120.0	0.2747
4	180.0	0.6502

が与えられたとき, xl における補間値を求める.

(b) 入力データ

データ点 (x, y), n=4, xl=150.0

eps $= 5.0 \times 10^{-3}$

(c) 主プログラム

```
/*      C interface example for ASL_dpdopl */

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

int main()
{
    double *x;
    double *y;
    int n;
    double x1;
    double eps;
    double y1;
    double d1;
    double *wk;
    int ierr;
    int i;
    FILE *fp;

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

```
    }

    printf( "    *** ASL_dpdopl ***\n" );
    printf( "\n     ** Input **\n\n" );
    n=4;
    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) * n ));
    if( y  == NULL )
    {
        printf( "no enough memory for array y\n" );
        return -1;
    }

    wk  = ( double * )malloc((size_t)( sizeof(double) * (n*5) ));
    if( wk  == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf%lf", &x[i],&y[i] );
    }

    fscanf( fp, "%lf", &x1 );
    fscanf( fp, "%lf", &eps);
    printf( "\tNumber of Data Points = %6d\n", n  );
    printf( "\n\tData Points (x,y)\n\n" );
    printf( "\t     i         x[i]        y[i]\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf(  "\t%6d    %8.3g    %8.3g\n", i,x[i],y[i] );
    }
    printf( "\t\n" );
    printf( "\tInterpolation Point = %8.3g \n",x1 );
    printf( "\tAbsolute Error      = %8.3g \n",eps );

    fclose( fp );

    ierr = ASL_dpdopl(x, y, n, x1, eps, &y1, &d1, wk);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tAbsolute Error      = %8.3g\n",d1 );
    printf( "\tInterpolated Value  = %8.3g\n",y1 );

    free( x );
    free( y );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dpdopl ***

 ** Input **

Number of Data Points =      4

Data Points (x,y)

     i         x[i]        y[i]
     0           0           0
     1          60      0.0824
     2         120       0.275
     3         180        0.65

Interpolation Point =      150
Absolute Error      =    0.005

 ** Output **

ierr =      0

Absolute Error      =  0.00458
Interpolated Value  =    0.435
```

5.2.2 ASL_dpdapn, ASL_rpdapn
不等間隔離散点補間値, 補間係数

(1) **機 能**

ニュートン法を用いて n 個の与えられた点 (x_i, y_i) $(i = 1, \cdots, n)$ を補間し補間多項式

$$f(x) = c_1 + \sum_{i=2}^{m+1} \prod_{j=1}^{i-1} c_i(x - u_j)$$

の係数 c_i $(i = 1, \cdots, m+1)$ と補間点 x_l での補間値 f_l を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dpdapn  (x, y, n, xl, m, c, & fl, u, wk);
```

単精度関数:

```
ierr = ASL_rpdapn  (x, y, n, xl, m, c, & fl, u, 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	x	{D∗ / R∗}	n	入 力	データ点の X 座標値 x_i
2	y	{D∗ / R∗}	n	入 力	データ点の Y 座標値 y_i
3	n	I	1	入 力	データ点の個数 n
4	xl	{D / R}	1	入 力	補間点 x_l
5	m	I	1	入 力	補間多項式次数 m
6	c	{D∗ / R∗}	m×m + 1	出 力	ニュートン差分商 (補間多項式の係数は c_i = c[m ∗ $(i-1)$]) (注意事項 (a) 参照)
7	fl	{D∗ / R∗}	1	出 力	x_l での補間値 f_l
8	u	{D∗ / R∗}	n	出 力	ソートされた X 座標値 u_j
9	wk	{D∗ / R∗}	n × 2	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) $m > 0$

(b) $n \geq m + 1$

(c) $x[i] \neq x[j]$ $(i \neq j)$

(d) $\min\limits_{i=1,\cdots,n}(x[i-1]) \leq xl \leq \max\limits_{i=1,\cdots,n}(x[i-1])$

(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), (b) を満足しなかった.	処理を打ち切る.
3010	制限条件 (c) を満足しなかった.	
3020	制限条件 (d) を満足しなかった.	
4000	補間値の計算の途中でオーバフローが発生した.	

(6) **注意事項**

(a) 求められた係数 $c[m*(i-1)]$ $(i = 1, 2, \cdots, m+1)$ は

$$\begin{aligned} yl = {} & c[0] + c[m] \times (xl - u[0]) \\ & + c[2*m] \times (xl - u[0]) \times (xl - u[1]) \\ & + \cdots \\ & + c[m*m] \times (xl - u[0]) \times \cdots \times (xl - u[m-1]) \end{aligned}$$

で表される多項式の係数である.

従って, この係数を用いて xl の近辺の値の補間値を求めたい場合は, 例えば次のようにする.

計算例 (c[m * i] を使い x 点での補間値 y を求める)

```
y=c[m*m];
for(i=m-1; i>-1; i--){
  y=c[m*i]+(x-u[i])*y;
}
```

ただし, xl の近辺の値の補間値を求める場合は上記の方法が有効であるが, xl から離れている値の補間値を求める場合に上記の方法を使うと精度が悪くなる恐れがあるので, もう 1 度この関数を使うか, 5.2.1 $\left\{\begin{matrix}\text{ASL_dpdopl}\\ \text{ASL_rpdopl}\end{matrix}\right\}$を使用したほうがよい.

(b) 入力データ

x [i] は xl 付近の $m + 1$ 個のみ有効となる.

(7) 使用例

(a) 問 題

i	x_i	y_i
1	-1.0	9.0
2	0.0	6.0
3	-3.0	-6.0
4	1.0	-4.0
5	2.0	-6.0

が与えられたとき, これらに対する補間多項式の係数, 補間値を求める.

(b) 入力データ

データ点 (x, y), n=5, xl=-2.0, m=4

(c) 主プログラム

```
/*      C interface example for ASL_dpdapn */

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

int main()
{
    double *x;
    double *y;
    int n;
    double x1;
    int m;
    double *c;
    double y1;
    double *sx;
    double *wk;
    int ierr;
    int i;
    FILE *fp;

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

    printf( "    *** ASL_dpdapn ***\n" );
    printf( "\n     ** Input **\n\n" );
    n=5;
    m=4;
    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) * n ));
    if( y  == NULL )
    {
        printf( "no enough memory for array y\n" );
        return -1;
    }
    c  = ( double * )malloc((size_t)( sizeof(double) * (m*(m+1)) ));
    if( c  == NULL )
    {
        printf( "no enough memory for array c\n" );
        return -1;
    }
    sx  = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( sx  == NULL )
    {
        printf( "no enough memory for array sx\n" );
        return -1;
    }
    wk  = ( double * )malloc((size_t)( sizeof(double) * (n*2) ));
    if( wk  == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf%lf", &x[i],&y[i] );
    }

    fscanf( fp, "%lf", &x1 );
    printf( "\n\tNumber of Data Points = %6d\n", n );
```

```
    printf( "\tInterpolating Polynomial's Degree = %6d\n", m );
    printf( "\n\tData Points (x,y)\n\n" );
    printf( "\t       i          x[i]         y[i]\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf(  "\t  %6d     %8.3g     %8.3g\n", i,x[i],y[i] );
    }
    printf( "\n\tInterpolation Point\n\n" );
    printf( "\t  %8.3g\n" ,x1);

    fclose( fp );

    ierr = ASL_dpdapn(x, y, n, x1, m, c, &y1, sx, wk);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tSorted x\n\n" );
    for( i=0 ; i<m+1 ; i++ )
    {
        printf(  "\t  u[%6d] = %8.3g\n", i,sx[i] );
    }
    printf( "\n\tInterpolated Value\n\n" );
    printf( "\t  %8.3g\n" ,y1);
    printf( "\n\tCoefficient\n\n" );
    for( i=0 ; i<m+1 ; i++ )
    {
        printf(  "\t  c[%6d] = %8.3g\n", i,c[m*i] );
    }
    free( x   );
    free( y   );
    free( c   );
    free( sx  );
    free( wk  );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dpdapn ***

 ** Input **


Number of Data Points =      5
Interpolating Polynomial's Degree =      4

Data Points (x,y)

       i          x[i]         y[i]
       0            -1            9
       1             0            6
       2            -3           -6
       3             1           -4
       4             2           -6

Interpolation Point

        -2

 ** Output **

ierr =      0

Sorted x

  u[     0] =       -1
  u[     1] =       -3
  u[     2] =        0
  u[     3] =        1
  u[     4] =        2

Interpolated Value

         2

Coefficient

  c[     0] =        9
  c[     1] =      7.5
  c[     2] =     -3.5
  c[     3] =        0
  c[     4] =      0.5
```

5.3 曲面補間

5.3.1 ASL_dplopl, ASL_rplopl
2 次元断面線上離散点補間値

(1) **機 能**

X, Y 平面上で Y 軸に平行な直線 $x = x_i$ $(i = 1, 2, \cdots, nx)$ (ここではこの直線を断面線と呼ぶ) を設定し, おのおのの x_i 断面線上のデータ点 $(y_{j,i}, z_{j,i})$ $(j = 1, \cdots, ny_i)$ を与えて, これを 3 次スプライン関数で補間し, 任意の点 (x_l, y_l) での補間値 f_l を求める. また各断面線について線上のデータを補間するスプライン係数も求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dplopl  (x, nx, y, z, my, ny, xl, yl, & fl, csp, & isw, wk);
```

単精度関数:

```
ierr = ASL_rplopl  (x, nx, y, z, my, ny, xl, yl, & fl, csp, & isw, wk);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	x	{D∗ / R∗}	nx	入 力	断面線の X 座標値 x_i
2	nx	I	1	入 力	断面線数 (配列 x の寸法)nx
3	y	{D∗ / R∗}	my×nx	入 力	x_i 断面線上のデータ点の Y 座標値 $y_{j,i}$
4	z	{D∗ / R∗}	my×nx	入 力	x_i 断面線上のデータ点の Z 座標値 $z_{j,i}$
5	my	I	1	入 力	断面線上のデータ点数の最大値 max (ny_i)
6	ny	I*	nx	入 力	各断面線上のデータ点数 ny_i
7	xl	{D / R}	1	入 力	補間点の X 座標値 x_l
8	yl	{D / R}	1	入 力	補間点の Y 座標値 y_l
9	fl	{D∗ / R∗}	1	出 力	補間点 (x_l, y_l) における補間値 f_l
10	csp	{D∗ / R∗}	内容参照	出 力	各断面線上のデータに対するスプライン係数 **大きさ:** $3 \times (\mathrm{my} - 1) \times \mathrm{nx}$
11	isw	I*	1	入出力	処理スイッチ. 初期値として 0 を入力. 処理終了後 1 を返す. isw≠0 のとき前回のスプライン係数にて処理を行う.

項番	引数と戻り値	型	大きさ	入出力	内　　容
12	wk	{D∗ R∗}	内容参照	ワーク	作業領域 **大きさ:** $5 \times \mathrm{nx} - 3$
13	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $\mathrm{nx} \geq 2,\ \mathrm{ny[i-1]} \geq 2\ (\mathrm{i} = 1, \cdots, \mathrm{nx})$

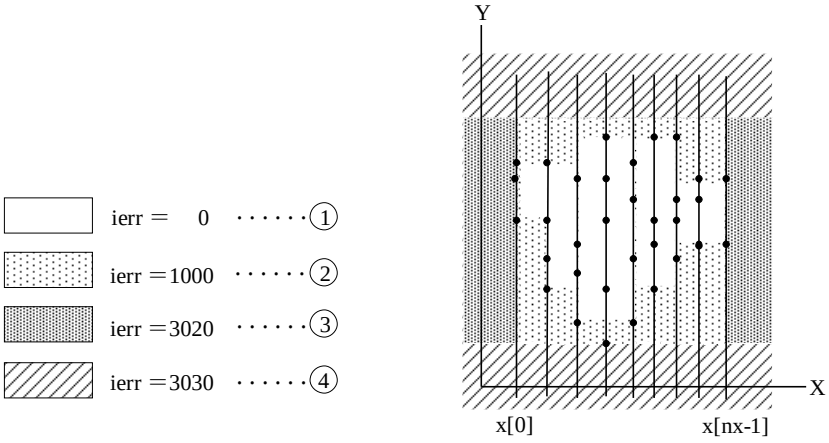
(b) $\mathrm{x[0]} < \mathrm{x[1]} < \cdots\cdots < \mathrm{x[nx-1]}$　(昇順)
$\mathrm{y[my} * \mathrm{(i-1)]} < \mathrm{y[1 + my} * \mathrm{(i-1)]} < \cdots < \mathrm{y[ny[i-1] - 1 + my} * \mathrm{(i-1)]}\ (\mathrm{i} = 1, \cdots, \mathrm{nx})$　(昇順)

(c) $\mathrm{x[0]} \leq \mathrm{xl} \leq \mathrm{x[nx-1]}$

(d) $\min\limits_{1 \leq \mathrm{i} \leq \mathrm{nx}} \mathrm{y[j-1+my} * \mathrm{(i-1)]} \leq \mathrm{yl} \leq \max\limits_{1 \leq \mathrm{i} \leq \mathrm{nx}} \mathrm{y[j-1+my} * \mathrm{(i-1)]}$

- 補間点と ierr の関係
  下図は, データ点の x – y 座標を表したものである.
  (図中 · で表示した部分) 縦線は断面線を表す.
  補間点 (xl, yl) の位置によって ierr の値が定まる (図中, 境界は値の小さな方に含まれる).

図 5–5



(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	補間点が図 5–5 の ② の範囲にある.	端点でのスプライン係数を利用し補外した値を出力する.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった. (補間点が図 5–5 の③ の範囲にある)	
3030	制限条件 (d) を満足しなかった. (補間点が図 5–5 の ④ の範囲にある)	

(6) **注意事項**

(a) スプライン係数 csp を使用して, 入力されるデータ点で囲まれる領域の体積を次の関数を用いて求めることができる.

6.2.7 {ASL_dgiipc / ASL_rgiipc}, 6.2.20 {ASL_dgiicz / ASL_rgiicz}

s ····· 各段面の面積 (実数型, 大きさ nx)

v ····· 立体の体積 (実数型, 大きさ 1)

c ····· ワーク (実数型, 大きさ (3×(nx − 1)))

プログラムの主要部分は次のようになる (倍精度版の例).

```
for(i=0; i<nx; i++)
{
   ierr = ASL_dgiicz(&y[my*i], &z[my*i], ny[i], *csp[3*i],
          y[my*i], y[ny[i]-1+my*i], &s[i]);
}
ierr = ASL_dgiipc(x, s, nx, x[0], x[nx-1], &v, c);
```

(b) 最初に本関数を呼び出すときには isw に 0 を代入しておくこと. それ以外の場合, 結果は不正である. 2 度目以降, 同一のデータを用いて別の補間点を補間する場合には isw の内容を保存し新しい xl, yl でおきかえて再度, 本関数を呼び出せばよい (このとき isw の値は 1 になっている). その場合, 既に求められたスプライン係数にて処理を行うのでより高速である.

⌇
xl=(補間点の X 座標値 ①)
yl=(補間点の Y 座標値 ①)
isw=0 ····· (isw を設定)
ierr={ASL_dplopl / ASL_rplopl} (x, ···, xl, yl, ···, isw, ···) ····· (1 度目)
⌇
xl=(補間点の X 座標値 ②)
····· (isw の設定は行わない)
yl=(補間点の Y 座標値 ②)⋮
·····
ierr={ASL_dplopl / ASL_rplopl} (x, ···, xl, yl, ···, isw, ···) ····· (2 度目)
⌇

(7) 使用例

(a) 問 題

$$x_1 = 1.0$$
$$x_2 = 2.0$$
$$x_3 = 3.0$$
$$x_4 = 4.0$$
$$x_5 = 5.0$$

j	1	2	3	4	5	6
$y_{j,1}$	0.0,	1.0,	2.0,	3.0		
$z_{j,1}$	3.0,	2.82843,	2.23607,	0.0		
$y_{j,2}$	0.0,	1.0,	2.0,	3.0,	4.0	
$z_{j,2}$	4.0,	3.87298,	3.46410,	2.64575,	0.0	
$y_{j,3}$	0.0,	1.0,	2.0,	3.0,	4.0,	4.58258
$z_{j,3}$	4.58258,	4.47214,	4.12311,	3.46410,	2.23607,	0.0
$y_{j,4}$	0.0,	1.0,	2.0,	3.0,	4.0,	4.89898
$z_{j,4}$	4.89898,	4.79583,	4.47214,	3.87298,	2.82843,	0.0
$y_{j,5}$	0.0,	1.0,	2.0,	3.0,	4.0,	5.0
$z_{j,5}$	5.0,	4.89898,	4.58258,	4.0,	3.0,	0.0

が与えられたとき, (x_l, y_l) =(1.6, 2.3), (3.2, 1.8) での補間値を求める.

(b) 入力データ

$\mathrm{x}, \mathrm{nx} = 5, \mathrm{y}, \mathrm{z}, \mathrm{my} = 6,$
$\mathrm{xl}[0] = 1.6, \mathrm{yl}[0] = 2.3,$
$\mathrm{xl}[1] = 3.2, \mathrm{yl}[1] = 1.8,$
$\mathrm{ny}[0] = 4, \mathrm{ny}[1] = 5, \mathrm{ny}[2] = 6, \mathrm{ny}[3] = 6, \mathrm{ny}[4] = 6$

(c) 主プログラム

```
/*      C interface example for ASL_dplopl */

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

int main()
{
    double *x;
    int nx;
    double *y;
    double *z;
    int my;
    int *ny;
    double xl1;
    double yl1;
    double xl2;
    double yl2;
    double fl1;
    double fl2;
    double *csp;
    int isw;
    double *wk;
    int ierr;
    int i,j;
    FILE *fp;

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

    printf( "    *** ASL_dplopl ***\n" );
    printf( "\n     ** Input **\n\n" );
    nx=5;
    my=6;
```

```
x  = ( double * )malloc((size_t)( sizeof(double) * nx ));
if( x  == NULL )
{
    printf( "no enough memory for array x\n" );
    return -1;
}
ny  = ( int * )malloc((size_t)( sizeof(int) * nx ));
if( ny  == NULL )
{
    printf( "no enough memory for array ny\n" );
    return -1;
}
y  = ( double * )malloc((size_t)( sizeof(double) * (my*nx) ));
if( y  == NULL )
{
    printf( "no enough memory for array y\n" );
    return -1;
}
z  = ( double * )malloc((size_t)( sizeof(double) * (my*nx) ));
if( z  == NULL )
{
    printf( "no enough memory for array z\n" );
    return -1;
}
csp  = ( double * )malloc((size_t)( sizeof(double) * (3*(my-1)*nx) ));
if( csp  == NULL )
{
    printf( "no enough memory for array csp\n" );
    return -1;
}
wk  = ( double * )malloc((size_t)( sizeof(double) * (5*nx-3) ));
if( wk  == NULL )
{
    printf( "no enough memory for array wk\n" );
    return -1;
}
for( i=0 ; i<nx ; i++ )
{
    fscanf( fp, "%lf", &x[i] );
}
for( i=0 ; i<nx ; i++ )
{
    fscanf( fp, "%d", &ny[i] );
}
for( i=0 ; i<nx ; i++ )
{
    for( j=0 ; j<ny[i] ; j++ )
    {
        fscanf( fp, "%lf", &y[j+my*i] );
    }
}
for( i=0 ; i<nx ; i++ )
{
    for( j=0 ; j<ny[i] ; j++ )
    {
        fscanf( fp, "%lf", &z[j+my*i] );
    }
}


fscanf( fp, "%lf", &xl1);
fscanf( fp, "%lf", &yl1);
fscanf( fp, "%lf", &xl2);
fscanf( fp, "%lf", &yl2);
printf( "\tnx    =   %6d\n", nx );
printf( "\tny[i] =   " );
for( i=0 ; i<nx ; i++ )
{
    printf(  "%6d ", ny[i] );
}
printf( "\n" );
printf( "\tmy    =   %6d\n", my );
printf( "\txl1   = %8.3g\n", xl1 );
printf( "\tyl1   = %8.3g\n", yl1 );
printf( "\txl2   = %8.3g\n", xl2 );
printf( "\tyl2   = %8.3g\n", yl2 );
printf( "\n\tx   =" );
for( i=0 ; i<nx ; i++ )
{
    printf(  "%8.3g ", x[i] );
}
printf( "\n" );
printf( "\n\ty   =\n" );
for( i=0 ; i<nx ; i++)
{
    printf( "\t     " );
    for( j=0 ; j<ny[i] ; j++ )
    {
        printf( "%8.3g ",  y[j+my*i] );
    }
    printf( "\n" );
}
printf( "\n\tz(x,y) =\n" );
```

```
    for( i=0 ; i<nx ; i++ )
    {
        printf( "\t     " );
        for( j=0 ; j<ny[i] ; j++ )
        {
            printf( "%8.3g ",  z[j+my*i] );
        }
        printf( "\n" );
    }

    fclose( fp );

    isw = 0;

    ierr = ASL_dplopl(x, nx, y, z, my, ny, xl1, yl1, &fl1, csp, &isw, wk);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tInterpolated Value at (xl1,yl1)\n\n" );
    printf( "\t  fl1 = %8.3g\n" ,fl1);

    ierr = ASL_dplopl(x, nx, y, z, my, ny, xl2, yl2, &fl2, csp, &isw, wk);


    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tInterpolated Value at (xl2,yl2)\n\n" );
    printf( "\t  fl2 = %8.3g\n" ,fl2);

    free( x );
    free( y );
    free( z );
    free( ny );
    free( csp );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dplopl ***

 ** Input **

nx    =        5
ny[i] =        4      5      6      6      6
my    =        6
xl1   =      1.6
yl1   =      2.3
xl2   =      3.2
yl2   =      1.8

x   =       1        2        3        4        5

y   =
             0        1        2        3
             0        1        2        3        4
             0        1        2        3        4     4.58
             0        1        2        3        4      4.9
             0        1        2        3        4        5

z(x,y) =
             3     2.83     2.24        0
             4     3.87     3.46     2.65        0
          4.58     4.47     4.12     3.46     2.24        0
           4.9      4.8     4.47     3.87     2.83        0
             5      4.9     4.58        4        3        0

 ** Output **

ierr =      0

Interpolated Value at (xl1,yl1)

  fl1 =     2.85

 ** Output **

ierr =      0

Interpolated Value at (xl2,yl2)

  fl2 =     4.31
```

5.3.2 ASL_dpgopl, ASL_rpgopl
2 次元格子上離散点補間値

(1) **機 能**

2 次元格子点 (x_i, y_j) $(i = 1, \cdots, nx;\ j = 1, \cdots, ny)$ 上のすべての Z 座標値 $z_{i,j}$ が与えられたとき, 格子内の任意の 1 点 (x_l, y_l) での補間値 f_l を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dpgopl (x, nx, y, ny, z, eps, xl, yl, & fl, wk);

単精度関数:

ierr = ASL_rpgopl (x, nx, y, ny, z, eps, xl, yl, & fl, 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	x	{D∗ / R∗}	nx	入 力	データ点の X 座標値 x_i
2	nx	I	1	入 力	配列 x の寸法 nx
3	y	{D∗ / R∗}	ny	入 力	データ点の Y 座標値 y_i
4	ny	I	1	入 力	配列 y の寸法 ny
5	z	{D∗ / R∗}	nx×ny	入 力	z[i − 1 + nx × (j − 1)] はデータ点 (x_i, y_j) での Z 座標値 $z_{i,j}$
6	eps	{D / R}	1	入 力	要求絶対精度 (既定値:誤差判定のための単位 × 64)
7	xl	{D / R}	1	入 力	補間点の X 座標値 x_l
8	yl	{D / R}	1	入 力	補間点の Y 座標値 y_l
9	fl	{D∗ / R∗}	1	出 力	補間点 (x_l, y_l) での補間値 f_l
10	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 **大きさ:** ny + 5 × max(nx, ny)
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) eps $\geq$ 誤差判定のための単位 $\times 64$

(b) nx ≥ 2, ny ≥ 2

(c) x[i $-$ 1] $\neq$ x[j $-$ 1], y[i $-$ 1] $\neq$ y[j $-$ 1] (i $\neq$ j)

(d) $\min\limits_{1\leq i\leq nx}(x[i-1]) \leq xl \leq \max\limits_{1\leq i\leq nx}(x[i-1])$
$\min\limits_{1\leq i\leq nx}(y[i-1]) \leq yl \leq \max\limits_{1\leq i\leq nx}(y[i-1])$

(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.	
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.
2500	補間値の絶対誤差が要求精度以下にならないため精度は保証されない.	要求した精度の解が得られないままで処理が終了.
3000	制限条件 (b) を満足しなかった.	処理を打ち切る.
3010	制限条件 (c) を満足しなかった.	
3020	制限条件 (d) を満足しなかった.	
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

$$x_1 = 1.0, \quad y_1 = 1.0$$
$$x_2 = 1.2, \quad y_2 = 1.2$$
$$x_3 = 1.4, \quad y_3 = 1.4$$
$$x_4 = 1.6, \quad y_4 = 1.6$$
$$y_5 = 1.8$$

$$\xrightarrow{j}$$

	$z_{i,j}$	y_1	y_2	y_3	y_4	y_5
	x_1	2.0,	1.56,	1.04,	0.44,	-0.24
$i\downarrow$	x_2	2.88,	2.44,	1.92,	1.32,	0.64
	x_3	3.92,	3.48,	2.96,	2.36,	1.68
	x_4	5.12,	4.68,	4.16,	3.56,	2.88

が与えられたとき, 補間点 (x_l, y_l) =(1.3, 1.5) での補間値を求める.

(b) 入力データ

x, nx=4, y, ny=5, z, eps=0.1, xl, yl

(c) 主プログラム

```
/*      C interface example for ASL_dpgopl */

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

```
int main()
{
    double *x;
    int nx;
    double *y;
    int ny;
    double *z;
    double eps;
    double xl;
    double yl;
    double fl;
    double *wk;
    int ierr;
    int i,j,nwk;
    FILE *fp;

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

    printf( "    *** ASL_dpgopl ***\n" );
    printf( "\n     ** Input **\n\n" );
    nx=4;
    ny=5;
    x  = ( double * )malloc((size_t)( sizeof(double) * nx ));
    if( x  == NULL )
    {
        printf( "no enough memory for array x\n" );
        return -1;
    }
    y  = ( double * )malloc((size_t)( sizeof(double) * ny ));
    if( y  == NULL )
    {
        printf( "no enough memory for array y\n" );
        return -1;
    }
    z  = ( double * )malloc((size_t)( sizeof(double) * (nx*ny) ));
    if( z  == NULL )
    {
        printf( "no enough memory for array z\n" );
        return -1;
    }
    nwk=ny+5*( (nx>ny) ? nx : ny );
    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<nx ; i++ )
    {
        fscanf( fp, "%lf", &x[i] );
    }
    for( i=0 ; i<ny ; i++ )
    {
        fscanf( fp, "%lf", &y[i] );
    }
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            fscanf( fp, "%lf", &z[i+nx*j] );
        }
    }


    fscanf( fp, "%lf", &eps);
    fscanf( fp, "%lf", &xl );
    fscanf( fp, "%lf", &yl );
    printf( "\tnx  =   %6d\n", nx );
    printf( "\tny  =   %6d\n", ny );
    printf( "\teps = %8.3g\n", eps );
    printf( "\txl  = %8.3g\n", xl  );
    printf( "\tyl  = %8.3g\n", yl  );
    printf( "\n\tData Points (x,y)\n\n" );
    printf( "\t  x = " );
    for( i=0 ; i<nx ; i++ )
    {
        printf(  "%8.3g ", x[i] );
    }
    printf( "\n\n" );
    printf( "\t  y =\n" );
    printf( "\t      " );
    for( i=0 ; i<ny ; i++ )
    {
        printf(  "%8.3g ", y[i] );
    }
    printf( "\n\n" );
    printf( "\t  z(x,y) =\n" );
    for( i=0 ; i<nx  ; i++ )
```

```
    {
        printf( "\t      " );
        for( j=0 ; j<ny ; j++ )
        {
            printf( "%8.3g ",  z[i+nx*j] );
        }
        printf( "\n" );
    }

    fclose( fp );

    ierr = ASL_dpgopl(x, nx, y, ny, z, eps, xl, yl, &fl, wk);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tInterpolated Value at (xl,yl)\n\n" );
    printf( "\t  %8.3g\n" ,fl );

    free( x );
    free( y );
    free( z );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dpgopl ***

 ** Input **

nx  =        4
ny  =        5
eps =      0.1
xl  =      1.3
yl  =      1.5

Data Points (x,y)

  x =        1      1.2      1.4      1.6

  y =
             1      1.2      1.4      1.6      1.8

  z(x,y) =
             2     1.56     1.04     0.44    -0.24
          2.88     2.44     1.92     1.32     0.64
          3.92     3.48     2.96     2.36     1.68
          5.12     4.68     4.16     3.56     2.88

 ** Output **

ierr =      0

Interpolated Value at (xl,yl)

      2.13
```

5.4 最小二乗近似

5.4.1 ASL_dndaao, ASL_rndaao
自動次数最小二乗近似直交多項式

(1) **機 能**

(x_i, y_i) $(i = 1, \cdots, n)$ を与えられた座標値として, 近似値 $f(x_i)$ と y_i の差の重みつき 2 乗和 $\sum_{i=1}^{n} w(x_i)\{y_i - f(x_i)\}^2$ を最小にする最適の多項式 $f(x) = \sum_{j=1}^{m+1} a_j x^{m+1-j}$ の次数 m と係数 a_j $(j = 1, \cdots, m+1)$ と $f(x_i)$ を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dndaao (x, y, w, n, a, & m, & sx, f, wk);

単精度関数:

ierr = ASL_rndaao (x, y, w, n, a, & m, & sx, f, 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	x	{D∗ / R∗}	n	入 力	データ点の X 座標値 x_i
2	y	{D∗ / R∗}	n	入 力	データ点の Y 座標値 y_i
3	w	{D∗ / R∗}	n	入 力	データ点での重み関数値 w [i-1] $= w(x_i)$ (注意事項 (a) 参照)
4	n	I	1	入 力	データ点の個数 n
5	a	{D∗ / R∗}	m + 1	出 力	近似多項式の係数 a_i (a [0] :最高次数項の係数, ⋯, a [m] :定数項)
6	m	I*	1	出 力	最適な近似多項式の次数 m
7	sx	{D∗ / R∗}	1	出 力	ierr=0 のとき sx=0 ierr=1000 のとき a は (x+sx) に対する多項式係数である.
8	f	{D∗ / R∗}	n	出 力	x_i における Y 座標近似値 $f(x_i)$
9	wk	{D∗ / R∗}	n × 8	ワーク	作業領域
10	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $\mathrm{n} \geq 2$

(b) $\mathrm{w[i-1]} \geq 0$ $(\mathrm{i} = 1, \cdots, \mathrm{n})$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	X 座標範囲に原点を含まないため一般の多項式係数では近似値が正確に求まらない.	$f(x+s)$ に対する多項式係数を a に出力し s を sx に出力する (注意事項 (b) 参照).
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.
3010	重みが 0.0 ではなく X 座標値で異なっていると見なせるデータ点の数が最適次数 +1 より少なかった. (ただし, ここでいう異なっているとは他のデータ点との座標の差が全範囲 $\times\sqrt{\text{誤差判定のための単位}}$ より大きいことをいう)	
4000	直交多項式の係数計算の途中でオーバフローまたはゼロ除算が発生した.	

(6) **注意事項**

(a) $w(x_k)$ を $w(x_j)$ $(j \neq k)$ より大きくすれば $x = x_k$ での f の値が y_k の値により近い値になる.

(b) X, Y 座標データの分布範囲に, $x = 0$ の原点を含むようにすること. 原点を含まないときは, x に対する多項式による近似計算精度が著しく低下するため, 次のように処理される.

- ierr = 1000 が返される.
- sx に x の座標シフト量 s が返される.
- a には $f(x+s) = \sum_{i=1}^{m+1} a'_i (x+s)^{m+1-i}$ に対する多項式係数 a'_i が出力される.

したがって近似多項式の値の計算は, たとえば次のようにする.

計算例 (x 点での近似多項式の値 y を求める)

```
if (ierr==1000){
  x+=sx;
}
y = a[0];
for (i=1; i<m+1; i++){
  y = y*x+a[i];
}
```

ただし, 入力データ点 x に対してこの計算で求めた y には誤差の範囲で関数を使用して求めた f の値と異なることがある. その理由は, f の値は関数中で近似多項式を求める途中の直交多項式によって得られているからである.

(7) 使用例

(a) 問 題

i	x_i	y_i	$w(x_i)$
1	0.0	5.312	1.0
2	0.3	6.044	1.0
3	0.7	8.276	1.0
4	1.0	11.000	1.0
5	1.2	13.496	1.0

が与えられたとき, これらに対する近似多項式の係数を求める.

(b) 入力データ

データ点 (x [i] , y [i]), 重み関数値 w [i] , n = 5

(c) 主プログラム

```
/*      C interface example for ASL_dndaao */

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

int main()
{
    double *x;
    double *y;
    double *w;
    int n;
    double *a;
    int m;
    double se;
    double *f;
    double *wk;
    int ierr;
    int i;
    FILE *fp;

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

    printf( "    *** ASL_dndaao ***\n" );
    printf( "\n     ** Input **\n\n" );
    n=5;
    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) * n ));
    if( y  == NULL )
    {
        printf( "no enough memory for array y\n" );
        return -1;
    }
    w  = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( w  == NULL )
    {
        printf( "no enough memory for array w\n" );
        return -1;
    }
    a  = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( a  == NULL )
    {
        printf( "no enough memory for array a\n" );
        return -1;
    }
    f  = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( f  == NULL )
    {
        printf( "no enough memory for array f\n" );
        return -1;
    }

    wk  = ( double * )malloc((size_t)( sizeof(double) * (n*8) ));
    if( wk  == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }
```

```
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf %lf %lf", &x[i],&y[i],&w[i] );
    }
    printf( "\tNumber of Data Points =%6d\n", n );
    printf( "\n\tData Points (x,y) ,Weight Function Value \n\n" );
    printf( "\t       i           x[i]          y[i]          w[i] \n" );
    for( i=0 ; i<n ; i++ )
    {
        printf(  "\t  %6d      %8.3g      %8.3g      %8.3g\n", i,x[i],y[i],w[i] );
    }

    fclose( fp );

    ierr = ASL_dndaao(x, y, w, n, a, &m, &se, f, wk);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tOptimal Degree = %6d\n", m );
    printf( "\n\tsx = %8.3g\n", se );
    printf( "\n\tCoefficients of Polynomial\n\n" );
    for( i=0 ; i<m+1 ; i++ )
    {
        printf( "\t  a[%6d]=%8.3g\n", i,a[i] );
    }
    printf( "\n\tApproximate Value\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t  f[%6d]=%8.3g\n", i,f[i] );
    }

    free( x );
    free( y );
    free( w );
    free( a );
    free( f );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dndaao ***

 ** Input **

Number of Data Points =     5

Data Points (x,y) ,Weight Function Value

       i           x[i]          y[i]          w[i]
       0              0          5.31             1
       1            0.3          6.04             1
       2            0.7          8.28             1
       3              1            11             1
       4            1.2          13.5             1

 ** Output **

ierr =      0

Optimal Degree =      4

sx =        0

Coefficients of Polynomial

  a[     0]=    1.24
  a[     1]=   -1.96
  a[     2]=    5.47
  a[     3]=   0.942
  a[     4]=    5.31

Approximate Value

  f[     0]=    5.31
  f[     1]=    6.04
  f[     2]=    8.28
  f[     3]=      11
  f[     4]=    13.5
```

5.4.2 ASL_dndapo, ASL_rndapo
最小二乗近似直交多項式

(1) **機 能**

(x_i, y_i) $(i = 1, \cdots, n)$ を与えられた座標値として, 近似値 $f(x_i)$ と y_i の差の重みつき 2 乗和 $\sum_{i=1}^{n} w(x_i)\{y_i - f(x_i)\}^2$ を最小にする最適の m 次多項式 $f(x) = \sum_{j=1}^{m+1} a_j x^{m+1-j}$ の係数 a_j $(j = 1, \cdots, m+1)$ と $f(x_i)$ を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dndapo (x, y, w, n, a, m, & sx, f, wk);

単精度関数:

ierr = ASL_rndapo (x, y, w, n, a, m, & sx, f, 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	x	{D∗ / R∗}	n	入 力	データ点の X 座標値 x_i
2	y	{D∗ / R∗}	n	入 力	データ点の Y 座標値 y_i
3	w	{D∗ / R∗}	n	入 力	データ点での重み関数値 w [i-1] $=w(x_i)$ (注意事項 (a) 参照)
4	n	I	1	入 力	データ点の個数
5	a	{D∗ / R∗}	m + 1	出 力	近似多項式の係数 a_j (a [0] :最高次数項の係数, ⋯, a [m] :定数項)
6	m	I	1	入 力	近似多項式の次数
7	sx	{D∗ / R∗}	1	出 力	ierr=0 のとき sx=0 ierr=1000 のとき a は (x+sx) に対する多項式係数である.
8	f	{D∗ / R∗}	n	出 力	x_i における Y 座標近似値 $f(x_i)$
9	wk	{D∗ / R∗}	n×8	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) m>0

(b) $\mathrm{n}' \geq \mathrm{m}+1$
n′:重みが 0.0 でなく X 座標幅が全データ分布範囲の $\sqrt{\varepsilon}$ をこえるデータ点の数
ε:誤差判定のための単位

(c) $\mathrm{w}[\mathrm{i}-1] \geq 0$ $(\mathrm{i}=1,\cdots,\mathrm{n})$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	X 座標範囲に原点を含まないため, 一般の多項式係数では近似値が正確に求まらない.	$f(x+s)$ に対する多項式係数を a に出力し s を sx に出力する (注意事項 (c) 参照).
3000	制限条件 (a) または (c) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
4000	演算の途中でオーバフローが発生した.	

(6) **注意事項**

(a) $w(x_k)$ を $w(x_j)$ $(j \neq k)$ より大きくすれば $x=x_k$ での f の値が y_k の値により近い値になる.

(b) もし, X 座標値が X 座標データ分布範囲の $\sqrt{\varepsilon}$ 以下の幅のデータを入力する場合は, 制限条件 (b) を満たすようにする.

(c) X, Y 座標データの分布範囲に, $x=0$ の原点を含むようにすること. 原点を含まないときは, x に対する多項式による近似計算精度が著しく低下するため, 次のように処理される.

- ierr = 1000 が返される.
- sx に X 座標シフト量 s が返される.
- a には $f(x+s)=\sum_{i=1}^{m+1} a'_i(x+s)^{m+1-i}$ に対する多項式係数 a'_i が出力される.

従って近似多項式の値の計算は, 例えば次のようにする.
計算例 (x 点での近似多項式の値 y を求める)

```
if (ierr==1000){
  x+=sx;
}
y = a[0];
for (i=1; i<m+1; i++){
  y = y*x+a[i];
}
```

ただし, 入力データ点 x に対してこの計算で求めた y には誤差の範囲で, 関数を使用して求めた f の値と異なることがある. その理由は, f の値は関数中で近似多項式を求める途中の直交多項式によって得られているからである.

(d) 近似多項式の次数 m を大きくとるとき, 精度低下をおこしやすいので倍精度を利用した方がよい.

(7) 使用例

(a) 問 題

i	x_i	y_i	$w(x_i)$
1	0.0	5.312	1.0
2	0.3	6.044	1.0
3	0.7	8.276	1.0
4	1.0	11.000	1.0
5	1.2	13.496	1.0

が与えられたとき, これらに対する近似多項式の係数を求める.

(b) 入力データ

データ点 (x [i] , y [i]), 重み関数値 w [i] , n=5, m=3

(c) 主プログラム

```
/*      C interface example for ASL_dndapo */

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

int main()
{
    double *x;
    double *y;
    double *w;
    int n;
    double *a;
    int m;
    double se;
    double *f;
    double *wk;
    int ierr;
    int i;
    FILE *fp;

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

    printf( "    *** ASL_dndapo ***\n" );
    printf( "\n     ** Input **\n\n" );
    n=5;
    m=3;
    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) * n ));
    if( y  == NULL )
    {
        printf( "no enough memory for array y\n" );
        return -1;
    }
    w  = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( w  == NULL )
    {
        printf( "no enough memory for array w\n" );
        return -1;
    }
    a  = ( double * )malloc((size_t)( sizeof(double) * (m+1)  ));
    if( a  == NULL )
    {
        printf( "no enough memory for array a\n" );
        return -1;
    }
    f  = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( f  == NULL )
    {
        printf( "no enough memory for array f\n" );
        return -1;
    }

    wk  = ( double * )malloc((size_t)( sizeof(double) * (n*8) ));
    if( wk  == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }
```

```
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf,%lf,%lf", &x[i],&y[i],&w[i] );
    }
    printf( "\tDegree of Approximate Value =%6d\n", m );
    printf( "\tNumber of Data Points =%6d\n", n );
    printf( "\n\tData Points (x,y) ,Weight Function Value \n\n" );
    printf( "\t       i           x[i]          y[i]          w[i] \n" );
    for( i=0 ; i<n ; i++ )
    {
        printf(  "\t  %6d      %8.3g      %8.3g      %8.3g\n", i,x[i],y[i],w[i] );
    }

    fclose( fp );

    ierr = ASL_dndapo(x, y, w, n, a, m, &se, f, wk);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tsx = %8.3g\n", se );
    printf( "\n\tCoefficients of x\n\n" );
    for( i=0 ; i<m+1 ; i++ )
    {
        printf(  "\t  a[%6d]=%8.3g\n", i,a[i] );
    }
    printf( "\n\tApproximate Value\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf(  "\t  f[%6d]=%8.3g\n", i,f[i] );
    }

    free( x );
    free( y );
    free( w );
    free( a );
    free( f );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dndapo ***

 ** Input **

Degree of Approximate Value =     3
Number of Data Points =     5

Data Points (x,y) ,Weight Function Value

       i           x[i]          y[i]          w[i]
       0             0          5.31             1
       1           0.3          6.04             1
       2           0.7          8.28             1
       3             1            11             1
       4           1.2          13.5             1

 ** Output **

ierr =      0

sx =        0

Coefficients of x

  a[     0]=    1.02
  a[     1]=    3.28
  a[     2]=    1.41
  a[     3]=    5.31

Approximate Value

  f[     0]=    5.31
  f[     1]=    6.06
  f[     2]=    8.26
  f[     3]=      11
  f[     4]=    13.5
```

5.4.3 ASL_dndanl, ASL_rndanl
最小二乗近似非線形型関数

(1) **機 能**

与えられた座標値 (x_i, y_i) $(i = 1, \cdots, n)$ に近似関数 $f(x, \boldsymbol{a})$ を当てはめ, 残差 $y_i - f(x_i, \boldsymbol{a})$ $(i = 1, \cdots, n)$ の二乗和が極小になるようにパラメータ a_i $(i = 1, \cdots, m)$ を最適化する (a_i は$\boldsymbol{a}$の成分).

(2) **使用法**

倍精度関数:

ierr = ASL_dndanl (f, x, y, n, er, & nev, a, m, yf, & s, iwk, wk);

単精度関数:

ierr = ASL_rndanl (f, x, y, n, er, & nev, a, m, yf, & 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	f	{D / R}	−	入 力	近似する関数 $f(x, \boldsymbol{a})$ を定義する関数 f(x, a) の関数名 (注意事項 (a) 参照)
2	x	{D∗ / R∗}	n	入 力	データ点の X 座標値 x_i
3	y	{D∗ / R∗}	n	入 力	データ点の Y 座標値 y_i
4	n	I	1	入 力	データ点の個数 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*	1	入 力	関数 $f(x,\ \boldsymbol{a})$ の最大評価回数 (既定値:100 × n × m)
				出 力	実際の関数評価回数
7	a	{D∗ / R∗}	m	入 力	係数の初期値 $\boldsymbol{a}^0$
				出 力	最適の係数 $\boldsymbol{a}^*$
8	m	I	1	入 力	係数の数 m
9	yf	{D∗ / R∗}	n	出 力	x_i での近似関数 $f(x_i, \boldsymbol{a}^*)$ の値
10	s	{D∗ / R∗}	1	出 力	残差の二乗和の値 $s = \sum_{i=1}^{n}(y_i - f(x_i, \boldsymbol{a}^*))^2$
11	iwk	I*	4×m	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**: n × (2 × m + 1) + m × (m + 4)
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) $0 < \mathrm{m} \leq \mathrm{n}$

(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 a, yf, s の値を出力して処理を打ち切る.
4100	最急降下解を計算できなかった.	
4200	2m 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 f の作り方は次のようにする.

```
double FORTRAN f(double *x, double *a)
{
        return f(*x, a);
}
```

(b) 収束判定は次式によって行い, $\boldsymbol{a} + \Delta\boldsymbol{a}$ を解とする.

$$\|\Delta\boldsymbol{a}\| \leq \mathrm{er} \times \max(1, \|\boldsymbol{a} + \Delta\boldsymbol{a}\|)$$

ここで, $\Delta\boldsymbol{a}$ は$\boldsymbol{a}$に対する修正ベクトルであり, $\|\boldsymbol{a}\| = \max|a_i|$ 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) x_i, y_i の座標のオーダは, ほぼ等しい必要がある. オーダが異なるときは x_i 側を y_i と同じオーダになるようにスケーリング (すべての x_i に同じ値を乗じる) して a_i を求め, もとの x_i の値に対する a_i になるように a_i を変換する.

例 ワイブル分布曲線へのあてはめ

$$y = a_i[1 - \exp\{-(x - a_3)/a_2\}] + a_4$$

に対する最適の a_1, a_2, a_3, a_4 を求める.

(i) y_i $(i = 1, \cdots, n)$ の最小値を $y_{\min}$, x_i $(i = 1, \cdots, n)$ の最小値を $x_{\min}$ とする.

$$y_i' = y_i - y_{\min}\ (i = 1, \cdots, n),\ a_4' = y_{\min}$$

$$x_i' = x_i - x_{\min}\ (i = 1, \cdots, n),\ a_3' = x_{\min}$$

と変換して

y_i' $(i = 1, \cdots, n)$ の最大値を $y_{\max}$, x_i' $(i = 1, \cdots, n)$ の最大値を $x_{\max}$ とする ($a_3 = a_4 = 0$ のときは, この操作は不要). (座標変換)

(ii) $s = y_{\max}/x_{\max}$ とし,
$x''_i = sx'_i$ $(i = 1, \cdots, n)$ と変換 (スケーリング) する.

(iii) a_1, a_2 の初期値は $y_{\max}$ 程度, a_3, a_4 の初期値は 0 としてデータ (x''_i, y'_i) $(i = 1, \cdots, n)$ を用いて $f(x, \boldsymbol{a}) = a'_1[1 - \exp\{-(x - a'_3)/a'_2\}] + a'_4$ に対して非線形最小二乗近似を行い, a'_1, a'_2, a'_3, a'_4 を求める.

(iv) a_1, a_2, a_3, a_4 は,

$$\begin{aligned} a_1 &= a'_1 \\ a_2 &= a'_2/s \\ a_3 &= a'_3/s + x_{\min} \\ a_4 &= a'_4 + y_{\min} \end{aligned}$$

と求まる.

(7) 使用例

(a) 問 題

11 個のデータ点 (x, y)

$(\ -5.0,\ \ 2.7\ )$
$(\ -4.0,\ \ 2.9\ )$
$(\ -3.0,\ \ 3.1\ )$
$(\ -2.0,\ \ 3.4\ )$
$(\ \ 1.0,\ \ 3.9\ )$
$(\ \ 0.0,\ \ 4.7\ )$
$(\ \ 1.0,\ \ 6.0\ )$
$(\ \ 2.0,\ \ 7.8\ )$
$(\ \ 3.0,\ \ 7.9\ )$
$(\ \ 4.0,\ \ 6.3\ )$
$(\ \ 5.0,\ \ 5.2\ )$

に対して関数,

$$f(x) = \frac{hw^2}{(x - x_0)^2 + w^2} + a_0 + a_1 x$$

を当てはめて, x_0, w, h, a_0, a_1 の最適値を求める.

初期値は,

$x_0 = 0.0$, $w = 1.0$, $h = 6.0$, $a_0 = 3.5$, $a_1 = 0.2$ とする.

(b) 入力データ

関数名: fndanl

配列 x, y, n=11, er=0.0 nev=0

配列 a:

x_0, w, h, a_0, a_1 を順に, a [0] , a [1] , a [2] , a [3] , a [4] に割り当てる

m=5

(c) 主プログラム

```
/*      C interface example for ASL_dndanl */

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

#ifdef __cplusplus
extern "C"
{
#endif
```

```
#ifdef __STDC__
double f(double *x,double *a)
#else
double f(x,a)
double *x;
double *a;
#endif
{
    double f1,f2;
    f1=a[2]*a[1]*a[1]/(((*x)-a[0])*((*x)-a[0])+(a[1]*a[1]));
    f2=a[3]+a[4]*(*x);
    return (f1+f2);
}
#ifdef __cplusplus
}
#endif

int main()
{
    double *xd;
    double *yd;
    int n;
    double er;
    int nev;
    double *x;
    int m;
    double *y;
    double s;
    int *iwk;
    double *wk;
    int ierr;
    int i;
    FILE *fp;

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

    printf( "    *** ASL_dndanl ***\n" );
    printf( "\n     ** Input **\n\n" );
    n=11;
    m=5;
    xd  = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( xd  == NULL )
    {
        printf( "no enough memory for array xd\n" );
        return -1;
    }
    yd  = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( yd  == NULL )
    {
        printf( "no enough memory for array yd\n" );
        return -1;
    }
    x  = ( double * )malloc((size_t)( sizeof(double) * m ));
    if( x  == NULL )
    {
        printf( "no enough memory for array x\n" );
        return -1;
    }
    y  = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( y  == NULL )
    {
        printf( "no enough memory for array y\n" );
        return -1;
    }

    wk  = ( double * )malloc((size_t)( sizeof(double) * (n*(2*m+1)+m*(m+4)) ));
    if( wk  == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

    iwk  = ( int * )malloc((size_t)( sizeof(int) * (3*m) ));
    if( iwk  == NULL )
    {
        printf( "no enough memory for array iwk\n" );
        return -1;
    }

    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &xd[i] );
    }
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &yd[i] );
    }
    fscanf( fp, "%d", &nev );
    fscanf( fp, "%lf", &er );
    for( i=0 ; i<m ; i++ )
    {
        fscanf( fp, "%lf", &x[i] );
    }
```

```
    printf( "\tm  =  %6d\n", m );
    printf( "\tn  =  %6d\n", n );
    printf( "\tnev=  %6d\n", nev );
    printf( "\ter =%8.3g\n", er );
    printf( "\n\tCoordinates (x,y)\n\n" );
    printf( "\t       i           x[i]          y[i]\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf(  "\t  %6d      %8.3g      %8.3g\n", i,xd[i],yd[i] );
    }
    printf( "\n\tInitial Value of Coefficients\n\n" );
    for( i=0 ; i<m ; i++ )
    {
        printf(  "\t  a[%6d]=%8.3g\n", i,x[i] );
    }

    fclose( fp );

    ierr = ASL_dndanl(f, xd, yd, n, er, &nev, x, m, y, &s, iwk, wk);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tnev  = %6d\n", nev  );
    printf( "\n\tOptimized Coefficients\n\n" );
    for( i=0 ; i<m ; i++ )
    {
        printf(  "\t  a[%6d]=%8.3g\n", i,x[i] );
    }
    printf( "\n\tLeast Squares\n\n" );
    printf( "\t  s =%8.3g\n",s );
    printf( "\n\tFunction Value\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf(  "\t  yf[%6d]=%8.3g\n", i,y[i] );
    }

    free( xd );
    free( yd );
    free( x );
    free( y );
    free( iwk );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dndanl ***

 ** Input **

m  =       5
n  =      11
nev=       0
er =       0

Coordinates (x,y)

       i           x[i]          y[i]
       0             -5           2.7
       1             -4           2.9
       2             -3           3.1
       3             -2           3.4
       4             -1           3.9
       5              0           4.7
       6              1             6
       7              2           7.8
       8              3           7.9
       9              4           6.3
      10              5           5.2

Initial Value of Coefficients

  a[     0]=       0
  a[     1]=       1
  a[     2]=       6
  a[     3]=     3.5
  a[     4]=     0.2

 ** Output **

ierr =      0

nev  =    759

Optimized Coefficients

  a[     0]=    2.49
  a[     1]=     1.8
  a[     2]=    4.97
  a[     3]=    2.96
  a[     4]=   0.108

Least Squares
```

```
  s = 0.00361

Function Value

  yf[     0]=    2.69
  yf[     1]=    2.88
  yf[     2]=    3.12
  yf[     3]=    3.43
  yf[     4]=     3.9
  yf[     5]=    4.66
  yf[     6]=    6.02
  yf[     7]=     7.8
  yf[     8]=    7.89
  yf[     9]=    6.31
  yf[    10]=    5.19
```

5.5 最小二乗曲面近似

5.5.1 ASL_dnrapl, ASL_rnrapl
2 次元任意データ最小二乗近似多項式

(1) **機 能**

平面上の任意のデータ点 (x_k, y_k, z_k) $(k = 1, \cdots, n)$ において, データ点での Z 座標値 z_k と多項式の値 $f(x_k, y_k)$ との差の二乗和が最小となるような x, y について m 次の近似多項式

$$f(x,y) = \sum_{j=1}^{m+1} \sum_{i=1}^{m+2-j} a_{i,j} x^{i-1} y^{j-1}$$

の係数 $a_{i,j}$ と入力データ点における近似値 $f(x_k, y_k)$ $(k = 1, \cdots, n)$ を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dnrapl (x, y, z, n, m, a, f, iw, wk);

単精度関数:

ierr = ASL_rnrapl (x, y, z, n, m, a, f, iw, 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	x	{D∗ R∗}	n	入 力	データ点の X 座標値 x_k
2	y	{D∗ R∗}	n	入 力	データ点の Y 座標値 y_k
3	z	{D∗ R∗}	n	入 力	データ点の Z 座標値 z_k
4	n	I	1	入 力	データ点の個数 n
5	m	I	1	入 力	近似多項式の x, y に関する次数 m
6	a	{D∗ R∗}	内容参照	出 力	近似多項式係数 $a_{i,j}$ **大きさ:** $(\mathrm{m}+1) \times (\mathrm{m}+2)/2$
7	f	{D∗ R∗}	n	出 力	(x_k, y_k) での Z 座標の近似値 $f(x_k, y_k)$
8	iw	I*	内容参照	ワーク	作業領域 **大きさ:** $(\mathrm{m}+1) \times (\mathrm{m}+2)/2$
9	wk	{D∗ R∗}	内容参照	ワーク	作業領域 **大きさ:** $(\mathrm{m}+2)^4/4$
10	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $\mathrm{n} \geq (\mathrm{m}+1) \times (\mathrm{m}+2)/2$

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

(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	近似次数が 10 次を越える. (近似誤差が大きい可能性がある. 注意事項 (c) 参照)	与えられた次数により処理を続ける.
2100	連立 1 次方程式方程式を解く際の LU 分解において, 対角要素が 0 に近いものがあった. 精度の良い結果が得られない場合がある.	処理を続ける.
3000	制限条件 (a) を満足しなかった. またはデータ (x [k] , y [k]) のうち相異なるものの個数を n' としたとき $\mathrm{n}' < (\mathrm{m}+1) \times (\mathrm{m}+2)/2$	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
4000+i	内部で正規方程式を解くために使用している LU 分解の関数の i 段目の処理において, ピボットが 0.0 となった.	

(6) **注意事項**

(a) 全入力データ点 (x_k, y_k) $(k = 1, \cdots, n)$ が一直線上にあると, 曲面を形成することができない. この場合得られた結果は不定解の 1 つである.

(b) 本関数では効率化のため, $a_{i,j}$ を 1 次元配列に格納して出力する. $a_{i,j}$ と a の対応関係は, 次数 m に対して次のようになる.

$$\mathrm{a_{i,j}} = \mathrm{a[i-1+\{(j-1)*(2*m-j+4)\}/2]}$$

これは下図において, 次のように対応している.
まず 1 行目に並んでいる項を左から右にみていく. 1 項目は定数項であり a [0] が対応する.
以下順に $\mathrm{a}[1] = x$ の係数, $\mathrm{a}[2] = x^2$ の係数, $\cdots$, $\mathrm{a}[m] = x^m$ の係数というようになっている. 右端まできたので次の行に移り同様に項をみていく.
$\mathrm{a}[m+1] = y$ の係数, $\cdots$, $\mathrm{a}[2m] = yx^{m-1}$ となる. こうして $m+1$ 行目の $\mathrm{a}[-1+(m+1)*(m+2)/2] = y^m$ の係数まで対応している.

$$\begin{array}{lllll} 1 & x & x^2 & x^3 & \cdots \quad x^m \\ y & xy & x^2y & \cdots & x^{m-1}y \\ y^2 & xy^2 & \cdots & . & \\ \cdots & & . & & \\ \cdots & . & & & \\ y^m & & & & \end{array}$$

(c) この関数は与えられた曲面上の点をただ 1 つの多項式によって近似するのでデータによっては無意味な結果を出すことがある. 特に近似多項式の次数が大きい場合には, 出力結果が適切かどうかチェックした方

がよい.

出力結果が適切であるかどうかをチェックする一つの指標として次式で定義される fitting 率がある.

$$\mathrm{zs}\cdots \parallel f_k \parallel_2 = \sqrt{\sum_{\mathrm{k}=1}^{\mathrm{n}-1} \mathrm{z[k]}^2} \qquad \text{(実数型, 大きさ 1)}$$

$$\mathrm{rs}\cdots \mid \mathrm{x} \mid = \sqrt{\sum_{\mathrm{k}=1}^{\mathrm{n}-1} (\mathrm{f[k]} - \mathrm{z[k]})^2} \qquad \text{(実数型, 大きさ 1)}$$

$$\mathrm{df}\cdots \text{ワーク用変数} \qquad \text{(実数型, 大きさ 1)}$$

$$\mathrm{fit}\cdots \text{fitting 率} = \frac{\parallel f_k \parallel_2}{\mid x \mid + \parallel f_k \parallel_2} \times 100 \qquad \text{(実数型, 大きさ 1)}$$

fitting 率は次のようなプログラムで計算できる (主要部分のみ).

```
zs = 0.0;
rs = 0.0;
for( k = 0; k < n; k++)
{
    zs = zs + z[k] * z[k];
    df = f[k] - z[k];
    rs = rs + df * df;
}
zs = sqrt(zs);
rs = sqrt(rs);
fit = 1.0e2;
if((zs != 0.0) || (rs != 0.0)){
    fit = zs/(zs + rs) * 1.0e2;
}
```

この fitting 率が低いときには近似区間の分割, 入力データの原点付近へのシフト, スケーリングなどを行ってからこの関数を利用されたい.

(d) 近似多項式係数 a の値は次のような場合に, 単精度と倍精度あるいは OS によって異なることがある.

- 次数が高次の場合.
- 入力データの振動が激しい場合.

(e) 多項式係数 a を用いて入力データ点以外の点で近似値を求める方法.

xl	···	近似点の x 座標値	(実数型, 大きさ 1)
yl	···	近似点の y 座標値	(実数型, 大きさ 1)
fl	···	近似値	(実数型, 大きさ 1)
s	···	ワーク	(実数型, 大きさ 1)
w	···	ワーク	(実数型, 大きさ m + 1)
id	···	a の指標用ワーク	(整数型, 大きさ 1)

とすればプログラムの主要部分は次のようになる (倍精度版の例).

```
fl = a[0];
s = 1.0;
w[0] = 1.0;
for( i=1 ; i<(m+1) ; i++ )
{
```

```
    s *= xl;
    w[i] = s;
    fl += s * a[i];
  }
  for( j=1 ; j<(m+1) ; j++ )
  {
    for( i=0 ; i<(m+2-j) ; i++ )
    {
      w[i] *= y1;
      id = (i+1) + j*(2*(m+1)-j+1)/2;
      fl += w[i]*a[id-1];
    }
  }
```

(7) **使用例**

(a) 問 題

k	x_k	y_k	z_k
1	6.95	−0.48	48.24
2	2.44	9.70	−17.57
3	0.89	−0.70	0.67
4	7.27	−7.51	38.75
5	−7.36	−1.18	53.82
6	−0.07	−4.72	−5.56
7	4.55	−5.84	12.18
8	−1.26	7.45	−12.29

が与えられたとき近似多項式の係数および入力点での近似値を求める.

(b) 入力データ

x, y, z, n = 8, m = 2

(c) 主プログラム

```
/*      C interface example for ASL_dnrapl */

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

int main()
{
    double *x;
    double *y;
    double *z;
    int n;
    int m;
    double *a;
    double *f;
    int *iw;
    double *wk;
    int ierr;
    int i,niwk,nwk;
    FILE *fp;

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

    printf( "    *** ASL_dnrapl ***\n" );
    printf( "\n     ** Input **\n\n" );
    n=8;
    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) * n ));
    if( y  == NULL )
    {
        printf( "no enough memory for array y\n" );
        return -1;
    }
    z  = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( z  == NULL )
    {
        printf( "no enough memory for array z\n" );
        return -1;
    }
    f  = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( f  == NULL )
    {
        printf( "no enough memory for array f\n" );
        return -1;
    }

    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &x[i] );
    }
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &y[i] );
    }
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &z[i] );
    }
    fscanf( fp, "%d", &m );
    niwk=((m+1)*(m+2))/2;
    a  = ( double * )malloc((size_t)( sizeof(double) * niwk ));
    if( a  == NULL )
    {
        printf( "no enough memory for array a\n" );
        return -1;
    }
    iw  = ( int * )malloc((size_t)( sizeof(int) * niwk ));
    if( iw  == NULL )
    {
        printf( "no enough memory for array iw\n" );
        return -1;
    }
    nwk=((m+2)*(m+2)*(m+2)*(m+2))/4;
    wk  = ( double * )malloc((size_t)( sizeof(double) * nwk ));
    if( wk  == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

    printf( "\tn  =%6d\n", n );
    printf( "\tm  =%6d\n", m );
    printf( "\n\t     x             y             z\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf(  "\t%8.3g      %8.3g      %8.3g\n", x[i],y[i],z[i] );
    }

    fclose( fp );

    ierr = ASL_dnrapl(x, y, z, n, m, a, f, iw, wk);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tCoefficient of Polynomial for x,y\n\n" );
    for( i=0 ; i<((m+1)*(m+2)/2) ; i++ )
    {
        printf(  "\t  a[%6d] = %8.3g\n", i,a[i] );
    }
    printf( "\n\tApproximate Value of z\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf(  "\t  f[%6d] = %8.3g\n", i,f[i] );
    }

    free( x );
    free( y );
    free( z );
    free( a );
    free( f );
    free( iw );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dnrapl ***

 ** Input **

n  =     8
m  =     2

      x             y             z
    6.95         -0.48          48.2
    2.44           9.7         -17.6
    0.89          -0.7          0.67
    7.27         -7.51          38.8
   -7.36         -1.18          53.8
   -0.07         -4.72         -5.56
    4.55         -5.84          12.2
   -1.26          7.45         -12.3

 ** Output **

ierr =      0

Coefficient of Polynomial for x,y

  a[     0] =  0.00104
  a[     1] = -0.000183
  a[     2] =         1
  a[     3] = -0.000689
  a[     4] = 6.84e-05
  a[     5] =    -0.25

Approximate Value of z

  f[     0] =     48.2
  f[     1] =    -17.6
  f[     2] =    0.671
  f[     3] =     38.8
  f[     4] =     53.8
  f[     5] =    -5.56
  f[     6] =     12.2
  f[     7] =    -12.3
```

5.5.2 ASL_dngapl, ASL_rngapl
2 次元格子データ最小二乗近似多項式

(1) **機 能**

2 次元格子点 (x_i, y_j) $(i = 1, \cdots, nx;\ j = 1, \cdots, ny)$ 上のすべての Z 座標値 $z_{i,j}$ が与えられたとき, それと多項式の値 $f(x_i, y_j)$ との差の二乗和が最小となるような x, y についての近似多項式

$$f(x, y) = \sum_{i=1}^{ix+1} \sum_{j=1}^{iy+1} a_{i,j} x^{i-1} y^{j-1}$$

の係数 $a_{i,j}$ と入力格子点における近似値 $f(x_i, y_j)$ $(i = 1, \cdots, nx;\ j = 1, \cdots, ny)$ を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dngapl  (x, nx, y, ny, z, ix, iy, a, f, wk);
```

単精度関数:

```
ierr = ASL_rngapl  (x, nx, y, ny, z, ix, iy, a, f, 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	x	{D∗ / R∗}	nx	入 力	データ点の X 座標値 x_i
2	nx	I	1	入 力	X 方向データ点数 nx
3	y	{D∗ / R∗}	ny	入 力	データ点の Y 座標値 y_j
4	ny	I	1	入 力	Y 方向データ点数 ny
5	z	{D∗ / R∗}	nx×ny	入 力	データ点 (x_i, y_j) での Z 座標値 $z_{i,j}$
6	ix	I	1	入 力	近似多項式の x に関する最高次数 ix
7	iy	I	1	入 力	近似多項式の y に関する最高次数 iy
8	a	{D∗ / R∗}	内容参照	出 力	x, y についての近似多項式係数 $a_{i,j}$ **大きさ:** $(\mathrm{ix}+1) \times (\mathrm{iy}+1)$
9	f	{D∗ / R∗}	nx×ny	出 力	(x_i, y_j) での Z 座標値の近似値 $f(x_i, y_j)$
10	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 **大きさ:** $9 \times (\mathrm{nx} + 2 \times \mathrm{ny})$
11	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $\mathrm{nx} > 2, \mathrm{ny} > 2$

(b) $\mathrm{x[i]} \neq \mathrm{x[j]}$ $(\mathrm{i} \neq \mathrm{j})$
$\mathrm{y[i]} \neq \mathrm{y[j]}$ $(\mathrm{i} \neq \mathrm{j})$

(c) $0 < \mathrm{ix} \leq \min(8, \mathrm{nx} - 1)$
$0 < \mathrm{iy} \leq \min(8, \mathrm{ny} - 1)$

(5) **エラーインディケータ (戻り値)**

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

(6) **注意事項**

(a) 2 次元格子点 (x_i, y_j) に対応する Z 座標値を $z_{i,j}$ $(i = 1, \cdots, \mathrm{nx};\ j = 1, \cdots, \mathrm{ny})$ としたとき, $z_{i,j}$ は配列 z へ次のように格納する.
$\mathrm{z}[(i-1) + \mathrm{nx} \times (j-1)] = z_{i,j}$

(b) 配列 a には, $a_{i,j}$ が $a_{1,1}, \cdots, a_{1,iy+1}, a_{2,1}, \cdots, a_{2,iy+1}, \cdots, a_{ix+1,1}, \cdots, a_{ix+1,iy+1}$ の順に格納される.

(c) 多項式係数 a を用いて入力データ点以外の点で近似値を求める方法.

xl	$\cdots$	近似点の x 座標値	(実数型, 大きさ 1)
yl	$\cdots$	近似点の y 座標値	(実数型, 大きさ 1)
fl	$\cdots$	近似値	(実数型, 大きさ 1)
s	$\cdots$	ワーク	(実数型, 大きさ 1)
id	$\cdots$	a の指標用ワーク	(整数型, 大きさ 1)
ie	$\cdots$	a の指標用ワーク	(整数型, 大きさ 1)
k, lt	$\cdots$	変数	(整数型, 大きさ 1)

とすればプログラムの主要部分は次のようになる.

```
id = (ix + 1) * (iy + 1);
fl = a[id - 1];
for ( lt = 1; lt < (iy + 1); i++)
{
  fl = fl * yl + a[id - l - 1];
}
for ( k = ix; k > 0; k--)
{
  ie = k * (iy + 1)
  s = a[ie - 1]
  for ( lt = 1; lt < iy + 1; lt--)
  {
    s = s * yl + a[ie - l - 1];
  }
```

$$\mathrm{fl} = \mathrm{fl} * \mathrm{xl} + \mathrm{s};$$
}

(d) この関数は与えられた曲面上の点をただ 1 つの多項式によって近似するのでデータによっては無意味な結果を出すことがある. 特に近似多項式の次数が大きい場合には出力結果が適切かどうかチェックした方がよい. 出力結果が適切であるかどうかをチェックする一つの指標として fitting 率 (5.5.1 参照) がある. この fitting 率が低いときには近似区間の分割, 入力データの原点付近へのシフト, スケーリングなどを行ってからこの関数を利用されたい.

(7) 使用例

(a) 問 題

格子座標

$x_1 = -3.0 \quad y_1 = -2.0$
$x_2 = -1.0 \quad y_2 = -1.0$
$x_3 = 1.0 \quad y_3 = 0.0$
$x_4 = 3.0 \quad y_4 = 1.0$
$y_5 = 2.0$

		j →				
	$z_{i,j}$	y_1	y_2	y_3	y_4	y_5
	x_1	28.0	29.5	27.0	20.5	10.0
$i\downarrow$	x_2	−2.0	2.5	3.0	−0.5	−8.0
	x_3	−8.0	−0.5	3.0	2.5	−2.0
	x_4	10.0	20.5	27.0	29.5	28.0

が与えられたとき, 近似多項式の係数および入力格子点上の近似値を求める.

(b) 入力データ

x, nx=4, y, ny=5, z, ix=2, iy=2

(c) 主プログラム

```
/*      C interface example for ASL_dngapl */

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

int main()
{
    double *x;
    int nx;
    double *y;
    int ny;
    double *z;
    int ix;
    int iy;
    double *a;
    double *f;
    double *wk;
    int ierr;
    int i,j,nwk;
    FILE *fp;

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

    printf( "    *** ASL_dngapl ***\n" );
    printf( "\n     ** Input **\n\n" );
    nx=4;
    ny=5;
    x  = ( double * )malloc((size_t)( sizeof(double) * nx ));
    if( x  == NULL )
    {
        printf( "no enough memory for array x\n" );
        return -1;
    }
    y  = ( double * )malloc((size_t)( sizeof(double) * ny ));
```

```
if( y  == NULL )
{
    printf( "no enough memory for array y\n" );
    return -1;
}
z  = ( double * )malloc((size_t)( sizeof(double) * (nx*ny) ));
if( z  == NULL )
{
    printf( "no enough memory for array z\n" );
    return -1;
}
f  = ( double * )malloc((size_t)( sizeof(double) * (nx*ny) ));
if( f  == NULL )
{
    printf( "no enough memory for array f\n" );
    return -1;
}

nwk=9*(nx+2*ny);
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<nx ; i++ )
{
    fscanf( fp, "%lf", &x[i] );
}
for( i=0 ; i<ny ; i++ )
{
    fscanf( fp, "%lf", &y[i] );
}
for( i=0 ; i<nx ; i++ )
{
    for( j=0 ; j<ny ; j++ )
    {
        fscanf( fp, "%lf", &z[i+nx*j] );
    }
}

fscanf( fp, "%d", &ix );
fscanf( fp, "%d", &iy );
a  = ( double * )malloc((size_t)( sizeof(double) * ((ix+1)*(iy+1)) ));
if( a  == NULL )
{
    printf( "no enough memory for array a\n" );
    return -1;
}
printf( "\tnx =%6d\n", nx );
printf( "\tny =%6d\n", ny );
printf( "\tix =%6d\n", ix );
printf( "\tiy =%6d\n", iy );
printf( "\n\tCoordinates \n" );
printf( "\t  x =" );
for( i=0 ; i<nx ; i++ )
{
    printf( "%8.3g ", x[i] );
}
printf( "\n\n" );
printf( "\t  y =" );
for( i=0 ; i<ny ; i++ )
{
    printf( "%8.3g ", y[i] );
}
printf( "\n\n" );
printf( "\t  z(x,y) = \n" );
for( i=0 ; i<nx ; i++ )
{
    printf( "\t     " );
    for( j=0 ; j<ny ; j++ )
    {
        printf( "%8.3g ", z[i+nx*j] );
    }
    printf( "\n" );
}

fclose( fp );

ierr = ASL_dngapl(x, nx, y, ny, z, ix, iy, a, f, wk);

printf( "\n     ** Output **\n\n" );
printf( "\tierr = %6d\n", ierr );
printf( "\n\tCoefficient of Polynomial for x,y\n\n" );
for( i=0 ; i<((ix+1)*(iy+1)) ; i++ )
{
    printf(  "\t  a[%6d]=%8.3g\n", i,a[i] );
}
printf( "\n\tf(x,y) = \n" );
for( i=0 ; i<nx ; i++ )
{
    printf( "\t     " );
    for( j=0 ; j<ny ; j++ )
```

```
            {
                printf( "%8.3g ", f[i+nx*j] );
            }
            printf( "\n" );
        }

        free( x );
        free( y );
        free( z );
        free( a );
        free( f );
        free( wk );

        return 0;
    }
```

(d) 出力結果

```
    *** ASL_dngapl ***

     ** Input **

    nx =     4
    ny =     5
    ix =     2
    iy =     2

    Coordinates
      x =      -3       -1        1        3

      y =      -2       -1        0        1        2

      z(x,y) =
                28     29.5       27     20.5       10
                -2      2.5        3     -0.5       -8
                -8     -0.5        3      2.5       -2
                10     20.5       27     29.5       28

     ** Output **

    ierr =      0

    Coefficient of Polynomial for x,y

      a[     0]=       0
      a[     1]=       0
      a[     2]=      -2
      a[     3]=       0
      a[     4]=     1.5
      a[     5]=       0
      a[     6]=       3
      a[     7]=       0
      a[     8]=       0

    f(x,y) =
                28     29.5       27     20.5       10
                -2      2.5        3     -0.5       -8
                -8     -0.5        3      2.5       -2
                10     20.5       27     29.5       28
```

5.6 チェビシェフ近似

5.6.1 ASL_dncbpo, ASL_rncbpo
チェビシェフ近似

(1) **機 能**

有限区間 $[a,b]$ 上で定義された関数 $f(x)$ をチェビシェフ近似し, $F(x) \sim [\sum_{j=0}^{\mathrm{n}} \mathrm{c}_j \mathrm{t}_j(x)] - \frac{1}{2}\mathrm{c}_0$ を満たすチェビシェフ係数 c_j を求める (isw=0 のとき). また, $f(x) \sim \sum_{k=0}^{m} y_k x^k$ であるような多項式係数 y_k を求める (isw=1 のとき).

なお, $\mathrm{c}_j = \frac{\mathrm{n}}{2} \sum_{k=1}^{\mathrm{n}} [F[\cos(\pi(k-\frac{1}{2})/\mathrm{n}) \cos(\pi j(k-\frac{1}{2})/\mathrm{n})]$ である.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dncbpo  (f, a, b, n, ceps, & aeps, c, & nc, isw, wk);
```

単精度関数:

```
ierr = ASL_rncbpo  (f, a, b, n, ceps, & aeps, c, & nc, isw, wk);
```

(3) 引数と戻り値

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	f	{D / R}	–	入 力	被チェビシェフ関数 f を定義する関数名 (注意事項 (a) 参照)
2	a	{D / R}	1	入 力	近似範囲の下端 a
3	b	{D / R}	1	入 力	近似範囲の上端 b
4	n	I	1	入 力	チェビシェフ係数の要求次数 n
5	ceps	{D / R}	1	入 力	チェビシェフ係数打切り誤差 (注意事項 (c) 参照)
6	aeps	{D∗ / R∗}	1	入 力	許容定数 (注意事項 (b) 参照)
				出 力	推定残差 (注意事項 (b) 参照)
7	c	{D∗ / R∗}	n+1	出 力	チェビシェフ係数 $c_j(j = 0, 1, \cdots, n)$ または, 多項式係数 $y_k(k$=0, 1, $\cdots, m)$
8	nc	I*	1	出 力	打切り次数 m(注意事項 (c) 参照)
9	isw	I	1	入 力	isw=0: チェビシェフ係数のみ求める isw=1: チェビシェフ係数を求め, その係数による多項式近似まで行う (注意事項 (c) 参照)
10	wk	{D∗ / R∗}	2∗(n+1)	ワーク	作業領域
11	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

(a) a < b

(b) n ≥ 0

(c) 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.	
1000	制限条件 (a) を満足しなかった.	a = b のとき: c [0] =f(a), c[i] = 0.0 (i = 1, ···, n) として返す. a > b のとき: a と b を入れ換えて処理を続ける.
3000	制限条件 (b) を満足しなかった.	処理を打ち切る.
3010	制限条件 (c) を満足しなかった.	
3500	関数と近似多項式の誤差が, 許容定数 aeps より大きい.	誤差を出力, 得られた結果を返す.

(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) 許容定数とは, 関数 $f(x)$ とそれをチェビシェフ近似した関数の予測される最大誤差であり, 推定残差とは区間 [a, b] を分割した時の, 近似関数と元の関数 $f(x)$(a $\leq x \leq$ b) の推定される最大の誤差である.

(c) ここで言う "打切り" とは, チェビシェフ近似を最適に行うために, 求めたチェビシェフ係数を評価し, 無視することが不可能な係数の限界である次数 nc(=m) を求め, nc+1 次以上の係数を切り捨てることである.

なお, 打切り誤差は級数を含む公式が特定の項または決まった様式で打ち切られるとき, 常に生ずる規則的な誤差を意味する.

推定される残差がある決まった許容定数でおさえられるときだけ, 級数が打ち切られ, その時におこるもっと乱数的な誤差は丸め誤差の一部とみなされる.

(7) 使用例

(a) 問 題

区間 $[-1, 1]$ で関数 $f(x) = x^2(x^2 - 2)\sin(x)$ が与えられたとき, これに対するチェビシェフ近似多項式の係数を求める.

(b) 入力データ

関数 dnchev

近似範囲の下端	a=− 1.0
近似範囲の上端	b= 1.0
チェビシェフ係数の要求次数	n=10
チェビシェフ係数打切り誤差	ceps=0.01
許容定数	aeps=0.0001
チェビシェフ係数による多項式近似の有無	isw=1

(c) 主プログラム

```
/*      C interface example for ASL_dncbpo */

#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 ((*x)*(*x)*((*x)*(*x)-2)*sin(*x));
}
#ifdef __cplusplus
}
#endif

int main()
{
    double a;
    double b;
    int n;
    double ceps;
    double aeps;
    double *c;
    int nc;
    int isw;
    double *wk;
    int ierr;
    int j;
    FILE *fp;

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

    printf( "    *** ASL_dncbpo ***\n" );
    printf( "\n     ** Input **\n" );
    n = 10;
    isw = 1;

    fscanf( fp, "%lf", &a );
    fscanf( fp, "%lf", &b );
    fscanf( fp, "%lf", &ceps );
    fscanf( fp, "%lf", &aeps );

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

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

```
    printf( "\ta    = %8.3g \n", a);
    printf( "\tb    = %8.3g \n", b );
    printf( "\tn    =   %6d \n", n );
    printf( "\tceps = %8.3g \n", ceps );
    printf( "\taeps = %8.3g \n", aeps );
    printf( "\tisw  =   %6d \n", isw );

    fclose( fp );

    ierr = ASL_dncbpo(f, a, b, n, ceps, &aeps, c, &nc, isw, wk);

    printf( "\n     ** Output **\n" );
    printf( "\n\tierr =   %6d\n", ierr );
    printf( "\n\tnc   =   %6d\n", nc );
    printf( "\n\taeps = %8.3g \n", aeps );

    printf( "\n     ** POLYNOMIAL COEFFICIENT **\n" );
      for( j=0 ; j<nc+1 ; j ++ )
      {
      printf( "\tc[%6d]=          %8.3g\n", j,c[j] );
      }

    free( c );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dncbpo ***

 ** Input **
a    =       -1
b    =        1
n    =       10
ceps =   0.0001
aeps =     0.01
isw  =        1

 ** Output **

ierr =        0

nc   =        7

aeps = 3.19e-05

 ** POLYNOMIAL COEFFICIENT **
c[     0]=          -6.59e-17
c[     1]=          -0.00029
c[     2]=          6.45e-16
c[     3]=                -2
c[     4]=          -1.78e-15
c[     5]=              1.32
c[     6]=          1.33e-15
c[     7]=            -0.164
```

第 6 章　スプライン関数

6.1　概 要

本章ではデータ点のスプライン関数を用いた補間や近似, および得られたスプライン関数の微分値や積分値を計算する関数について説明する.

本ライブラリでは, データ点の 3 次スプライン関数 (1 変数関数) や双 3 次スプライン関数 (2 変数関数), および B スプラインによる補間や近似を行う機能が用意されている. また, 入力データに全体のデータから見て不調和なデータがあるとき, そのデータを検出し修正を行う関数も用意されている.

通常, スプライン関数の関数ではスプライン係数を求めてから補間値, 微分値, または積分値を計算するように設計されている. また, スプライン関数を求める場合, そのレベルに応じて次のような 3 つのカテゴリをもうけて区別している.

(1) **補 間**
与えられたデータ点を完全に通るスプライン関数を作る. この場合, スプライン関数の節点位置は与えられたデータ点と一致する. 本ライブラリでは, スプライン関数の両端での条件 (端条件) の決め方に応じて幾つかの機能を提供している.

(2) **平滑化した補間**
スプライン関数の節点の横座標と与えられたデータ点の横座標を一致させたスプライン関数を作る. 得られたスプライン関数は与えられたデータ点を通るとは限らないので, データ点の補間曲線としては不適切ではあるが, 単純補間を行う場合よりもより滑らかな近似関数が得られる. データ点全体を同じ補正率で補正し, 近似したい場合に有効である. 本ライブラリでは, 補正率に相当する制御変数を入力して制御する機能や統計学的手法により最適の補正関数を自動的に得る機能を提供している.

(3) **最小二乗法補間**
スプライン関数の節点位置を調整可能なパラメータとしてスプライン関数を作る. この場合, スプライン係数は節点間ごとに最小二乗近似を行いながら求める. 目的によっては平滑化した補間の場合よりも適切なスプライン関数が得られる. 特に, データ点が非常に多い場合に, 効果的に利用できる. 本ライブラリでは, 節点位置を入力パラメータとする機能や全体の最小二乗誤差が最小となる位置を自動的に探査する機能を提供している.

6.1.1 使用上の注意

(1) データ点を近似 (または補間) する関数は無数に存在する. したがって, どのような近似関数を選ぶかによって結果として得られる値は異なるのが普通である. 近似関数としてどのような関数を選ぶかは目的に応じて使い分ける必要がある. スプライン関数はデータ点を滑らかな近似関数で補間または近似したい場合に特に有効な近似関数である. なお, スプライン関数にも色々な種類があり, 目的に応じて使い分ける必要がある.

(2) ASL_[L]d/r ではじまる関数では, スプライン係数 (1 次 ～ 3 次項) の配列 c への格納方法は

配列 c[3×(nx−1)] の格納状態

$\hat{C}_{1,1}$	c[0]	$\hat{C}_{1,2}$	c[3]		$\hat{C}_{1,\mathrm{nx}-1}$	c[(nx−2)×3]	↕ 3
$\hat{C}_{2,1}$	c[1]	$\hat{C}_{2,2}$	c[4]	⋯	$\hat{C}_{2,\mathrm{nx}-1}$	c[(nx−2)×3+1]	
$\hat{C}_{3,1}$	c[2]	$\hat{C}_{3,2}$	c[5]		$\hat{C}_{3,\mathrm{nx}-1}$	c[(nx−2)×3+2]	
← − − − − − − − nx−1 − − − − − − − →							

備 考

a. $\hat{C}_{1,j}$, $\hat{C}_{2,j}$ と $\hat{C}_{3,j}$ $(j = 1, 2, \cdots, \mathrm{nx}-1)$:スプライン係数

となっているが, ASL_[L]w/v ではじまる関数では,

配列 c[3×nx] の格納状態

$C_{1,1}$	c[0]	$C_{1,2}$	c[nx]		$C_{1,3}$	c[2×nx]	↕ nx-1
$C_{2,1}$	c[1]	$C_{2,2}$	c[nx+1]		$C_{2,3}$	c[2×nx+1]	
$C_{3,1}$	c[2]	$C_{3,2}$	c[nx+2]		$C_{3,3}$	c[2×nx+2]	
⋮	⋮	⋮	⋮		⋮	⋮	
$C_{\mathrm{nx}-1,1}$		$C_{\mathrm{nx}-1,2}$			$C_{\mathrm{nx}-1,3}$		
0.0		0.0			0.0		
← − − − − − − − 3 − − − − − − − →							

備 考

a. $C_{i,1}$, $C_{i,2}$ と $C_{i,3}$ $(i = 1, 2, \cdots, \mathrm{nx}-1)$: スプライン係数; $C_{i,k} = \hat{C}_{k,i}$ $(k = 1, 2, 3)$

となり c[nx − 1], c[2 × nx − 1], c[3 × nx − 1] には 0 が格納され, 実際のスプライン係数は
c[i − 1 + nx × (j − 1)] i = 1, ⋯, nx − 1; j = 1, ⋯, 3 に格納されている. したがって配列 c の大きさは, (nx×3) 必要となる. この変更はスプライン係数を求めるための連立 1 次方程式をベクトル計算機で高速に解くために必要である.

(3) 入力データに誤りが含まれる可能性のある場合は, 誤りデータの検出補正の関数を使用すればよい.

(4) 一般に, 端条件, 平滑化方法により得られるスプライン係数は異なる. 従って, 異なった条件, 平滑化方法での各計算を行う場合は, その条件に合ったスプライン係数を求める関数を使用する必要がある.

(5) 同一の端条件, 平滑化方法でくり返しスプライン補間値や微分値等を求める場合は, 一度スプライン係数を求める関数を使用した後, スプライン係数によって補間値や微分値等を求める関数をくり返し使用すれば効率がよい.

(6) 双 3 次スプラインの関数 6.3.1 $\left\{\begin{matrix}\mathrm{ASL\_dgisxb}\\ \mathrm{ASL\_rgisxb}\end{matrix}\right\}$, 6.3.3 $\left\{\begin{matrix}\mathrm{ASL_dgiizb}\\ \mathrm{ASL_rgiizb}\end{matrix}\right\}$では, 双 3 次スプライン係数を求めることはできない.

6.1.2 使用しているアルゴリズム

6.1.2.1 3 次非周期スプライン関数 (端条件入力)

スプライン関数は, 部分区間ごとに定義された m 次, またはそれ以下の多項式を, 関数値および $m-1$ 次以下の導関数が全区間で連続であるという条件で接続した区分的多項式である.
$m=3$ としたスプライン関数を特に 3 次スプライン関数と呼ぶ. 一般に, 非周期スプラインの 2 次微係数 M_i は, 次の連立 1 次方程式の解として得られる.

$$\begin{bmatrix} 2 & \lambda_1 & & & & & \\ \mu_2 & 2 & \lambda_2 & & & 0 & \\ & \mu_3 & \ddots & \ddots & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \lambda_{n-2} & \\ & 0 & & & \mu_{n-1} & 2 & \lambda_{n-1} \\ & & & & & \mu_n & 2 \end{bmatrix} \begin{bmatrix} M_1 \\ M_2 \\ \vdots \\ \vdots \\ \vdots \\ M_{n-1} \\ M_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ \vdots \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix} \tag{6.1}$$

ここで, 各区間の節点の X 座標を

$$-\infty < \xi_1 < \xi_2 < \cdots < \xi_n < \infty$$

各節点での関数値を $y_1, \cdots, y_n$
節点間の距離を $h_i = \xi_{i+1} - \xi_i$
とすると,

$$\lambda_i = \frac{h_i}{h_{i-1}+h_i} \quad (i = 2, \cdots, n-1)$$

$$\mu_i = 1 - \lambda_i \quad (i = 2, \cdots, n-1) \tag{6.2}$$

$$d_i = \frac{6}{h_{i-1}+h_i}\{\frac{y_{i+1}-y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}}\} \quad (i = 2, \cdots, n-1)$$

なお, $\lambda_1, \mu_n, d_1, d_n$ は, 各端条件により次のように決められる.
スプライン関数を $f(x)$ とすると,

(1) $f'(\xi_1), f'(\xi_n)$(端点での 1 次微分値) がわかっているとき,

$$\lambda_1 = 1.0, \quad \mu_n = 1.0$$

$$d_1 = \frac{6}{h_1}\left\{\frac{y_2 - y_1}{h_1} - f'(\xi_1)\right\} \tag{6.3}$$

$$d_n = \frac{6}{h_{n-1}}\left\{f'(\xi_n) - \frac{y_n - y_{n-1}}{h_{n-1}}\right\}$$

(2) $f''(\xi_1), f''(\xi_n)$(端点での 2 次微分値) がわかっているとき,

$$\begin{aligned} &\lambda_1 = \mu_n = 0.0 \\ &d_1 = 2f''(\xi_1) \\ &d_n = 2f''(\xi_n) \end{aligned} \tag{6.4}$$

端条件がわからない場合は, $f''(\xi_1) = f''(\xi_n) = 0.0$ とした, いわゆる 3 次の**自然スプライン**とする方法がある.

(3) $f'''(\xi_1), f'''(\xi_n)$(端点での 3 次微分値) がわかっているとき,

$$\begin{aligned}
&\lambda_1 = \mu_n = -2.0 \\
&d_1 = -2h_1 f'''(\xi_1) \\
&d_n = 2h_{n-1} f'''(\xi_n)
\end{aligned} \tag{6.5}$$

端条件がわからない場合は, $f'''(\xi_1) = f'''(\xi_n) = 0.0$ とする方法もある. これは **P–スプライン**と呼ばれる.
この連立 1 次方程式は, 次のようにして解くことができる.

$$\begin{aligned}
&b_i = 2 \quad (i = 1, \cdots, n) \\
&\text{(前進過程)} \\
&\left.\begin{aligned} b_i &= b_i - \frac{\mu_i \lambda_{i-1}}{b_{i-1}} \\ d_i &= d_i - \frac{\mu_i d_{i-1}}{b_{i-1}} \end{aligned}\right\} \quad (i = 2, \cdots, n) \\
&M_n = d_n - b_n \\
&\text{(後退過程)} \\
&M_i = \frac{d_i - \lambda_i M_{i+1}}{b_i} \quad (i = n-1, \cdots, 1)
\end{aligned}$$

求まった M_i よりスプライン係数 $c_{1\sim3,i}$ を求めるには,

$$\begin{cases}
c_{1,i} = \dfrac{y_{i+1} - y_i}{h_i} - \dfrac{h_i}{6}(M_{i+1} - M_i) - \dfrac{h_i M_i}{2} \\
c_{2,i} = \dfrac{M_i}{2} \\
c_{3,i} = \dfrac{M_{i+1} - M_i}{6h_i}
\end{cases} \tag{6.6}$$

6.1.2.2　3 次周期スプライン関数

周期スプラインの 2 次微係数 M_i を求める連立 1 次方程式は, 次のようになる.

$$\begin{bmatrix} 2 & \lambda_2 & \cdots & \cdots & 0 & \cdots & \mu_2 \\ \mu_3 & 2 & \lambda_3 & & & 0 & \vdots \\ \vdots & \mu_4 & \ddots & \ddots & & & 0 \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ 0 & & & \ddots & \ddots & \lambda_{n-2} & \vdots \\ \vdots & 0 & & & \mu_{n-1} & 2 & \lambda_{n-1} \\ \lambda_n & \cdots & 0 & \cdots & \cdots & \mu_n & 2 \end{bmatrix} \begin{bmatrix} M_2 \\ M_3 \\ \vdots \\ \vdots \\ \vdots \\ M_{n-1} \\ M_n \end{bmatrix} = \begin{bmatrix} d_2 \\ d_3 \\ \vdots \\ \vdots \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix} \tag{6.7}$$

$M_1 = M_n, y_1 = y_n$

この連立 1 次方程式を解くには, 次のアルゴリズムを用いる.

$$\begin{aligned} b_1 &= 2 \\ b_i &= 2 - \lambda_i \mu_{i+1} / b_{i-1} && (i = 2, \cdots, n-2) \\ p_i &= -\lambda_{i+1} / b_i && (i = 1, \cdots, n-2) \\ q_1 &= -\mu_2 / 2 \\ q_i &= -q_{i-1} \mu_{i+1} / b_{i-1} && (i = 2, \cdots, n-2) \\ r_1 &= d_2 / 2 \\ r_i &= (d_{i+1} - r_{i-1} \mu_{i+1}) / b_{i-1} && (i = 2, \cdots, n-2) \\ t_i &= p_i t_{i+1} + q_i && (t_{n-1} = 1, i = n-2, \cdots, 1) \\ v_i &= p_i v_{i+1} + r_i && (v_{n-1} = 0, i = n-2, \cdots, 1) \\ M_n &= (d_n - \lambda_n v_1 - \mu_n v_{n-2}) / (\lambda_n t_1 + \mu_n t_{n-2} + 2) \\ M_i &= t_{i-1} M_n + v_{i-1} && (i = n, \cdots, 2) \\ M_1 &= M_n \\ y_1 &= y_n \end{aligned}$$

得られた M_i よりスプライン係数を求める方法は, 非周期スプラインの場合と同様である.

6.1.2.3　3 次非周期スプライン関数 (端条件入力不要)

この場合は, 1 次微分値 $m_i = f'(\xi_1)$ を用いたアルゴリズムを使用する. この時の m_i を求める連立 1 次方程式は, 次のように表せる. (λ, μ は, 端条件入力の場合と同様.)

$$\begin{bmatrix} 2 & \mu_1 & & & & & \\ \lambda_2 & 2 & \mu_2 & & & 0 & \\ & \lambda_3 & \ddots & \ddots & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \mu_{n-3} & \\ & 0 & & & \lambda_{n-2} & 2 & \mu_{n-2} \\ & & & & & \lambda_{n-1} & 2 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ \vdots \\ \vdots \\ \vdots \\ m_{n-2} \\ m_{n-1} \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ \vdots \\ \vdots \\ c_{n-2} \\ c_{n-1} \end{bmatrix} \tag{6.8}$$

$$c_i = 3 \left\{ \lambda_i \frac{(y_i - y_{i-1})}{h_{i-1}} + \mu_i \frac{(y_{i+1} - y_i)}{h_i} \right\} \quad (i = 2, \cdots, n-1) \tag{6.9}$$

端条件としては **“not-a-knot” 条件**とする．すなわち, 点 $x_1 \sim x_3, x_{n-2} \sim x_n$ でのスプライン関数の 3 次導関数 $f'''(x)$ が連続であり, 点 x_3 および, 点 x_{n-2} では, 1 次および 2 次導関数が連続であるものとする．この場合, 端条件は次のようになる．

$$\begin{aligned}
\mu_1 &= \frac{2(h_1+h_2)}{h_2} \\
\lambda_{n-1} &= \frac{2h_{n-1}}{h_{n-1}+h_{n-2}} \\
c_1 &= \frac{2[(h_1+2(h_1+h_2))(y_2-y_1)/h_1 + h_1^2(y_3-y_2)/h_2^2]}{h_1+h_2} \\
c_{n-1} &= \frac{2[h_{n-2}^2(y_n-y_{n-1})/h_{n-1} + h_{n-1}(3h_{n-2}+2h_{n-1})(y_{n-1}-y_{n-2})/h_{n-2}]}{(h_{n-1}+h_{n-2})^2}
\end{aligned} \tag{6.10}$$

m_i を用いてスプライン係数 $c_{1\sim3,i}$ は次式から求めることができる．

$$\begin{cases}
c_{1,i} = m_i \\
c_{2,i} = \dfrac{y_{i+1}-y_i}{{h_i}^2} - \dfrac{m_i}{h_i} - c_{3,i}h_i \\
c_{3,i} = \dfrac{m_{i+1}+m_i}{{h_1}^2} - 2\dfrac{y_{i+1}-y_i}{{h_i}^3}
\end{cases} \tag{6.11}$$

6.1.2.4 制御変数指定 3 次スプライン平滑化

データの集合を, データ点を節点とした滑らかな曲線で近似することで, 次の関数を最小にする自然スプライン ($f''(\xi_1) = f''(\xi_n) = 0$) を考える．(参考文献 (6) 参照)

$$\begin{aligned}
&S_i = \textstyle\int_{\xi_1}^{\xi_n} (f''(x))^2 dx \\
&S_m = \textstyle\sum_{i=1}^{n} \left(\dfrac{f(\xi_1)-y_i}{\delta y_i} \right)^2 \\
&S_i + pS_m \to \min
\end{aligned} \tag{6.12}$$

ここで, p を指定するかわりに S_m のとりうる値 S_f を入力する制御変数の値とする．(6.12) 式の条件で p を変化させ $S_m = S_f$ になる p を求める．
δy_i として y_i の偏差の推定値を使用するなら, S_f としては, n をデータ点数として

$$n - (2n)^{1/2} \le S_f \le n + (2n)^{1/2}$$

の範囲内の値が望ましい．

3 重対角行列 Q の成分 $q_{i,j}$ を

$$q_{i-1,i} = \frac{1}{h_{i-1}}, \quad q_{i,i} = -\frac{1}{h_{i-1}} - \frac{1}{h_i}, \quad q_{i+1,i} = \frac{1}{h_i} \tag{6.13}$$

3 重対角行列 T の成分 $t_{i,j}$ を

$$t_{i,j} = \frac{2}{3}(h_{i-1}+h_i), \quad t_{i,i+1} = t_{i+1,i} = \frac{h_i}{3} \tag{6.14}$$

対角行列 D の成分を

$$d_i = \delta y_i \tag{6.15}$$

とする．スプライン係数を求めるアルゴリズムは, これらの記号を用いると次のようになる．

① $\bar{p} = 0$ より始める．

② $Q^T DQ$, $Q^T \boldsymbol{y}$, T を計算する ($\boldsymbol{y}$は y_i を成分とするベクトル).

③ 連立 1 次方程式 $(\bar{p}Q^TD^2Q+T)\boldsymbol{u}=Q^T\boldsymbol{y}$ を解き解$\boldsymbol{u}$を得る. なお, 式 $\bar{p}Q^TD^2Q+T$ のコレスキー分解を R^TR とする.

④ $\boldsymbol{v}=\frac{\boldsymbol{y}-\boldsymbol{s}}{\bar{p}}=D^2Q\boldsymbol{u}$ を計算する ($\boldsymbol{s}$: 節点でのスプライン関数値 $f(\xi_i)$ を成分とするベクトル).
また $e=\boldsymbol{v}\cdot Q\boldsymbol{u}=\dfrac{(\boldsymbol{y}-\boldsymbol{s})^2}{\bar{p}D^2}$, $S'_m=\bar{p}^2e=\sum\left(\dfrac{f(\xi_i)-y_i}{\delta y_i}\right)^2$ を求める.

⑤ もし, $S'_m>S_f$ が成立すれば, 以下のように新しく $\bar{p}$ を求めた後に②にもどる. 連立 1 次方程式 $R^T\boldsymbol{g}=Q^TD^2Q\boldsymbol{u}$ を解き解$\boldsymbol{g}$を求める.

$$\begin{aligned} f &= \boldsymbol{g}^T\boldsymbol{g} \\ h &= e-\bar{p}f \\ \bar{p} &= \bar{p}+\frac{S_f-S'_m}{(\sqrt{S_f/e}+\bar{p})h} \end{aligned}$$

⑥ もし, $S'_m\le S_f$ ならば, 以下のようにスプライン係数を計算して終了する.

$$\begin{aligned} \boldsymbol{s} &= \boldsymbol{y}-\bar{p}\boldsymbol{v} \\ c_{2,i} &= u_i \\ h_i &= x_{i+1}-x_i \\ c_{3,i} &= \frac{u_{i+1}-u_i}{3h_i} \\ c_{1,i} &= \frac{f_{i+1}-f_i}{h_i}-c_{2,i}h_i-c_{3,i}h_1^2 \end{aligned}$$

6.1.2.5 3 次スプライン自動平滑化

これは, データ集合の最適の近似曲線を得るために, **クロスバリデーション関数 (cross varidation function)** を最小とする p(前節参照) の値を得て平滑化を行う. アルゴリズムをまとめると次のようになる. (参考文献 (5) 参照)

① $T^{-1/2}$ を次のようにして求める (行列 T は (6.14) 式参照).

$$\begin{aligned} T^{1/2} &= UXU^T \quad \text{(特異値分解)} \\ &\quad \left(\begin{array}{l} U: T \text{ の固有ベクトルよりなる行列.} \\ X: T \text{ の固有値 } (\lambda_i) \text{ の平方根を対角要素とする対角行列.} \end{array}\right) \\ T^{-1/2} &= UEU^T \\ &\quad (E: \frac{1}{\sqrt{\lambda_i}}\text{を対角要素とする対角行列}) \end{aligned}$$

② $F=QT^{-1/2}$ より F を求める (行列 Q は (6.13) 式参照).

③ F を特異値分解し, $F=UWV^T$ を得る.
$\boldsymbol{s}$ (節点での関数値) $=A\boldsymbol{y}$ としたとき,

$$\begin{aligned} I-A &= Q(Q^TQ+pT)^{-1}Q^T \\ &= F(F^TF+pI)^{-1}F^T \quad \text{(重み = 1 とする)} \\ I-A &= U\begin{pmatrix} \frac{{d_1}^2}{{d_1}^2+p} & & 0 \\ & \ddots & \\ 0 & & \frac{{d_{n-2}}^2}{{d_{n-2}}^2+p} \end{pmatrix}U^T \quad d_i: W \text{ の対角要素} \end{aligned}$$

④ クロスバリデーション関数の近似関数 $V(\hat{p})$ を次のように定義する.

$$V(\hat{p}) = \frac{1}{n}\sum_{j=1}^{n-2}\left(\frac{{d_j}^2}{{d_j}^2+p}\right){z_j}^2 \Big/ \left[\frac{1}{n}\sum_{j=1}^{n-2}\left(\frac{{d_j}^2}{{d_j}^2+p}\right)\right]^2$$

ここで$\boldsymbol{z} = U^T\boldsymbol{y}$

$V(\hat{p})$ の極小値を, ダビドン法による極値探査により求める. 極小となる点の $\hat{p}$ の値より, 節点での関数値と 2 次微係数を求め, スプライン係数を計算する (前節の制御変数指定平滑化法参照). なお, ダビドン法による極値探査は, まず, 最小点をはさむ 2 点を決め, これらの点での関数値と傾斜から関数を 3 次多項式により補間し, 極小値を求める.

6.1.2.6 3 次スプライン係数 (節点位置指定最小二乗法)

入力データ (y_i) とスプライン関数値 $f(x_i)$ の間の最小二乗誤差が最小となるように, 補区間を節点 (ξ_i) により指定してスプライン関数を求める. (参考文献 (7) 参照)

最小二乗誤差としては, $S = \sqrt{\left(\sum_{i=1}^{n} w_i(f(x_i)-y_i)^2\right)}$ とし, w_i は各データ点の重みで次の値とする.

$$w_1 = \frac{x_2 - x_1}{x_n - x_1}$$

$$w_i = \frac{x_{i+1} - x_{i-1}}{x_n - x_1} \quad (i = 2, \cdots, n-1) \tag{6.16}$$

$$w_n = \frac{x_n - x_{n-1}}{x_n - x_1}$$

スプライン関数は, 同時に正規直交関数の基底 $\{\Psi_i\}_{i=1}^m$ で表されるものとする.

$$< \Psi_i, \Psi_j >= \delta_{ij}, \quad i, j = 1, \cdots, m \tag{6.17}$$

(δ はクロネッカーのデルタ, $<>$ は内積)

関数値 $u = \sum_{i=1}^{m} < u, \Psi_i > \Psi_i$, $< u, \Psi_j >$ は Ψ_i に対する係数.

3 次スプラインの基底 $\{\Phi_i\}_{i=1}^m$ を与え, これより正規直交多項式基底を得るには, 修正グラム–シュミット直交化法を用いる. この方法は次のように表せる.

$$\left.\begin{array}{l} {\Phi_i}^{(1)} = \Phi_i \\ {\Phi_i}^{(j+1)} = {\Phi_i}^{(j)} - < {\Phi_i}^{(j)}, \Psi_j > \Psi_j, \quad j = 1, \cdots, i-1 \\ \Psi_i = \dfrac{{\Phi_i}^{(i)}}{\|{\Phi_i}^{(i)}\|} \quad (\|{\Phi_i}^{(i)}\| = (< {\Phi_i}^{(i)}, {\Phi_i}^{(i)} >)^{1/2}) \end{array}\right\} (i = 1, \cdots, m) \tag{6.18}$$

このように, 直交多項式化すれば, 追加節点に対する近似関数の更新は, 新たな直交多項式基底とその係数を追加することで求まり, 追加節点での関数値や 1 次微係数は, 節点を追加する前の値に, 新たな多項式成分により求まる値を加えればよい.

以上のアルゴリズムを, 処理ステップに分けて説明する.

① 重みの計算

② 最初の節点 ξ_0 と最後の節点 ξ_m の間の 1 区間で, 3 次の最小二乗近似多項式を直交多項式を利用して求め, この基底を $\Psi_1 \sim \Psi_4$ とする. また, これに対する最小二乗誤差も求める.

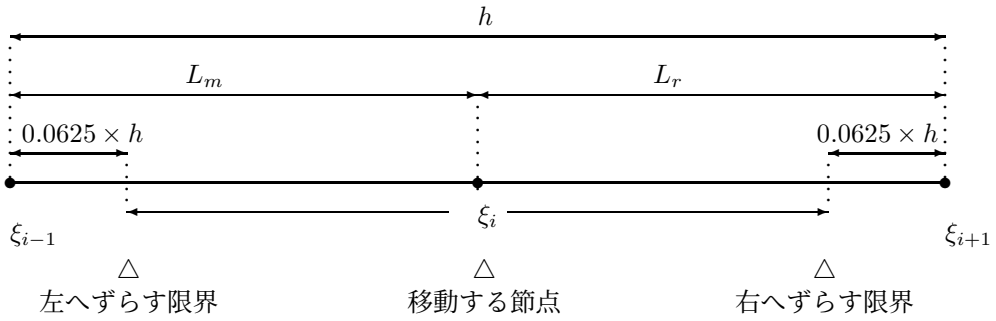
③ 次に節点を一つずつ追加しながら, その追加された節点に対応する 3 次スプライン Φ_i を計算し, 修正グラム–シュミット直交化法で正規直交多項式基底を得る. それと同時に, 追加節点での関数値と 1 次微係数値を計算し, 最小二乗誤差の更新を行う.

④ 最後に, 全ての節点の追加が終了すれば関数値と 1 次微係数値よりスプライン多項式を得る.

⑤ 節点の追加, 更新を行うときは, 多項式を正規直交多項式化して行い, 最後に再度スプライン多項式化する.

6.1.2.7 3 次スプライン係数 (節点位置自動最小二乗法)

節点の数を固定しておいて, 内部節点 ξ_i $(i = 2, 3, \cdots, n-1)$ を動かすことによって最小二乗誤差 E_S を減少させる.
アルゴリズムとしては両端の節点を固定しておき, 右側の節点から左側の節点へ順々に, 各区間ごとの最小二乗誤差 E_i が最小となる最適位置に動かしていく方法である. (参考文献 (8) 参照) ここで, 右または左へ節点を動かす大きさは, 次のようになる.



左への移動量=L_m×((前回の節点移動量)/h の全区間平均) 最初は $0.4 \times L_m$
右への移動量=L_r×((前回の節点移動量)/h の全区間平均) 最初は $0.4 \times L_r$

6.1.2.8 3 次スプライン係数による補間値

補間点に対して, $\xi_i \leq x < \xi_{i+1}$ となる区間を探す. 次に次式の補間式を計算して補間値を求める.

$$f(x) = s_i + c_{1,i}(x - \xi_i) + c_{2,i}(x - \xi_i)^2 + c_{3,i}(x - \xi_i)^3 \tag{6.19}$$

s_i は節点でのスプライン関数値 $f(\xi_i)$ で, 平滑化や最小二乗法を行わないときは, y_i に等しい.

6.1.2.9 3 次スプライン係数による微分値

補間値と同様, 微分値を計算する点を含む区間を探し, 次式より求める.

$$\begin{aligned} f'(x) &= c_{1,i} + 2c_{2,i}(x - \xi_i) + 3c_{3,i}(x - \xi_i)^2 \\ f''(x) &= 2c_{2,i} + 6c_{3,i}(x - \xi_i) \end{aligned} \tag{6.20}$$

6.1.2.10　3 次スプライン係数による積分値

積分区間を [a, b] とし, a は $\xi_i \le a < \xi_{i+1}$, b は $\xi_j \le b < \xi_{j+1}$ の区間にあるものとする.

$$\begin{aligned}\int_a^b f(x)dx &= \int_{\xi_i}^{\xi_{i+1}} f(x)dx - \int_{\xi_i}^{a} f(x)dx + \int_{\xi_{i+1}}^{\xi_j} f(x)dx + \int_{\xi_j}^{b} f(x)dx \\ &= \sum_{k=i}^{j-1}\int_{\xi_k}^{\xi_{k+1}} f(x)dx - \int_{\xi_i}^{a} f(x)dx + \int_{\xi_j}^{b} f(x)dx \end{aligned} \tag{6.21}$$

ここで,

$$\begin{aligned}\int_{\xi_k}^{\xi_{k+1}} f(x)dx &= s_i h_i + \frac{c_{1,i}}{2}h_i^2 + \frac{c_{2,i}}{3}h_i^3 + \frac{c_{3,i}}{4}h_i^4 \\ &= \frac{h_i}{2}(y_{i+1}+y_i) - \frac{h_i^3}{12}(c_{2,i+1}+c_{2,i}) \\ \int_{\xi_k}^{a} f(x)dx &= s_i(a-\xi_k) + \frac{c_{1,i}}{2}(a-\xi_k)^2 + \frac{c_{2,i}}{3}(a-\xi_k)^3 + \frac{c_{3,i}}{4}(a-\xi_k)^4\end{aligned}$$

6.1.2.11　双 3 次スプライン係数

双 3 次スプラインは, 3 次スプラインの拡張として定義される. まず X, Y 平面上で格子分割を (6.22) 式に従い行う.

$$\begin{aligned}&a < x_1 < x_2 < \cdots < x_m = b \\ &c < x_1 < y_2 < \cdots < y_m = d\end{aligned} \tag{6.22}$$

x_i, y_i 上の関数値を $z_{i,j}$ とする.

1. まず $y = y_k$ と固定し, データ点 $(x_i, z_{i,k})$ $(i = 1, 2, \cdots, m)$ と端点条件 “not-a-knot” 条件を用いて 3 次スプラインで補間する. 得られた 3 次スプライン係数を用いて点 (x_i, y_k) での微分値 $\dfrac{\partial z_{i,k}}{\partial x}$ $(i = 1, \cdots, m)$ を求める. これを $k = 1, \cdots, n$ について行う.

2. ステップ 1 と同様に $x = x_k$ と固定し, データ点 $(y_i, z_{k,j})(j = 1, \cdots, n)$ と “not-a-knot” の端点条件を用いて 3 次スプラインで補間し, 微分値 $\dfrac{\partial z_{k,i}}{\partial y}$ $(i = 1, \cdots, n, k = 1, \cdots, m)$ を求める.

3. 四隅を “not-a-knot” とする端点条件とし, ステップ 1, 2 で得られた $\dfrac{\partial z_{i,j}}{\partial x}, \dfrac{\partial z_{i,j}}{\partial y}$ $(i = 1, \cdots, m; j = 1, \cdots, n)$ をデータ点としてこれを 3 次スプラインで補間する. 得られた 3 次スプラインを用いて, 点 (x_i, y_j) での微分値 $\dfrac{\partial^2 z_{i,j}}{\partial x \partial y}(i = 1, \cdots, m; j = 1, \cdots, n)$ を求める.

なお, 上記のステップ 1. ～ 3. は, 1 次微係数を使った 3 次スプライン補間係数の計算を利用すれば次のように容易にできる.

1. $j = 1, \cdots, n$ に対して

$$\begin{aligned}&h_i z_x(i+1,j) + 2(h_i + h_{i+1})z_x(i,j) + h_{i+1}z_x(i-1,j) \\ &= 3\left\{\frac{h_i}{h_{i+1}}(z_{i+1,j} - z_{i,j}) + \frac{h_{i+1}}{h_i}(z_{i,j} - z_{i-1,j})\right\} \qquad (i = 2, \cdots, m)\end{aligned}$$

$j = 1, n$ に対して (両端部)

$$\begin{aligned}&h_i z_{xy}(i+1,j) + 2(h_i + h_{i+1})z_{xy}(i,j) + h_{i+1}z_{xy}(i-1,j) \\ &= 3\left\{\frac{h_i}{h_{i+1}}(z_y(i+1,j) - z_y(i,j)) + \frac{h_{i+1}}{h_i}(z_y(i,j) - z_y(i-1,j))\right\} \qquad (i = 2, \cdots, m)\end{aligned}$$

を解き $z_x(i,j)$ を求める.

2. $i = 1, \cdots, m$ に対して

$$k_j z_y(i, j+1) + 2(k_j + k_{j+1}) z_y(i, j) + k_{j+1} z_y(i, j+1)$$
$$= 3 \left\{ \frac{k_j}{k_{j+1}} (z_{i,j+1} - z_{i,j}) + \frac{k_{j+1}}{k_j} (z_{i,j} - z_{i,j-1}) \right\} \qquad (j = 1, \cdots, n)$$

を解き $z_y(i, j)$ を求める.

3. $i = 1, \cdots, m$ に対して

$$k_j z_{xy}(i, j+1) + 2(k_j + k_{j+1}) z_{xy}(i, j) + k_{j+1} z_{xy}(i, j+1)$$
$$= 3 \left\{ \frac{k_j}{k_{j+1}} (z_x(i, j+1) - z_x(i, j)) + \frac{k_{j+1}}{k_j} (z_x(i, j) - z_x(i, j-1)) \right\} \qquad (j = 1, \cdots, n)$$

を解き $z_{xy}(i, j)$ を求める.

6.1.2.12 双 3 次スプライン補間値

点 (x, y) における関数値は, 次式で表される.

$$f(x, y) = \sum_{m=0}^{3} \sum_{r=0}^{3} \alpha_{m,r}^{i,j} \left(\frac{x - x_i}{x_{i+1} - x_i} \right)^m \left(\frac{y - y_j}{y_{j+1} - y_j} \right)^r \tag{6.23}$$

$\alpha_{m,r}^{i,j}$ は, 次の行列演算より求まる.

$$\begin{aligned}
&\Gamma_{ij} = A(h_i) K_{ij} A(k_j)^T \\
&\Gamma_{ij} \quad = \begin{bmatrix} \alpha_{00} & \alpha_{01} & \alpha_{02} & \alpha_{03} \\ \alpha_{10} & \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{20} & \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{30} & \alpha_{31} & \alpha_{32} & \alpha_{33} \end{bmatrix} \qquad (\alpha_{mr}\text{の各要素}) \\
&A(t) = \begin{bmatrix} 1 & 0 & 0 & 0 & : \\ 0 & t & 0 & 0 & : \\ -3 & -2t & 3 & -t & : \\ 2 & t & -2 & t & : \end{bmatrix} \\
&K_{ij} = \begin{bmatrix} z(i,j) & z_y(i,j) & z(i,j+1) & z_y(i,j+1) \\ z_x(i,j) & z_{xy}(i,j) & z_x(i,j+1) & z_{xy}(i,j+1) \\ z(i+1,j) & z_y(i+1,j) & z(i+1,j+1) & z_y(i+1,j+1) \\ z_x(i+1,j) & z_{xy}(i+1,j) & z_x(i+1,j+1) & z_{xy}(i+1,j+1) \end{bmatrix}
\end{aligned} \tag{6.24}$$

ここで

$$\begin{aligned}
&x_i < x < x_{i+1}, \quad y_j < y < y_{j+1} \\
&h_i = x_{i+1} - x_i, \quad k_j = y_{j+1} - y_j \\
&z(i, j) = z_{i,j}, \quad z_x(i, j) = \frac{\partial z_{i,j}}{\partial x}, \quad z_y(i, j) = \frac{\partial z_{i,j}}{\partial y} \\
&z_{xy}(i, j) = \frac{\partial^2 z_{i,j}}{\partial x \partial y}
\end{aligned}$$

とする.
なお, 本ライブラリでは, 次の値をスプライン係数とする.

$$
\begin{aligned}
C(1,I,1,J) &= z_{i,j} = z(i,j) \\
C(2,I,1,J) &= \frac{\partial z_{i,j}}{\partial x} = z_x(i,j) \\
C(1,I,2,J) &= \frac{\partial z_{i,j}}{\partial y} = z_y(i,j) \\
C(2,I,2,J) &= \frac{\partial^2 z_{i,j}}{\partial x \partial y} = z_{xy}(i,j)
\end{aligned}
$$

従って, スプライン関数値を求めるには, $\alpha_{m,r}^{i,j}$ を使わず, 直接 (6.23) 式と (6.24) 式の行列演算を組合わせた計算を行う.

$$
\begin{aligned}
m &= 1 \\
h_x &= x_{i+1} - x_i \\
h_y &= y_{j+1} - y_j \\
U &= (x - x_i)/h_x \\
V &= (y - y_j)/h_y
\end{aligned}
$$

for $k = j \ to \ j+1$
  for $m = 1 to 2$
    $\alpha_0 = C(1,i,m,k)$
    $\alpha_1 = h_x C(2,i,m,k)$
    $\alpha_2 = 3(C(1,i+1,m,k) - \alpha_0) - h_x C(2,i+1,m,k) - 2\alpha_1$
    $\alpha_3 = 2(\alpha_0 - C(1,i+1,m,k)) + h_x C(2,i+1,m,k) + \alpha_1$
    $S_m = \alpha_0 + U(\alpha_1 + U(\alpha_2 + U\alpha_3))$
    $m = m + 1$

$$
\begin{aligned}
\alpha_0 &= S_1 \\
\alpha_1 &= h_y S_2 \\
\alpha_2 &= 3(S_3 - \alpha_0) - h_y S_4 - 2\alpha_1 \\
\alpha_3 &= 2(\alpha_0 - S_3) + h_y S_4 + \alpha_1 \\
f(x,y) &= \alpha_0 + V(\alpha_1 + V(\alpha_2 + V\alpha_3))
\end{aligned}
$$

なお, 補間値は, 次のようにして求める. まず X 軸方向にスプライン補間し, X 軸上の新しい格子点に対する補間値を求める. 次にこの補間値をもとに Y 軸方向にスプライン補間し, 新しい格子点に対する補間値を求める.

6.1.2.13 双 3 次スプライン混合偏微分値

$$
f(x,y) = \sum_{m=0}^{3}\sum_{r=0}^{3} \alpha_{m,r}^{i,j} \left(\frac{x - x_i}{x_{i+1} - x_i}\right)^m \left(\frac{y - y_j}{y_{j+1} - y_j}\right)^r \tag{6.25}
$$

$$
\frac{\partial f}{\partial x} = \sum_{m=1}^{3}\sum_{r=0}^{3} -m\alpha_{m,r}^{i,j} \frac{(x - x_i)^{m-1}}{(x_{i+1} - x_i)^m} \left(\frac{y - y_j}{y_{j+1} - y_j}\right)^r \tag{6.26}
$$

$$
\frac{\partial f}{\partial y} = \sum_{m=0}^{3}\sum_{r=1}^{3} -r\alpha_{m,r}^{i,j} \left(\frac{x - x_i}{x_{i+1} - x_i}\right)^m \frac{(y - y_j)^{r-1}}{(y_{j+1-y_j})^r} \tag{6.27}
$$

$$
\frac{\partial^2 f}{\partial x \partial y} = \sum_{m=1}^{3}\sum_{r=1}^{3} mr\alpha_{m,r}^{i,j} \frac{(x - x_i)^{m-1}}{(x_{i+1} - x_i)^m} \frac{(y - y_j)^{r-1}}{(y_{j+1} - y_j)^r} \tag{6.28}
$$

なお, この演算は補間値の場合と同様に行う.

6.1.2.14 双 3 次スプライン 2 重積分値

X 方向積分区間 [A, B], Y 方向積分区間 [C, D] の長方形区間の 2 重積分をこの区間に含まれる格子や区画の積分を行い, これらを合計することにより求める.
1 区画の積分は, 次の式より求める.

$$\int_{x_i}^{x}\int_{y_j}^{y} f(x,y)dydx = \sum_{m=0}^{3}\sum_{r=0}^{3}\alpha_{m,r}^{i,j}\frac{(x_{i+1}-x_i)(y_{j+1}-y_j)}{(m+1)(r+1)}\left(\frac{x-x_i}{x_{i+1}-x_i}\right)^{m+1}\left(\frac{y-y_j}{y_{j+1}-y_j}\right)^{r+1} \tag{6.29}$$

2 重積分値は次のようにして求める. 格子幅を半分にして各格子点上の補間値を求める. この補間値と入力データを用い, 不定間隔分割の積分に拡張したシンプソン則によりまず Y 軸方向に積分し, 次にこの積分値を X 軸方向に積分する.

6.1.2.15 平面データの補間

補間については, データ点間の距離を横座標 X 座標を縦座標としてスプライン補間し, 補間された X 座標を決める. 横座標を上記のままとし, Y 座標を縦座標としてスプライン補間し, 補間された Y 座標を決める. このようにして求められた補間値 x, y を出力する.
平滑化は, スプライン補間するときに 3 次スプライン自動平滑化関数を利用する.
ここで, 開曲線としたいときは, 端条件入力不要の方法でスプライン係数を出し, 閉曲線としたいときは周期条件の方法でスプライン係数を出すものとする.

6.1.2.16 B–スプライン関数を用いた補間 (1 次元)

データ $(x_i, y_i)(i = 0, 1, \cdots, N)$ は領域 $R : a = x_0 \leq x \leq x_N = b$ で与えられているものとする. このとき, あらかじめ決められた (内部の) 節点

$$\xi_1 \leq \xi_2 \leq \cdots \leq \xi_n \tag{6.30}$$

をもつ m 階 ($m-1$ 次) のスプライン関数

$$S(x_i) = y_i \quad (i = 0, 1, \cdots, N) \tag{6.31}$$

を用いて補間する. このとき, 一意的に補間できるためには,

$$N + 1 = m + n \tag{6.32}$$

および Schoenberg–Whitney の条件が成り立つ必要がある.
ここで, Schoenberg–Whitney の条件とは, 不等式

$$\left.\begin{array}{l} x_0 < \xi_1 < x_m, \\ x_1 < \xi_2 < x_{m+1}, \\ \cdots\cdots\cdots\cdots\cdots \\ x_{N-m} < \xi_n < x_N \end{array}\right\} \tag{6.33}$$

が満足されることをいう.
スプライン関数 $S(x)$ の基底関数として m 階 ($m-1$ 次) の B–スプラインを用いる. 必要な基底を作るために $2m$ 個の付加節点

$$\left.\begin{array}{l} \xi_{1-m} = \xi_{2-m} = \cdots = \xi_0 = a, \\ \xi_{n+1} = \xi_{n+2} = \cdots = \xi_{n+m} = b \end{array}\right\} \tag{6.34}$$

を導入する. これは区間 $[a,b]$ の両端に m 重の節点を入れることに相当する.
すると, $\xi_0 \le x \le \xi_{n+1}$ において, $S(x)$ は

$$S(x) = \sum_{i=1}^{n+m} c_i^* M_{mi}(x) = \sum_{i=1}^{n+m} c_i N_{mi}(x) \tag{6.35}$$

と表される. ここに $M_{mi}(x)$ は節点 $\xi_{i-m}, \xi_{i-m+1}, \cdots, \xi_i$ に対して定義された階数 m の B–スプラインである. また $N_{mi}(x)$ は, 正規化された B–スプラインで,

$$N_{mi}(x) = (\xi_i - \xi_{i-m}) M_{mi}(x) \tag{6.36}$$

で定義される.
B–スプライン $M_{mi}(x)$ の値は de Boor–Cox のアルゴリズムによって容易に計算できる. de Boor–Cox のアルゴリズムとは次の漸化式によって計算する算法である.

$$M_{rj}(x) = \frac{(x - \xi_{j-r}) M_{r-1,j-1}(x) + (\xi_j - x) M_{r-1,j}(x)}{\xi_j - \xi_{j-r}} \quad (r = 2, 3, \cdots, m) \tag{6.37}$$

$$M_{1j} = \begin{cases} \dfrac{1}{\xi_j - \xi_{j-1}} & (\xi_{j-1} \le x < \xi_j) \\ 0 & (\text{その他}) \end{cases} \tag{6.38}$$

(6.35) を (6.31) へ代入すると連立 1 次方程式

$$\sum_{i=1}^{n+m} c_i N_{mi}(x_j) = y_j \quad (j = 0, 1, \cdots, N) \tag{6.39}$$

を得る. これを行列表現により

$$A\boldsymbol{c} = \boldsymbol{y} \tag{6.40}$$

と表すことにする. (6.40) は, 部分選択つきのガウスの消去法を用いて, 効果的に解くことができる. その解を (6.35) へ代入すると, 補間スプライン $S(x)$ が決定される.

6.1.2.17 B–スプライン関数を用いた補間 (多次元)

(1) で述べた B–スプラインを用いる 1 次元データの補間法を 2 次元データの場合へ拡張しよう. 関数値 $f_{ij} = f(x_i, y_j)$ は矩形領域 $R : a = x_0 \le x \le x_I = b; c = y_0 \le y \le y_J = d$ の格子点 $(x_i, y_j)(i = 0, 1, \cdots, I; j = 0, 1, 2, \cdots, J)$ の上で与えられているものとする. このとき x 方向の (内部の) 節点

$$\xi_1 \le \xi_2 \le \cdots \le \xi_h \tag{6.41}$$

y 方向の (内部の) 節点

$$\zeta_1 \le \zeta_2 \le \cdots \le \zeta_k \tag{6.42}$$

をもつ 2 変数の m 階 ($m-1$ 次) および n 階 ($n-1$ 次) のスプライン関数

$$S(x_i, y_j) = f_{ij} \quad (i = 0, 1, \cdots, I; j = 0, 1, \cdots, J) \tag{6.43}$$

を用いて補間する. このとき, x 方向について

$$I + 1 = h + m, \tag{6.44}$$

$$\left.\begin{array}{l} x_0 < \xi_1 < x_m \\ x_1 < \xi_2 < x_{m+1} \\ \cdots\cdots\cdots\cdots \\ x_{I-m} < \xi_h < x_I \end{array}\right\} \tag{6.45}$$

が成り立ち, y 方向について

$$J + 1 = k + n, \tag{6.46}$$

$$\left.\begin{array}{l} y_0 < \zeta_1 < y_n \\ y_1 < \zeta_2 < y_{n+1} \\ \cdots\cdots\cdots\cdots \\ y_{J-n} < \zeta_k < y_J \end{array}\right\} \tag{6.47}$$

が成り立つと仮定する.
スプライン関数 $S(x, y)$ は一組の基底関数を用いて構成できる. この基底関数は 1 次元の基底関数のテンソル積で作ることができる. 必要な基底関数を作るために, x 方向に $2m$ 個の付加節点

$$\left.\begin{array}{l} \xi_{1-m} = \cdots = \xi_0 = a, \\ \xi_{h+1} = \cdots = \xi_{h+m} = b \end{array}\right\} \tag{6.48}$$

を導入し, y 方向に $2n$ 個の付加節点

$$\left.\begin{array}{l} \zeta_{1-n} = \cdots = \zeta_0 = c, \\ \zeta_{k+1} = \cdots = \zeta_{k+n} = d \end{array}\right\} \tag{6.49}$$

を導入する.
このとき, 節点 $\boldsymbol{\xi} = (\xi_{1-m}, \xi_{2-m}, \cdots, \xi_{h+m})$ に対する m 階 ($m-1$ 次) の B–スプライン $M_{mi}(x), (i = 1, 2, \cdots, h+m)$ と, 節点 $\boldsymbol{\zeta} = (\zeta_{1-n}, \zeta_{2-n}, \cdots, \zeta_{k+n})$ に対する n 階 ($n-1$ 次) の B–スプライン $M_{nj}(y), (j = 1, 2, \cdots, k+n)$ の積 $M_{mi}(x)M_{nj}(y)$ $(i = 1, 2, \cdots, h+m; j = 1, 2, \cdots, k+n)$ はスプライン関数 $S(x, y)$ の基底関数となる. すると, $\xi_0 \le x \le \xi_{h+1}, \zeta_0 \le y \le \zeta_{k+1}$ において $S(x, y)$ は

$$S(x, y) = \sum_{i=1}^{h+m} \sum_{j=1}^{k+n} c^*_{ij} M_{mi}(x) M_{nj}(y) = \sum_{i=1}^{h+m} \sum_{j=1}^{k+n} c_{ij} N_{mi}(x) N_{nj}(y) \tag{6.50}$$

と表される. ここに $N_{mi}(x)$, $N_{nj}(y)$ はそれぞれ正規化された m 階 ($m-1$ 次) および n 階 ($n-1$ 次) の B–スプラインで,

$$\left.\begin{array}{l} N_{mi}(x) = (\xi_i - \xi_{i-m}) M_{mi}(x) \\ N_{nj}(y) = (\zeta_j - \zeta_{j-n}) M_{nj}(y) \end{array}\right\} \tag{6.51}$$

を満たす. これらの B–スプラインの値は (1) で述べた de Boor–Cox のアルゴリズムによって容易に計算できる.
(6.50) を (6.43) へ代入すると

$$\sum_{r=1}^{h+m} \sum_{s=1}^{k+n} c_{rs} N_{mr}(x_i) N_{ns}(y_j) = f_{ij} \quad (i = 0, 1, \cdots, I; j = 0, 1, \cdots, J) \tag{6.52}$$

を得る. (6.52) は c_{rs} を未知数とする連立 1 次方程式であり, (6.44)∼(6.47) としたので一意的な解を有する. (6.52) を行列表現すると,

$$A\boldsymbol{c} = \boldsymbol{f} \tag{6.53}$$

ここに

$$\boldsymbol{c} = [c_{11}, c_{21}, \cdots, c_{h+m,1}, \cdots, c_{h+m,k+n}]^T \tag{6.54}$$

$$\boldsymbol{f} = [f_{00}, f_{10}, \cdots, f_{I0}, \cdots, f_{IJ}]^T \tag{6.55}$$

と書ける. (6.53) は部分選択つきのガウスの消去法で解くことができる. その解を (6.50) へ代入すると, 補間スプライン $S(x, y)$ が決定される.
以上で述べた 2 次元データの補間法は, 同様にして 3 次以上の多次元データの補間へと拡張できる.

6.1.2.18 B–スプラインによる平滑化 (1 次元データ)

いま, データは区間 $[a, b]$ 内において与えられ,

$$F_k = f(x_k) + e_k \quad (k = 1, 2, \cdots, N) \tag{6.56}$$

であると仮定する. ここに $f(x)$ はデータのもとにある関数 (未知関数) で, e_k は平均値 0, 分散 σ^2 の正規分布をする互いに独立な誤差である.
最小二乗法を用いてデータ (6.56) へ (6.35) をあてはめる. 節点は与えられたものとする. 残差の 2 乗和は

$$Q = \sum_{k=1}^{N} \{S(x_k) - F_k\}^2 \tag{6.57}$$

である. (6.57) をパラメータ $c_i (i = 1, 2, \cdots, n + m)$ で偏微分して 0 とおくと, 正規方程式

$$A\boldsymbol{c} = \boldsymbol{d} \tag{6.58}$$

を得る. ここに

$$\boldsymbol{c} = (c_1, c_2, \cdots, c_{n+m})^T \tag{6.59}$$

$$\boldsymbol{d} = \{ \sum_{x_k \in [\xi_{1-m}, \xi_1]} N_{m1}(x_k) F_k, \sum_{x_k \in [\xi_{2-m}, \xi_2]} N_{m2}(x_k) F_k, \quad \cdots, \sum_{x_k \in [\xi_n, \xi_{n+m}]} N_{m,n+m}(x_k) F_k \}^T \tag{6.60}$$

である. T は転置を表す.
A の i 行 j 列の要素は

$$a_{ij} = \sum N_{mi}(x_k) N_{mj}(x_k) \tag{6.61}$$

で表される. B–スプライン $N_{mi}(x)$ の値は (1) で述べたようにして容易に計算できる. また (6.58) は, 係数行列 A が帯行列であることを考慮してコレスキー法を用いると効果的に解くことができる. その解 $\boldsymbol{c}$ を (6.35) へ代入すれば近似関数 $S(x)$ が決まる.
さて, よい近似を得るためには節点の数と位置を適切に決める必要がある. ここでは, あてはめの基準 AIC を用いて, 考えられるいくつかのあてはめの中から最もよいものを選び出す (すなわち, よい節点の数と位置を求める) ことにする.
これは赤池の情報量規準とも呼ばれているもので,

$$AIC = (-2) \log_e (\text{最大尤度}) + 2(\text{パラメータ数}) \tag{6.62}$$

と定義されている. ここにパラメータ数は統計的モデル内で自由に変えられるものの数である. AIC の値を最小にするモデルが最もよいモデルであるとみなされる. AIC を用いる場合には主観的な判断はまったく必要なく, 考えられる複数個のモデルの中から (そのうちで) 最もよいモデルを自動的に決定できる.
次の回帰模型を考えよう.

$$F_k = S(x_k) + e_k \quad (k = 1, 2, \cdots, N) \tag{6.63}$$

この模型のパラメータは $S(x)$ の係数 $c_i(i=1,2,\cdots,n+m)$ と誤差の分散 σ^2 である．$S(x)$ の節点は, あてはめに先立って決められているので, パラメータの数には加えない．そこで (6.62) より,

$$AIC = N\log_e Q + 2(n+m) \tag{6.64}$$

となる．(6.64) を最小にする模型が最もよい近似関数とみなされる．したがって, 節点の数と位置をいろいろ変えてみて, AIC ができるだけ小さくなるあてはめを探し, そのあてはめが満足できるものであればよい近似が得られたものとする．

6.1.2.19 B–スプラインによる平滑化 (多次元データ)

(3) で述べた B–スプラインを用いる 1 次元の平滑化を 2 次元データの場合へ拡張しよう．データは, $x-y$ 平面上の矩形領域 $R=[a,b]\times[c,d]$ 内で与えられ,

$$F_{ij} = f(x_i,y_j) + e_r \quad (i=1,2,\cdots,I; j=1,2,\cdots,J; r=1,2,\cdots,IJ) \tag{6.65}$$

であると仮定する．ここに, $f(x,y)$ はデータのもとにある関数 (未知関数) で, e_r は平均値が 0 で分散が σ^2 の正規分布に従う誤差で互いに独立であると仮定する．標本点 (x_i,y_j) は格子点で与えられている．
最小二乗法を用いてデータ (6.65) へ (6.50) をあてはめる．各方向の節点は与えられたものとする．残差の 2 乗和は

$$Q = \sum_{r=1}^{N}\{S(x_r,y_r) - F_r\}^2 \tag{6.66}$$

である．(6.66) をパラメータ c_{ij} $(i=1,2,\cdots,h+m; j=1,2,\cdots,k+n)$ で偏微分して 0 とおくと, 正規方程式

$$A\boldsymbol{c} = \boldsymbol{d} \tag{6.67}$$

を得る．ここに

$$\boldsymbol{c} = (c_{11}, c_{21}, \cdots, c_{h+m,1}, \cdots, c_{h+m,k+n})^T \tag{6.68}$$

$$\begin{aligned}\boldsymbol{d} = \{ & \sum_{x_r\in[\xi_{1-m},\xi_1]\cap y_r\in[\zeta_{1-n},\zeta_1]} N_{m1}(x_r)N_{n1}(y_r)F_r, \\ & \sum_{x_r\in[\xi_{2-m},\xi_2]\cap y_r\in[\zeta_{1-n},\zeta_1]} N_{m2}(x_r)N_{n1}(y_r)F_r, \\ & \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots, \\ & \sum_{x_r\in[\xi_h,\xi_{h+m}]\cap y_r\in[\zeta_{1-n},\zeta_1]} N_{m,h+m}(x_r)N_{n1}(y_r)F_r, \\ & \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots, \\ & \sum_{x_r\in[\xi_h,\xi_{h+m}]\cap y_r\in[\zeta_k,\zeta_{k+n}]} N_{m,h+m}(x_r)N_{n,k+n}(y_r)F_r\}^T\end{aligned} \tag{6.69}$$

である．(6.67) は, A が帯行列であることを考慮して, コレスキー法を用いて解くことができる．
次の回帰模型を考えよう．

$$F_{ij} = f(x_i,y_j) + e_r \quad (i=1,2,\cdots,I; j=1,2,\cdots,J; r=1,2,\cdots,IJ) \tag{6.70}$$

(6.62) より, AIC は

$$AIC = N\log_e Q + 2(h+m)(k+n) \tag{6.71}$$

となる．節点の数と位置をいろいろ変えてみて, AIC ができるだけ小さくなるあてはめを探し, そのあてはめが満足できるものであれば, よい近似が得られたものとする．以上で述べた 2 次元データの平滑化は, 同様にして 3 次以上の多次元データの平滑化へと拡張できる．

6.1.3 参考文献

(1) de Boor, C. , “A Practical Guide to Splines”, Springer–Verlag, New York (1978).

(2) Ahlberg, J. , Nilson, E. and Walsh, J. , “The Theory of Splines and Their Applications”, Academic Press, New York (1967).

(3) 市田浩三, 吉本富士市, “スプライン関数とその応用”, 教育出版 (1979).

(4) Guerra, V. and Tapia, R. A. , “A Local Procedure for Error Detection and Data Smoothing”, MRC Technical Summary Report #1452, Mathematics Research Center, University of Wisconsin–Madison (1974).

(5) Craven, P. and Wahaba, G. , “Smoothing Noisy Data with Spline Functions”, Numer. Math. , Vol. 31, PP. 377–403 (1979).

(6) Reinsch, C. H. , “Smoothing by Spline Functions”, Numer. Math. , Vol. 10, PP. 177–183 (1967).

(7) de Boor, C. and Rice, J. R. , “Least Squares Cubic Spline Approximation I - Fixed Knots”, Computer Sciences Department TR20, Purdue Univ. , (1968).

(8) de Boor, C. and Rice, J. R. , “Least Squares Cubic Spline Approximation II - Variable Knots”, Computer Sciences Department TR21, Purdue Univ. , (1968).

(9) 桜井 明編, “スプライン関数入門”, 東京電機大学出版局 (1981).

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

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

6.2 3 次スプライン (曲線補間)

6.2.1 ASL_dgispc, ASL_rgispc
補間値と 3 次スプライン係数

(1) **機 能**

端条件入力不要とした “not-a-knot” 条件の 3 次スプライン係数を求め, 指定点での補間値を計算する. なお, 節点位置は標本点と一致させる.

(2) **使用法**

倍精度関数:

ierr = ASL_dgispc (x, y, n, xl, fl, m, c);

単精度関数:

ierr = ASL_rgispc (x, y, n, xl, fl, m, c);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	x	{D∗ / R∗}	n	入 力	標本点 $(x[i-1], y[i-1]), i = 1, \cdots, n$ の横座標値 $x[i-1]$ $(x[i-1] < x[i], i \neq n)$
2	y	{D∗ / R∗}	n	入 力	標本点の縦座標値, あるいは, 関数値 $y[i-1]$
				出 力	3 次スプライン係数の 0 次項 (入力値と同じ)
3	n	I	1	入 力	標本点の数
4	xl	{D∗ / R∗}	m	入 力	補間値を計算する点の横座標値
5	fl	{D∗ / R∗}	m	出 力	$xl[i-1]$ での 3 次スプライン補間値
6	m	I	1	入 力	補間点の数
7	c	{D∗ / R∗}	$3\times(n-1)$	出 力	3 次スプライン係数の k 次項 (k=1, 2, 3): $c[(k-1)+3\times(j-1)]$ 横座標値 $t(x[j-1] \le t < x[j])$ における 3 次スプライン関数の値 $f(t)$ は $f(t) = ((c[2+3\times(j-1)]\times d + c[1+3\times(j-1)])\times d + c[3\times(j-1)])\times d + y[j-1]$ ただし, $d = t - x[j-1]$
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) $n \ge 2$

(b) $x[0] < x[1] < \cdots < x[n-1]$ (昇順)

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	xl[i − 1] が補間区間の範囲外であった (xl[i − 1] < x[0] or xl[i − 1] > x[n − 1]).	端点での 3 次スプライン係数を利用し, 補外した値を出力する.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	

(6) **注意事項**

(a) 補間値, 微分値, または積分値をさらに続けて求めたい場合は, この関数を呼び出した後, 6.2.18 $\left\{\begin{matrix}\text{ASL\_dgiscx}\\ \text{ASL\_rgiscx}\end{matrix}\right\}$, 6.2.19 $\left\{\begin{matrix}\text{ASL\_dgidcy}\\ \text{ASL\_rgidcy}\end{matrix}\right\}$ または 6.2.20 $\left\{\begin{matrix}\text{ASL\_dgiicz}\\ \text{ASL\_rgiicz}\end{matrix}\right\}$ をそれぞれ呼び出せばよい. この時, 配列 x, y, c および変数 n の内容はそのまま後続の関数の対応する引数の入力とする. ただし, 微分値を求めたい場合は, 配列 y の内容を渡す必要がない. このようにすれば, 3 次スプライン係数の計算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.

(7) **使用例**

(a) 問 題

$$y_i = x_i e^{-4.0x_i}$$

の式より $\{x_i\} = \{0.0, 0.1, 0.23, 0.34, 0.47, 0.59, 0.73, 0.92, 1.0\}$ における各点の y_i $(i = 1, 2, \cdots, 9)$ をとり出し, これらを標本点として 3 次スプライン関数で補間する. さらに $xl_j = 0.1 \times j (j = 1, 2, \cdots, 10)$ の等間隔点での補間値を求める.

(b) 入力データ

x[i − 1]=x_i, y[i − 1]=y_i (i = 1, ⋯ , n),
xl[j − 1]=xl_j (j = 1, ⋯ , m),
n = 9, m = 10

(c) 主プログラム

```
/*      C interface example for ASL_dgispc */

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

int main()
{
    double *x;
    double *y;
    int nx;
    double *u;
    double *s;
    int m;
    double *c;
    int ierr;
    int i,j;
    FILE *fp;

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

    printf( "    *** ASL_dgispc ***\n" );
    printf( "\n     ** Input **\n\n" );
    nx=9;
    m=10;
```

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

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


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

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


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


printf( "\tn = %6d\tm = %6d\n", nx, m );
for( i=0 ; i<nx ; i++ )
{
    fscanf( fp, "%lf", &x[i] );
}
for( i=0 ; i<m ; i++ )
{
    fscanf( fp, "%lf", &u[i] );
}
for( i=0 ; i<nx ; i++ )
{
    fscanf( fp, "%lf", &y[i] );
}

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

printf( "\n\tSpecified Points\n\n" );
printf( "\t     j    xl[j]\n");
for( j=0 ; j<m ; j++ )
{
    printf( "\t%6d %8.3g\n", j,u[j] );
}

fclose( fp );

ierr = ASL_dgispc(x, y, nx, u, s, m, c);

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

printf( "\n" );
for( j=0 ; j<m ; j++ )
{
    printf( "\tfl[%6d]=%8.3g\n", j,s[j] );
}

printf( "\n" );
printf( "\tc[i,j]\n" );
printf( "\t               i=0      i=1      i=2\n" );
for( j=0 ; j<nx-1 ; j++ )
{
    printf( "\t j=%6d",j );
    for( i=0 ; i<3 ; i++ )
    {
        printf( " %8.3g", c[i+3*j] );
    }
    printf( "\n" );
}

free( x );
free( y );
free( u );
```

```
    free( s );
    free( c );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dgispc ***

 ** Input **

n =      9  m =     10

Coordinates (x,y)

     i     x[i]     y[i]
     0        0        0
     1      0.1    0.067
     2     0.23   0.0917
     3     0.34   0.0873
     4     0.47   0.0717
     5     0.59   0.0557
     6     0.73   0.0394
     7     0.92   0.0232
     8        1   0.0183

Specified Points

     j    xl[j]
     0      0.1
     1      0.2
     2      0.3
     3      0.4
     4      0.5
     5      0.6
     6      0.7
     7      0.8
     8      0.9
     9        1

 ** Output **

ierr =      0

fl[     0]=   0.067
fl[     1]=  0.0901
fl[     2]=  0.0904
fl[     3]=  0.0808
fl[     4]=  0.0676
fl[     5]=  0.0544
fl[     6]=  0.0426
fl[     7]=  0.0326
fl[     8]=  0.0246
fl[     9]=  0.0183

c[i,j]
                i=0      i=1      i=2
 j=     0    0.963    -3.29     3.66
 j=     1    0.414     -2.2     3.66
 j=     2   0.0281   -0.769     1.36
 j=     3  -0.0917    -0.32    0.783
 j=     4   -0.135  -0.0145    0.245
 j=     5   -0.128   0.0736   0.0659
 j=     6   -0.104    0.101  -0.0269
 j=     7   -0.068    0.086  -0.0269
```

6.2.2 ASL_dgissc, ASL_rgissc
平滑化した補間値と 3 次スプライン係数

(1) **機 能**

最適の平滑化 3 次スプライン係数を自動的に求め, 指定点での補間値を計算する. なお, 節点の横座標は標本点の横座標と一致させる.

(2) **使用法**

倍精度関数:

ierr = ASL_dgissc (x, yd, n, xl, fl, m, y, c, isw, wk);

単精度関数:

ierr = ASL_rgissc (x, yd, n, xl, fl, m, y, c, isw, wk);

(3) 引数と戻り値

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	x	{D∗ / R∗}	n	入 力	標本点 $(x[i-1], yd[i-1]), i = 1, \cdots, n$ の横座標値 $x[i-1]$ $(x[i-1] < x[i], i \neq n)$
2	yd	{D∗ / R∗}	n	入 力	標本点の縦座標値, あるいは, 関数値 $yd[i-1]$
3	n	I	1	入 力	標本点の数
4	xl	{D∗ / R∗}	m	入 力	補間値を計算する点の横座標値
5	fl	{D∗ / R∗}	m	出 力	$xl[i-1]$ での平滑化 3 次スプライン補間値
6	m	I	1	入 力	補間点の数
7	y	{D∗ / R∗}	n	出 力	3 次スプライン係数の 0 次項
8	c	{D∗ / R∗}	$3\times(n-1)$	出 力	3 次スプライン係数の k 次項 (k=1, 2, 3): $c[(k-1)+3\times(j-1)]$ 横座標値 $t(x[j-1] \le t < x[j])$ における 3 次スプライン関数の値 $f(t)$ は $f(t) = ((c[2+3\times(j-1)]\times d + c[1+3\times(j-1)])\times d + c[3\times(j-1)])\times d + y[j-1]$ ただし, $d = t - x[j-1]$
9	isw	I	1	入 力	処理スイッチ isw=1: 横座標値の間隔が等間隔とは限らないとき, 選択する. ただし, 標本点が無相関に近いときは, n の値を 20 程度までと制限すること. isw=2: 横座標値の間隔が等間隔であるとき, 選択する.
10	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 **大きさ**: $n\times(2\times n+4)$ (isw=1 のとき) $6\times n$ (isw=2 のとき)
11	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

(a) $n \ge 4$

(b) $x[0] < x[1] < \cdots < x[n-1]$ (昇順)

(c) $isw = 2$ のときは, 横座標値が等間隔であること

(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.	
1000	xl[i − 1] が補間区間の範囲外であった (xl[i − 1] < x[0] or xl[i − 1] > x[n − 1]).	端点での 3 次スプライン係数を利用し, 補外した値を出力する.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	
4000	クロスバリデーションの最小値が見つからない (データが無相関である).	

(6) **注意事項**

(a) 補間値, 微分値, または積分値をさらに続けて求めたい場合は, この関数を使用した後, 6.2.18 $\left\{\begin{matrix}\text{ASL\_dgiscx}\\ \text{ASL\_rgiscx}\end{matrix}\right\}$, 6.2.19 $\left\{\begin{matrix}\text{ASL\_dgidcy}\\ \text{ASL\_rgidcy}\end{matrix}\right\}$ または 6.2.20 $\left\{\begin{matrix}\text{ASL_dgiicz}\\ \text{ASL_rgiicz}\end{matrix}\right\}$をそれぞれ呼び出せばよい. この時, 配列 x, y, c および変数 n の内容はそのまま後続の関数の対応する引数の入力とする. ただし, 微分値を求めたい場合は, 配列 y の内容を渡す必要がない. このようにすれば, 3 次スプライン係数の計算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.

(7) **使用例**

(a) 問 題

$$y_i = \sin(3\pi x_i/2) + e_i (e_i : [-0.2, 0.2] \text{ の間の一様乱数})$$

の式より $x_i = (i-1)/24$ $(i = 1, 2, \cdots, 25)$ における各点の y_i をとり出し, これらを標本点として平滑化した 3 次スプライン関数で近似する. さらに $xl_j = 0.1 \times (j-1) (j = 1, 2, \cdots, 10)$ の等間隔点での平滑化 3 次スプライン補間値を求める.

(b) 入力データ
x[i − 1]=x_i, yd[i − 1]=y_i (i = 1, ⋯, n),
xl[j − 1]=xl_j (j = 1, ⋯, m),
n=25, isw=2, m=10

(c) 主プログラム

```
/*      C interface example for ASL_dgissc */

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

int main()
{
    double *x;
    double *f;
    int nx;
    double *u;
    double *s;
    int m;
    double *y;
    double *c;
    int isw;
    double *wk;
    int ierr;
    int i,j;
    FILE *fp;

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

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

printf( "    *** ASL_dgissc ***\n" );
printf( "\n     ** Input **\n\n" );
nx=25;
m=10;
isw=2;

c = ( double * )malloc((size_t)( sizeof(double) * (3*(nx-1)) ));
if( c == NULL )
{
    printf( "no enough memory for array c\n" );
    return -1;
}

wk = ( double * )malloc((size_t)( sizeof(double) * (6*nx) ));
if( wk == NULL )
{
    printf( "no enough memory for array wk\n" );
    return -1;
}

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

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

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

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

y = ( double * )malloc((size_t)( sizeof(double) * nx ));
if( y == NULL )
{
    printf( "no enough memory for array y\n" );
    return -1;
}
printf( "\tn   = %6d\n", nx  );
printf( "\tm   = %6d\n", m   );
printf( "\tisw = %6d\n", isw );
for( i=0 ; i<nx ; i++ )
{
    fscanf( fp, "%lf", &x[i] );
}
for( i=0 ; i<nx ; i++ )
{
    fscanf( fp, "%lf", &f[i] );
}
for( i=0 ; i<m ; i++ )
{
    fscanf( fp, "%lf", &u[i] );
}

printf( "\n\tCoordinates (x,y)\n\n" );
printf( "\t     i     x[i]    yd[i]\n");
for( i=0 ; i<nx ; i++ )
{
    printf( "\t%6d %8.3g %8.3g\n", i,x[i],f[i] );
}

printf( "\n" );
printf( "\tSpecified Points\n\n" );
printf( "\t     j    xl[j]\n");
for( j=0 ; j<m ; j++ )
{
    printf( "\t%6d %8.3g\n", j,u[j] );
}

fclose( fp );

ierr = ASL_dgissc(x, f, nx, u, s, m, y, c, isw, wk);

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

```
    printf( "\n" );
    for( j=0 ; j<m ; j++ )
    {
        printf( "\tfl[%6d]=%8.3g\n", j,s[j] );
    }

    printf( "\n" );
    for( j=0 ; j<nx ; j++ )
    {
        printf( "\ty[%6d]=%8.3g\n", j,y[j] );
    }

    printf( "\n" );
    printf( "\tc[i,j]\n\n" );
    printf( "\t               i=0      i=1      i=2\n" );
    for( j=0 ; j<nx-1 ; j++ )
    {
        printf( "\t j=%6d",j );
        for( i=0 ; i<3 ; i++ )
        {
            printf( " %8.3g", c[i+3*j] );
        }
        printf( "\n" );
    }

    free( x );
    free( f );
    free( u );
    free( s );
    free( y );
    free( c );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dgissc ***

 ** Input **

n   =     25
m   =     10
isw =      2

Coordinates (x,y)

     i     x[i]    yd[i]
     0        0   0.0481
     1   0.0416    0.147
     2   0.0833    0.442
     3    0.125    0.579
     4    0.167    0.641
     5    0.208     0.74
     6     0.25    0.726
     7    0.292     1.09
     8    0.333    0.804
     9    0.375     1.02
    10    0.417    0.851
    11    0.458        1
    12      0.5    0.517
    13    0.542    0.692
    14    0.583    0.348
    15    0.625    0.385
    16    0.667   -0.102
    17    0.708   -0.135
    18     0.75   -0.339
    19    0.792   -0.571
    20    0.833   -0.525
    21    0.875   -0.841
    22    0.917   -0.795
    23    0.958    -1.08
    24        1    -1.18

Specified Points

     j    xl[j]
     0        0
     1      0.1
     2      0.2
     3      0.3
     4      0.4
     5      0.5
     6      0.6
     7      0.7
     8      0.8
     9      0.9

 ** Output **

ierr =      0

fl[     0]=   0.095
fl[     1]=   0.447
fl[     2]=   0.732
fl[     3]=   0.901
```

```
fl[     4]=   0.896
fl[     5]=   0.695
fl[     6]=   0.341
fl[     7]= -0.0861
fl[     8]=   -0.49
fl[     9]=  -0.849

y[     0]=   0.095
y[     1]=   0.245
y[     2]=   0.391
y[     3]=   0.527
y[     4]=   0.648
y[     5]=   0.751
y[     6]=   0.834
y[     7]=   0.893
y[     8]=    0.92
y[     9]=   0.915
y[    10]=   0.876
y[    11]=   0.802
y[    12]=   0.695
y[    13]=   0.563
y[    14]=   0.408
y[    15]=   0.237
y[    16]=  0.0578
y[    17]=  -0.121
y[    18]=  -0.294
y[    19]=  -0.458
y[    20]=  -0.613
y[    21]=  -0.761
y[    22]=  -0.907
y[    23]=   -1.05
y[    24]=    -1.2

c[i,j]

              i=0       i=1      i=2
 j=     0      3.6 2.13e-14     -5.9
 j=     1     3.57   -0.737    -18.1
 j=     2     3.41       -3    -11.7
 j=     3      3.1    -4.47    -5.13
 j=     4      2.7    -5.11    -5.98
 j=     5     2.25    -5.86    -7.46
 j=     6     1.72    -6.79    -21.1
 j=     7     1.05    -9.42     3.88
 j=     8     0.28    -8.94    -10.6
 j=     9    -0.52    -10.3     2.62
 j=    10    -1.36    -9.94    -0.59
 j=    11    -2.19      -10     24.5
 j=    12     -2.9    -6.95      2.1
 j=    13    -3.47    -6.69     18.3
 j=    14    -3.93     -4.4     10.8
 j=    15    -4.24    -3.05     29.4
 j=    16    -4.34    0.628     9.28
 j=    17    -4.24     1.79     7.52
 j=    18    -4.05     2.73     1.89
 j=    19    -3.81     2.96    -12.3
 j=    20    -3.63     1.42    -1.37
 j=    21    -3.52     1.25    -11.3
 j=    22    -3.47   -0.161     2.67
 j=    23    -3.47    0.173    -1.39
```

6.2.3 ASL_dgismc, ASL_rgismc
最小二乗補間値と 3 次スプライン係数

(1) **機 能**

最小二乗近似 3 次スプライン係数を求め, 指定点での補間値を計算する. また, 最適節点位置も求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dgismc  (x, yd, n, xk, nxk, & itmx, xl, fl, m, & s, y, c, iwk, wk1, wk2);
```

単精度関数:

```
ierr = ASL_rgismc  (x, yd, n, xk, nxk, & itmx, xl, fl, m, & s, y, c, iwk, wk1, wk2);
```

(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∗}	n	入 力	標本点 $(x[i-1], yd[i-1]), i = 1, \cdots, n$ の横座標値 $x[i-1]$ $(x[i-1] < x[i], i \neq n)$
2	yd	{D∗ R∗}	n	入 力	標本点の縦座標値, あるいは, 関数値 $yd[i-1]$
3	n	I	1	入 力	標本点の数
4	xk	{D∗ R∗}	nxk	入 力	節点の位置の初期推定値 $xk[i-1] < xk[i], i = 1, \cdots, nxk-1$ $xk[0] \leq x[0]$ $xk[nxk-1] \geq x[n-1]$
				出 力	最適節点位置
5	nxk	I	1	入 力	節点の数
6	itmx	I*	1	入 力	最大反復回数 (15 回程度が適当)
				出 力	実際の反復回数
7	xl	{D∗ R∗}	m	入 力	補間値を計算する点の横座標値
8	fl	{D∗ R∗}	m	出 力	$xl[i-1]$ での最小二乗近似 3 次スプライン補間値
9	m	I	1	入 力	補間点の数
10	s	{D∗ R∗}	1	出 力	3 次スプライン近似の最小二乗誤差
11	y	{D∗ R∗}	nxk	出 力	3 次スプライン係数の 0 次項
12	c	{D∗ R∗}	内容参照	出 力	3 次スプライン係数の k 次項 (k=1, 2, 3): $c[(k-1)+3\times(j-1)]$ 横座標値 $t(xk[j-1] \leq t < xk[j])$ における 3 次スプライン関数の値 $f(t)$ は $f(t) = ((c[2+3\times(j-1)]\times d + c[1+3\times(j-1)])\times d + c[3\times(j-1)])\times d + y[j-1]$ ただし, $d = t - xk[j-1]$ **大きさ:** $3\times(nxk-1)$
13	iwk	I*	75	ワーク	作業領域
14	wk1	{D∗ R∗}	内容参照	ワーク	作業領域 **大きさ:** $n \times (nxk+6)$
15	wk2	{D∗ R∗}	3811	ワーク	作業領域
16	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $n \geq 2, nxk \leq 28, itmx \leq 20$

(b) $x[0] < x[1] < \cdots < x[n-1]$ (昇順)

(c) 両端節点の範囲内に標本点が分布している.

(d) $xk[0] < xk[1] < \cdots < xk[nxk-1]$ (昇順)

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	$xl[i-1]$ が補間区間の範囲外であった ($xl[i-1] < x[0]$ or $xl[i-1] > x[n-1]$).	端点での 3 次スプライン係数を利用し, 補外した値を出力する.
1500	最小二乗誤差 (s) の極小が得られないまま, itmx 回の反復回数で計算が終了した (データが無相関に近い).	そのときの 3 次スプライン係数, 最小二乗誤差により補間する.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	
3030	制限条件 (d) を満足しなかった.	

(6) **注意事項**

(a) 節点位置の初期推定値の値によっては, 異なった 3 次スプライン係数が得られる場合や収束状況が変化する場合がある.

(b) y[0], y[nxk − 1] の値は, 通常, 3 次スプライン係数を使って補外した値となる.

(c) 補間値, 微分値, または積分値をさらに続けて求めたい場合は, この関数を使用した後, 6.2.18 $\left\{\begin{matrix}\text{ASL\_dgiscx}\\ \text{ASL\_rgiscx}\end{matrix}\right\}$, 6.2.19 $\left\{\begin{matrix}\text{ASL\_dgidcy}\\ \text{ASL\_rgidcy}\end{matrix}\right\}$ または 6.2.20 $\left\{\begin{matrix}\text{ASL\_dgiicz}\\ \text{ASL\_rgiicz}\end{matrix}\right\}$ をそれぞれ呼び出せばよい. この時, 配列 xk, y, c および変数 nxk の内容はそのまま後続の関数の対応する引数の入力とする. ただし, 微分値を求めたい場合は, 配列 y の内容を渡す必要がない. このようにすれば, 3 次スプライン係数の計算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.

(7) 使用例

(a) 問 題

$$y_i = \begin{cases} 1.0 - x_i & (0.0 \le x_i \le 0.5) \\ x_i & (0.5 < x_i \le 1.0) \\ 2.0 - x_i & (1.0 < x_i \le 2.0) \end{cases}$$

の式より $x_i = 0.1 \times (i-1)$ $(i = 1, 2, \cdots, 21)$ における各点の y_i をとり出し，これらを標本点として 4 つの節点 $\{\xi_j\} = \{0.0, 0.33, 1.33, 2.0\}$ を使って最小二乗近似 3 次スプライン補間を行う．さらに $xl_j = 0.25 \times k (k = 1, 2, \cdots, 7)$ の等間隔点での 3 次スプライン関数の値を求める．

(b) 入力データ

x[i − 1]=x_i, yd[i − 1]=y_i (i = 1, · · · , n),
xk[j − 1]=ξ_j (j = 1, · · · , nxk),
xl[j − 1]=xl_j (k = 1, · · · , m),
n = 21, nxk = 4, m = 7, itmx = 15

(c) 主プログラム

```
/*      C interface example for ASL_dgismc */

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

int main()
{
    double *x;
    double *f;
    int nx;
    double *xk;
    int nxk;
    int  itmx;
    double *u;
    double *s;
    int m;
    double er;
    double *y;
    double *c;
    int iwk[75];
    double *wk1;
    double wk2[3811];
    int ierr;
    int i,j,k,nwk;
    FILE *fp;

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

    printf( "    *** ASL_dgismc ***\n" );
    printf( "\n     ** Input **\n\n" );
    nx=21;
    nxk=4;
    m=7;
    itmx=15;

    c = ( double * )malloc((size_t)( sizeof(double) * (3*(nxk-1)) ));
    if( c == NULL )
    {
        printf( "no enough memory for array c\n" );
        return -1;
    }

    nwk=nx*(nxk+6);
    wk1 = ( double * )malloc((size_t)( sizeof(double) * nwk ));
    if( wk1 == NULL )
    {
        printf( "no enough memory for array wk1\n" );
        return -1;
    }

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

    f = ( double * )malloc((size_t)( sizeof(double) * nx ));
    if( f == NULL )
```

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

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

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

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

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

printf( "\tn   = %6d\n", nx  );
printf( "\tnxk = %6d\n", nxk );
printf( "\tm   = %6d\n", m   );
printf( "\titmx= %6d\n", itmx);
for( i=0 ; i<nxk ; i++ )
{
    fscanf( fp, "%lf", &xk[i] );
}
for( i=0 ; i<m ; i++ )
{
    fscanf( fp, "%lf", &u[i] );
}
for( i=0 ; i<nx ; i++ )
{
    fscanf( fp, "%lf", &x[i] );
}
for( i=0 ; i<nx ; i++ )
{
    fscanf( fp, "%lf", &f[i] );
}

printf( "\n\tCoordinates (x,y)\n\n" );
printf( "\t     i     x[i]    yd[i]\n");
for( i=0 ; i<nx ; i++ )
{
    printf( "\t%6d %8.3g %8.3g\n", i,x[i],f[i] );
}

printf( "\n" );
printf( "\tSpecified Points\n\n" );
printf( "\t     j    xl[j]\n");
for( j=0 ; j<m ; j++ )
{
    printf( "\t%6d %8.3g\n", j,u[j] );
}

printf( "\n" );
printf( "\tLocations of Knots\n\n" );
printf( "\t     k    xk[k]\n");
for( k=0 ; k<nxk ; k++ )
{
    printf( "\t%6d %8.3g\n", k,xk[k] );
}

fclose( fp );

ierr = ASL_dgismc(x, f, nx, xk, nxk, &itmx, u, s, m, &er, y, c, iwk, wk1, wk2);

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

for( j=0 ; j<m ; j++ )
{
    printf( "\tfl[%6d]=%8.3g\n", j,s[j] );
}
printf( "\n" );

for( j=0 ; j<nxk ; j++ )
{
    printf( "\ty[%6d]=%8.3g\n", j,y[j] );
}
```

```
    printf( "\n" );
    printf( "\tc[i,j]\n" );
    printf( "\t               i=0      i=1      i=2\n" );
    for( j=0 ; j<nxk-1 ; j++ )
    {
        printf( "\t j=%6d",j );
        for( i=0 ; i<3 ; i++ )
        {
            printf( " %8.3g", c[i+3*j] );
        }
        printf( "\n" );
    }
    printf( "\n" );
    printf( "\tOptimal Location of Knots\n\n" );
    for( k=0 ; k<nxk ; k++ )
    {
        printf( "\txk[%6d]= %8.3g\n", k,xk[k] );
    }
    printf( "\n" );
    printf( "\tNumber of Iterations\n\n" );
    printf( "\t  itmx=%6d\n",itmx );

    printf( "\n" );
    printf( "\tLeast Squares error\n\n" );
    printf( "\t  s= %8.3g\n", er );

    free( x );
    free( f );
    free( xk );
    free( u );
    free( s );
    free( y );
    free( c );
    free( wk1 );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dgismc ***

 ** Input **

n   =     21
nxk =      4
m   =      7
itmx=     15

Coordinates (x,y)

     i     x[i]    yd[i]
     0        0        1
     1      0.1      0.9
     2      0.2      0.8
     3      0.3      0.7
     4      0.4      0.6
     5      0.5      0.5
     6      0.6      0.6
     7      0.7      0.7
     8      0.8      0.8
     9      0.9      0.9
    10        1        1
    11      1.1      0.9
    12      1.2      0.8
    13      1.3      0.7
    14      1.4      0.6
    15      1.5      0.5
    16      1.6      0.4
    17      1.7      0.3
    18      1.8      0.2
    19      1.9      0.1
    20        2        0

Specified Points

     j    xl[j]
     0     0.25
     1      0.5
     2     0.75
     3        1
     4     1.25
     5      1.5
     6     1.75

Locations of Knots

     k    xk[k]
     0        0
     1     0.33
     2     1.33
     3        2

 ** Output **

ierr =      0
```

```
fl[     0]=   0.743
fl[     1]=   0.538
fl[     2]=   0.773
fl[     3]=   0.916
fl[     4]=   0.786
fl[     5]=    0.51
fl[     6]=   0.221

y[     0]=   0.984
y[     1]=   0.558
y[     2]=   0.801
y[     3]=  0.0485

c[i,j]
                i=0      i=1      i=2
 j=     0   -0.345    -3.86     5.53
 j=     1    0.648      5.6    -14.3
 j=     2     1.15    -3.14     1.39

Optimal Location of Knots

xk[     0]=        0
xk[     1]=     0.57
xk[     2]=    0.774
xk[     3]=        2

Number of Iterations

  itmx=     3

Least Squares error

  s=   0.0289
```

6.2.4 ASL_dgidpc, ASL_rgidpc
微分値と 3 次スプライン係数

(1) **機 能**

端条件入力不要とした “not-a-knot” 条件の 3 次スプライン係数を求め, 指定点での 1 階, 2 階微分値を計算する. なお, 節点位置は標本点と一致させる.

(2) **使用法**

倍精度関数:

ierr = ASL_dgidpc (x, y, n, xl, dl, ddl, m, c);

単精度関数:

ierr = ASL_rgidpc (x, y, n, xl, dl, ddl, m, c);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	x	{D∗ R∗}	n	入 力	標本点 $(x[i-1], y[i-1]), i = 1, \cdots, n$ の横座標値 $x[i-1]$ $(x[i-1] < x[i], i \neq n)$
2	y	{D∗ R∗}	n	入 力	標本点の縦座標値, あるいは, 関数値 $y[i-1]$
				出 力	3 次スプライン係数の 0 次項 (0 次項) (入力値と同じ)
3	n	I	1	入 力	標本点の数
4	xl	{D∗ R∗}	m	入 力	3 次スプライン関数の微分値を計算する点の横座標値
5	dl	{D∗ R∗}	m	出 力	xl[i − 1] での 3 次スプライン関数の 1 階微分値 $dl[i-1] = (3.0 \times c[2+3\times(j-1)] \times d + 2.0 \times c[1+3\times(j-1)]) \times d + c[3\times(j-1)]$ ただし, $x[j-1] \leq xl[i-1] < x[j]$ $d = xl[i-1] - x[j-1]$
6	ddl	{D∗ R∗}	m	出 力	xl[i − 1] での 3 次スプライン関数の 2 階微分値 $ddl[i-1] = 6.0 \times c[2+3\times(j-1)] \times d + 2.0 \times c[1+3\times(j-1)]$ ただし, $x[j-1] \leq xl[i-1] < x[j]$ $d = xl[i-1] - x[j-1]$
7	m	I	1	入 力	微分値を計算する点の数
8	c	{D∗ R∗}	3×(n − 1)	出 力	3 次スプライン係数の k 次項 (k=1, 2, 3): $c[(k-1)+3\times(j-1)]$ 横座標値 $t(x[j-1] \leq t < x[j])$ における 3 次スプライン関数の値 f(t) は $f(t) = ((c[2+3\times(j-1)] \times d + c[1+3\times(j-1)]) \times d + c[3\times(j-1)]) \times d + y[j-1]$ ただし, $d = t - x[j-1]$
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) $n \geq 2$

(b) $x[0] < x[1] < \cdots < x[n-1]$ (昇順)

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	xl[i − 1] が補間区間の範囲外であった (xl[i − 1] < x[0] or xl[i − 1] > x[n − 1]).	端点での 3 次スプライン係数を利用し, 補外した値を出力する.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	

(6) **注意事項**

(a) 補間値, 微分値, または積分値をさらに続けて求めたい場合は, この関数を使用した後, 6.2.18 $\left\{\begin{matrix}\text{ASL\_dgiscx}\\ \text{ASL\_rgiscx}\end{matrix}\right\}$, 6.2.19 $\left\{\begin{matrix}\text{ASL\_dgidcy}\\ \text{ASL\_rgidcy}\end{matrix}\right\}$ または 6.2.20 $\left\{\begin{matrix}\text{ASL\_dgiicz}\\ \text{ASL\_rgiicz}\end{matrix}\right\}$ をそれぞれ呼び出せばよい. この時, 配列 x, y, c および変数 n の内容はそのまま後続の関数の対応する引数の入力とする. ただし, 微分値を求めたい場合は, 配列 y の内容を渡す必要がない. このようにすれば, 3 次スプライン係数の計算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.

(7) **使用例**

(a) 問 題

$y_i = x_i e^{-40x_i}$ の式より $\{x_i\} = \{0.0, 0.1, 0.23, 0.34, 0.47, 0.59, 0.73, 0.92, 1.0\}$ における各点の y_i $(i = 1, 2, \cdots, 9)$ をとり出し, これらを標本点として 3 次スプライン関数で補間する. さらに $xl_j = 0.1 \times (j-1) (j = 1, 2, \cdots, 10)$ の等間隔点での 3 次スプライン関数の 1 階, 2 階微分値を求める.

(b) 入力データ

x[i − 1]=x_i, y[i − 1]=y_i (i = 1, ⋯, n), xl[j − 1]=xl_j (j = 1, ⋯, m), n = 9, m = 10

(c) 主プログラム

```
/*      C interface example for ASL_dgidpc */

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

int main()
{
    double *x;
    double *y;
    int nx;
    double *u;
    double *ds;
    double *dds;
    int m;
    double *c;
    int ierr;
    int i,j;
    FILE *fp;

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

    printf( "    *** ASL_dgidpc ***\n" );
    printf( "\n     ** Input **\n\n" );
    nx=9;
```

```
m=10;

c = ( double * )malloc((size_t)( sizeof(double) * (3*(nx-1)) ));
if( c == NULL )
{
    printf( "no enough memory for array c\n" );
    return -1;
}

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

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

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

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

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

printf( "\tn   = %6d\n", nx  );
printf( "\tm   = %6d\n", m   );
for( i=0 ; i<nx ; i++ )
{
    fscanf( fp, "%lf", &x[i] );
}
for( i=0 ; i<m ; i++ )
{
    fscanf( fp, "%lf", &u[i] );
}
for( i=0 ; i<nx ; i++ )
{
    fscanf( fp, "%lf", &y[i] );
}

printf( "\n" );
printf( "\tCoordinates (X,Y)\n\n" );
printf( "\t     i               x[i]               y[i]\n");
for( i=0 ; i<nx ; i++ )
{
    printf( "\t%6d           %8.3g           %8.3g\n", i,x[i],y[i] );
}

printf( "\n" );
printf( "\tSpecified Points\n\n" );
printf( "\t     j              xl[j]\n");
for( j=0 ; j<m ; j++ )
{
    printf( "\t%6d           %8.3g\n", j,u[j] );
}

fclose( fp );

ierr = ASL_dgidpc(x, y, nx, u, ds, dds, m, c);

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

printf( "\tThe Value of The First Derivative\n\n" );
for( j=0 ; j<m ; j++ )
{
    printf( "\t   dl[%6d]=%8.3g\n", j,ds[j] );
}
printf( "\n" );

printf( "\tThe Value of The Second Derivative\n\n" );
for( j=0 ; j<m ; j++ )
{
    printf( "\t  ddl[%6d]=%8.3g\n", j,dds[j] );
}
printf( "\n" );
```

```
    printf( "\tSpline Coefficients\n\n" );
    printf( "\t  c[i,j]\n" );
    printf( "\t\t\t       i=0               i=1               i=2\n" );
    for( j=0 ; j<nx-1 ; j++ )
    {
        printf( "\tj=%6d",j );
        for( i=0 ; i<3 ; i++ )
        {
            printf( "          %8.3g", c[i+3*j] );
        }
        printf( "\n" );
    }
    printf( "\n" );

    free( x );
    free( y );
    free( u );
    free( ds );
    free( dds );
    free( c );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dgidpc ***

 ** Input **

n   =      9
m   =     10

Coordinates (X,Y)

     i                x[i]               y[i]
     0                   0                  0
     1                 0.1              0.067
     2                0.23             0.0917
     3                0.34             0.0873
     4                0.47             0.0717
     5                0.59             0.0557
     6                0.73             0.0394
     7                0.92             0.0232
     8                   1             0.0183

Specified Points

     j               xl[j]
     0                 0.1
     1                 0.2
     2                 0.3
     3                 0.4
     4                 0.5
     5                 0.6
     6                 0.7
     7                 0.8
     8                 0.9
     9                   1

 ** Output **

ierr =      0

The Value of The First Derivative

   dl[     0]=   0.414
   dl[     1]=  0.0842
   dl[     2]= -0.0595
   dl[     3]=  -0.122
   dl[     4]=  -0.135
   dl[     5]=  -0.127
   dl[     6]=  -0.109
   dl[     7]= -0.0898
   dl[     8]= -0.0714
   dl[     9]= -0.0547

The Value of The Second Derivative

  ddl[     0]=   -4.39
  ddl[     1]=    -2.2
  ddl[     2]=  -0.966
  ddl[     3]=  -0.358
  ddl[     4]=   0.015
  ddl[     5]=   0.151
  ddl[     6]=   0.191
  ddl[     7]=   0.191
  ddl[     8]=   0.175
  ddl[     9]=   0.159

Spline Coefficients

  c[i,j]
                 i=0               i=1               i=2
j=     0             0.963             -3.29              3.66
j=     1             0.414              -2.2              3.66
j=     2            0.0281            -0.769              1.36
j=     3           -0.0917             -0.32             0.783
```

```
j=     4          -0.135          -0.0145           0.245
j=     5          -0.128           0.0736          0.0659
j=     6          -0.104            0.101         -0.0269
j=     7          -0.068            0.086         -0.0269
```

6.2.5 ASL_dgidsc, ASL_rgidsc
平滑化した微分値と 3 次スプライン係数

(1) **機 能**

最適な平滑化 3 次スプライン係数を自動的に求め, 指定点での 1 階, 2 階微分値を求める. なお, 節点の横座標は標本点の横座標と一致させる.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dgidsc  (x, yd, n, xl, dl, ddl, m, y, c, isw, wk);
```

単精度関数:

```
ierr = ASL_rgidsc  (x, yd, n, xl, dl, ddl, m, y, c, isw, 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	x	{D∗ / R∗}	n	入 力	標本点 $(x[i-1], yd[i-1]), i = 1, \cdots, n$ の横座標値 $x[i-1]$ $(x[i-1] < x[i], i \neq n)$
2	yd	{D∗ / R∗}	n	入 力	標本点の縦座標値, あるいは, 関数値 $yd[i-1]$
3	n	I	1	入 力	標本点の数
4	xl	{D∗ / R∗}	m	入 力	3 次スプライン関数の微分値を計算する点の横座標値
5	dl	{D∗ / R∗}	m	出 力	$xl[i-1]$ での 3 次スプライン関数の 1 階微分値 $dl[i-1] = (3.0 \times c[2+3\times(j-1)] \times d + 2.0 \times c[1+3\times(j-1)]) \times d + c[3\times(j-1)]$ ただし, $x[j-1] \leq xl[i-1] < x[j]$ $d = xl[i-1] - x[j-1]$
6	ddl	{D∗ / R∗}	m	出 力	$xl[i-1]$ での 3 次スプライン関数の 2 階微分値 $ddl[i-1] = 6.0 \times c[2+3\times(j-1)] \times d + 2.0 \times c[1+3\times(j-1)]$ ただし, $x[j-1] \leq xl[i-1] < x[j]$ $d = xl[i-1] - x[j-1]$
7	m	I	1	入 力	微分値を計算する点の数
8	y	{D∗ / R∗}	n	出 力	3 次スプライン係数の 0 次項

項番	引数と戻り値	型	大きさ	入出力	内　　容
9	c	$\left\{\begin{matrix}\text{D}* \\ \text{R}*\end{matrix}\right\}$	3×(n − 1)	出 力	3 次スプライン係数の k 次項 (k=1, 2, 3): c[(k − 1) + 3 × (j − 1)] 横座標値 t(x[j − 1] ≤ t < x[j]) における 3 次スプライン関数の値 f(t) は f(t) = ((c[2 + 3 × (j − 1)] × d + c[1 + 3 × (j − 1)]) × d + c[3 × (j − 1)]) × d + y[j − 1] ただし, d = t − x[j − 1]
10	isw	I	1	入 力	処理スイッチ isw=1: 横座標値の間隔が等間隔とは限らないとき, 選択する. ただし, 標本点が無相関に近いときは, n の値を 20 程度までと制限すること. isw=2: 横座標値の間隔が等間隔であるとき, 選択する.
11	wk	$\left\{\begin{matrix}\text{D}* \\ \text{R}*\end{matrix}\right\}$	内容参照	ワーク	作業領域 **大きさ**: n × (2 × n + 4) (isw=1 のとき) 6 × n(isw=2 のとき)
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 ≥ 4

(b) x[0] < x[1] < ⋯ < x[n − 1] (昇順)

(c) isw = 2 のときは, 横座標値が等間隔であること

(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.	
1000	xl[i − 1] が補間区間の範囲外であった (xl[i − 1] < x[0] or xl[i − 1] > x[n − 1]).	端点での 3 次スプライン係数を利用し, 補外した値を出力する.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	
4000	クロスバリデーションの最小値が見つからない (データが無相関である).	

(6) **注意事項**

(a) 補間値, 微分値, または積分値をさらに続けて求めたい場合は, この関数を使用した後, 6.2.18 $\left\{\begin{matrix}\text{ASL\_dgiscx} \\ \text{ASL\_rgiscx}\end{matrix}\right\}$, 6.2.19 $\left\{\begin{matrix}\text{ASL\_dgidcy} \\ \text{ASL\_rgidcy}\end{matrix}\right\}$ または 6.2.20 $\left\{\begin{matrix}\text{ASL\_dgiicz} \\ \text{ASL\_rgiicz}\end{matrix}\right\}$ をそれぞれ呼び出せばよい. この時, 配列 x, y, c および変数 n の内容はそのまま後続の関数の対応する引数の入力とする. ただし, 微分値を求めたい場合は, 配

列 y の内容を渡す必要がない. このようにすれば, 3 次スプライン係数の計算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.

(7) 使用例

(a) 問 題

$y_i = \sin(3\pi x_i/2) + e_i$ (e_i : $[-0.2, 0.2]$ の間の一様乱数) の式より $x_i = (i-1)/24$ $(i = 1, 2, \cdots, 25)$ における各点の y_i をとり出し, これらを標本点として平滑化した 3 次スプライン関数で近似する. さらに $xl_j = 0.1 \times (j-1)(j = 1, 2, \cdots, 10)$ の等間隔点での 3 次スプライン関数の 1 階, 2 階微分値を求める.

(b) 入力データ

x[i − 1]=x_i, yd[i − 1]=y_i (i = 1, ⋯ , n),

xl[j − 1]=xl_j (j = 1, ⋯ , m),

n=25, isw=2, m=10

(c) 主プログラム

```
/*      C interface example for ASL_dgidsc */

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

int main()
{
    double *x;
    double *f;
    int nx;
    double *u;
    double *ds;
    double *dds;
    int m;
    double *y;
    double *c;
    int isw;
    double *wk;
    int ierr;
    int i,j;
    FILE *fp;

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

    printf( "    *** ASL_dgidsc ***\n" );
    printf( "\n     ** Input **\n\n" );
    nx=25;
    m=10;
    isw=2;

    c = ( double * )malloc((size_t)( sizeof(double) * (3*(nx-1)) ));
    if( c == NULL )
    {
        printf( "no enough memory for array c\n" );
        return -1;
    }

    wk = ( double * )malloc((size_t)( sizeof(double) * (6*nx) ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

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

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

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

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

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

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

printf( "\tn   = %6d\n", nx  );
printf( "\tm   = %6d\n", m   );
printf( "\tisw = %6d\n", isw );
for( i=0 ; i<nx ; i++ )
{
    fscanf( fp, "%lf", &x[i] );
}
for( i=0 ; i<nx ; i++ )
{
    fscanf( fp, "%lf", &f[i] );
}
for( i=0 ; i<m ; i++ )
{
    fscanf( fp, "%lf", &u[i] );
}

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

printf( "\n" );
printf( "\tSpecified Points\n\n" );
printf( "\t     j               xl[j]\n");
for( j=0 ; j<m ; j++ )
{
    printf( "\t%6d           %8.3g\n", j,u[j] );
}

fclose( fp );

ierr = ASL_dgidsc(x, f, nx, u, ds, dds, m, y, c, isw, wk);

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

printf( "\tThe Value of The First Derivative\n\n" );
for( j=0 ; j<m ; j++ )
{
    printf( "\t   dl[%6d]=%8.3g\n", j,ds[j] );
}
printf( "\n" );

printf( "\tThe Value of The Second Derivative\n\n" );
for( j=0 ; j<m ; j++ )
{
    printf( "\t  ddl[%6d]=%8.3g\n", j,dds[j] );
}
printf( "\n" );

printf( "\tSpline Coefficients\n\n" );
for( j=0 ; j<nx ; j++ )
{
    printf( "\t    y[%6d]=%8.3g\n", j,y[j] );
}
printf( "\n" );

printf( "\tc[i,j]\n\n" );
printf( "\t\t\t       i=0               i=1               i=2\n" );
for( j=0 ; j<nx-1 ; j++ )
{
    printf( "\tj=%6d",j );
    for( i=0 ; i<3 ; i++ )
    {
        printf( "          %8.3g", c[i+3*j] );
    }
    printf( "\n" );
}
```

```
    printf( "\n" );

    free( c );
    free( wk );
    free( x );
    free( f );
    free( u );
    free( ds );
    free( dds );
    free( y );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dgidsc ***

 ** Input **

n   =     25
m   =     10
isw =      2

Coordinates (x,yd)

     i                x[i]              yd[i]
     0                   0            0.0481
     1              0.0416             0.147
     2              0.0833             0.442
     3               0.125             0.579
     4               0.167             0.641
     5               0.208              0.74
     6                0.25             0.726
     7               0.292              1.09
     8               0.333             0.804
     9               0.375              1.02
    10               0.417             0.851
    11               0.458                 1
    12                 0.5             0.517
    13               0.542             0.692
    14               0.583             0.348
    15               0.625             0.385
    16               0.667            -0.102
    17               0.708            -0.135
    18                0.75            -0.339
    19               0.792            -0.571
    20               0.833            -0.525
    21               0.875            -0.841
    22               0.917            -0.795
    23               0.958             -1.08
    24                   1             -1.18

Specified Points

     j               xl[j]
     0                   0
     1                 0.1
     2                 0.2
     3                 0.3
     4                 0.4
     5                 0.5
     6                 0.6
     7                 0.7
     8                 0.8
     9                 0.9

 ** Output **

ierr =      0

The Value of The First Derivative

   dl[     0]=     3.6
   dl[     1]=     3.3
   dl[     2]=    2.34
   dl[     3]=   0.888
   dl[     4]=   -1.03
   dl[     5]=    -2.9
   dl[     6]=   -4.07
   dl[     7]=   -4.27
   dl[     8]=   -3.77
   dl[     9]=   -3.48

The Value of The Second Derivative

  ddl[     0]=4.26e-14
  ddl[     1]=   -7.18
  ddl[     2]=   -11.4
  ddl[     3]=   -18.6
  ddl[     4]=   -20.1
  ddl[     5]=   -13.9
  ddl[     6]=   -7.72
  ddl[     7]=    3.12
  ddl[     8]=    5.31
  ddl[     9]=   0.809

Spline Coefficients

    y[     0]=   0.095
```

```
    y[     1]=   0.245
    y[     2]=   0.391
    y[     3]=   0.527
    y[     4]=   0.648
    y[     5]=   0.751
    y[     6]=   0.834
    y[     7]=   0.893
    y[     8]=    0.92
    y[     9]=   0.915
    y[    10]=   0.876
    y[    11]=   0.802
    y[    12]=   0.695
    y[    13]=   0.563
    y[    14]=   0.408
    y[    15]=   0.237
    y[    16]=  0.0578
    y[    17]=  -0.121
    y[    18]=  -0.294
    y[    19]=  -0.458
    y[    20]=  -0.613
    y[    21]=  -0.761
    y[    22]=  -0.907
    y[    23]=   -1.05
    y[    24]=    -1.2

c[i,j]

                i=0              i=1              i=2
j=     0          3.6         2.13e-14           -5.9
j=     1         3.57           -0.737          -18.1
j=     2         3.41               -3          -11.7
j=     3          3.1            -4.47          -5.13
j=     4          2.7            -5.11          -5.98
j=     5         2.25            -5.86          -7.46
j=     6         1.72            -6.79          -21.1
j=     7         1.05            -9.42           3.88
j=     8         0.28            -8.94          -10.6
j=     9        -0.52            -10.3           2.62
j=    10        -1.36            -9.94          -0.59
j=    11        -2.19              -10           24.5
j=    12         -2.9            -6.95            2.1
j=    13        -3.47            -6.69           18.3
j=    14        -3.93             -4.4           10.8
j=    15        -4.24            -3.05           29.4
j=    16        -4.34            0.628           9.28
j=    17        -4.24             1.79           7.52
j=    18        -4.05             2.73           1.89
j=    19        -3.81             2.96          -12.3
j=    20        -3.63             1.42          -1.37
j=    21        -3.52             1.25          -11.3
j=    22        -3.47           -0.161           2.67
j=    23        -3.47            0.173          -1.39
```

6.2.6 ASL_dgidmc, ASL_rgidmc
最小二乗微分値と 3 次スプライン係数

(1) **機 能**

最小二乗近似 3 次スプライン係数を求め, 指定点での 1 階, 2 階微分値を計算する. また, 最適節点位置も求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dgidmc (x, yd, n, xk, nxk, & itmx, xl, dl, ddl, m, & s, y, c, iwk, wk1, wk2);

単精度関数:

ierr = ASL_rgidmc (x, yd, n, xk, nxk, & itmx, xl, dl, ddl, m, & s, y, c, iwk, wk1, wk2);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	x	{D∗ / R∗}	n	入 力	標本点 $(x[i-1], yd[i-1]), i = 1, \cdots, n$ の横座標値 $x[i-1]$ $(x[i-1] < x[i], i \neq n)$
2	yd	{D∗ / R∗}	n	入 力	標本点の縦座標値, あるいは, 関数値 $yd[i-1]$
3	n	I	1	入 力	標本点の数
4	xk	{D∗ / R∗}	nxk	入 力	節点の位置の初期推定値 $xk[i-1] < xk[i], i = 1, \cdots, nxk-1$ $xk[0] \leq x[0]$ $xk[nxk-1] \geq x[n-1]$
				出 力	最適節点位置
5	nxk	I	1	入 力	節点の数
6	itmx	I*	1	入 力	最大反復回数 (15 回程度が適当)
				出 力	実際の反復回数
7	xl	{D∗ / R∗}	m	入 力	3 次スプライン関数の微分値を計算する点の横座標値
8	dl	{D∗ / R∗}	m	出 力	xl[i − 1] での 3 次スプライン関数の 1 階微分値 $dl[i-1] = (3.0 \times c[2+3\times(j-1)] \times d + 2.0 \times c[1+3\times(j-1)]) \times d + c[3\times(j-1)]$ ただし, $xk[j-1] \leq xl[i-1] < xk[j]$ $d = xl[i-1] - xk[j-1]$
9	ddl	{D∗ / R∗}	m	出 力	xl[i − 1] での 3 次スプライン関数の 2 階微分値 $ddl[i-1] = 6.0 \times c[2+3\times(j-1)] \times d + 2.0 \times c[1+3\times(j-1)]$ ただし, $xk[j-1] \leq xl[i-1] < xk[j]$ $d = xl[i-1] - xk[j-1]$
10	m	I	1	入 力	微分値を計算する点の数

項番	引数と戻り値	型	大きさ	入出力	内　　容
11	s	$\left\{\begin{matrix}\text{D*}\\ \text{R*}\end{matrix}\right\}$	1	出 力	3 次スプライン近似の最小二乗誤差
12	y	$\left\{\begin{matrix}\text{D*}\\ \text{R*}\end{matrix}\right\}$	nxk	出 力	3 次スプライン係数の 0 次項
13	c	$\left\{\begin{matrix}\text{D*}\\ \text{R*}\end{matrix}\right\}$	内容参照	出 力	3 次スプライン係数の k 次項 (k=1, 2, 3): c[(k − 1) + 3 × (j − 1)] 横座標値 t(xk[j − 1] ≤ t < xk[j]) における 3 次スプライン関数の値 f(t) は f(t) = ((c[2 + 3 × (j − 1)] × d+c[1+3×(j−1)])×d+c[3×(j−1)])×d+y[j−1] ただし, d = t − xk[j − 1] **大きさ:** 3×(nxk − 1)
14	iwk	I*	75	ワーク	作業領域
15	wk1	$\left\{\begin{matrix}\text{D*}\\ \text{R*}\end{matrix}\right\}$	内容参照	ワーク	作業領域 **大きさ:** n × (nxk + 6)
16	wk2	$\left\{\begin{matrix}\text{D*}\\ \text{R*}\end{matrix}\right\}$	3811	ワーク	作業領域
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) $\text{n} \geq 2, \text{nxk} \leq 28, \text{itmx} \leq 20$

(b) $\text{x[0]} < \text{x[1]} < \cdots < \text{x[n} - 1\text{]}$ (昇順)

(c) 両端節点の範囲内に標本点が分布している.

(d) $\text{xk[0]} < \text{xk[1]} < \cdots < \text{xk[nxk} - 1\text{]}$ (昇順)

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	xl[i − 1] が補間区間の範囲外であった (xl[i − 1] < x[0] or xl[i − 1] > x[n − 1]).	端点での 3 次スプライン係数を利用し, 補外した値を出力する.
1500	最小二乗誤差 (s) の極小が得られないまま, itmx 回の反復回数で計算が終了した (データが無相関に近い).	そのときの 3 次スプライン係数, 最小二乗誤差により補間する.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	
3030	制限条件 (d) を満足しなかった.	

(6) **注意事項**

(a) 節点位置の初期推定値の値によっては, 異なった 3 次スプライン係数が得られる場合や収束状況が変化する場合がある.

(b) y[0], y[nxk − 1] の値は, 通常, 3 次スプライン係数を使って補外した値となる.

(c) 補間値, 微分値, または積分値をさらに続けて求めたい場合は, この関数を使用した後, 6.2.18 $\left\{\begin{matrix}\text{ASL\_dgiscx}\\ \text{ASL\_rgiscx}\end{matrix}\right\}$, 6.2.19 $\left\{\begin{matrix}\text{ASL\_dgidcy}\\ \text{ASL\_rgidcy}\end{matrix}\right\}$ または 6.2.20 $\left\{\begin{matrix}\text{ASL_dgiicz}\\ \text{ASL_rgiicz}\end{matrix}\right\}$をそれぞれ呼び出せばよい. この時, 配列 xk, y, c および変数 nxk の内容はそのまま後続の関数の対応する引数の入力とする. ただし, 微分値を求めたい場合は, 配列 y の内容を渡す必要がない. このようにすれば, 3 次スプライン係数の計算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.

(7) **使用例**

(a) 問 題

$$y_i = \begin{cases} 1.0 - x_i & (0.0 \le x_i \le 0.5) \\ x_i & (0.5 < x_i \le 1.0) \\ 2.0 - x_i & (1.0 < x_i \le 2.0) \end{cases}$$

の式より $x_i = 0.1 \times (i-1)$ $(i = 1, 2, \cdots, 21)$ における各点の y_i をとり出し, これらを標本点として 4 つの節点 $\{\xi_j\} = \{0.0, 0.33, 1.33, 2.0\}$ を使って最小二乗近似 3 次スプライン補間を行う. さらに $xl_j = 0.25 \times k (k = 1, 2, \cdots, 7)$ の等間隔点での 3 次スプライン関数の 1 階, 2 階微分値を求める.

(b) 入力データ

x[i − 1]=x_i, yd[i − 1]=y_i (i = 1, ⋯ , n),
xk[j − 1]=ξ_j (j = 1, ⋯ , nxk),
xl[j − 1]=xl_j (k = 1, ⋯ , m),
n = 21, nxk = 4, m = 7, itmx = 15

(c) 主プログラム

```
/*      C interface example for ASL_dgidmc */

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

int main()
{
    double *x;
    double *f;
    int nx;
    double *xk;
    int nxk;
    int  itmx;
    double *u;
    double *ds;
    double *dds;
    int m;
    double  er;
    double *y;
    double *c;
    int iwk[75];
    double *wk1;
    double wk2[3811];
    int ierr;
    int i,j,k,nwk;
    FILE *fp;

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

    printf( "    *** ASL_dgidmc ***\n" );
    printf( "\n     ** Input **\n\n" );
```

```
nx=21;
nxk=4;
m=7;
itmx=15;

c = ( double * )malloc((size_t)( sizeof(double) * (3*(nxk-1)) ));
if( c == NULL )
{
    printf( "no enough memory for array c\n" );
    return -1;
}

nwk=nx*(nxk+6);
wk1 = ( double * )malloc((size_t)( sizeof(double) * nwk ));
if( wk1 == NULL )
{
    printf( "no enough memory for array wk1\n" );
    return -1;
}

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

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

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

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

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

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

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

printf( "\tn   = %6d\n", nx  );
printf( "\tnxk = %6d\n", nxk );
printf( "\tm   = %6d\n", m   );
printf( "\titmx= %6d\n", itmx);
for( i=0 ; i<nxk ; i++ )
{
    fscanf( fp, "%lf", &xk[i] );
}
for( i=0 ; i<m ; i++ )
{
    fscanf( fp, "%lf", &u[i] );
}
for( i=0 ; i<nx ; i++ )
{
    fscanf( fp, "%lf", &x[i] );
}
for( i=0 ; i<nx ; i++ )
{
    fscanf( fp, "%lf", &f[i] );
}

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

printf( "\n" );
```

```
    printf( "\tSpecified Points\n\n" );
    printf( "\t     j     xl[j]\n");
    for( j=0 ; j<m ; j++ )
    {
        printf( "\t%6d %8.3g\n", j,u[j] );
    }

    printf( "\n" );
    printf( "\tLocations of Knots\n\n" );
    printf( "\t     k     xk[k]\n");
    for( k=0 ; k<nxk ; k++ )
    {
        printf( "\t%6d %8.3g\n", k,xk[k] );
    }

    fclose( fp );

    ierr = ASL_dgidmc(x, f, nx, xk, nxk, &itmx, u, ds, dds, m, &er, y, c, iwk, wk1, wk2);

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

    printf( "\n\tThe value of the first derivative\n\n" );
    for( j=0 ; j<m ; j++ )
    {
        printf( "\t   dl[%6d]=%8.3g\n", j,ds[j] );
    }
    printf( "\n" );

    printf( "\n\tThe value of the second derivative\n\n" );
    for( j=0 ; j<m ; j++ )
    {
        printf( "\t  ddl[%6d]=%8.3g\n", j,dds[j] );
    }
    printf( "\n" );

    for( j=0 ; j<nxk ; j++ )
    {
        printf( "\t    y[%6d]=%8.3g\n", j,y[j] );
    }

    printf( "\n" );
    printf( "\tc[i,j]\n" );
    printf( "\t\t        i=0               i=1               i=2\n" );
    for( j=0 ; j<nxk-1 ; j++ )
    {
        printf( "\tj=%6d    ",j );
        for( i=0 ; i<3 ; i++ )
        {
            printf( "%8.3g          ", c[i+3*j] );
        }
        printf( "\n" );
    }
    printf( "\n" );
    printf( "\tOptimal Location of Knots\n\n" );
    for( k=0 ; k<nxk ; k++ )
    {
        printf( "\t   xk[%6d]= %8.3g\n", k,xk[k] );
    }
    printf( "\n" );
    printf( "\tNumber of Iterations\n" );
    printf( "\t  itmx=%6d\n",itmx );

    printf( "\n" );
    printf( "\tLeast Squares error\n" );
    printf( "\t  s= %8.3g\n", er );
    free( x );
    free( f );
    free( xk );
    free( u );
    free( ds );
    free( dds );
    free( y );
    free( c );
    free( wk1 );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dgidmc ***

 ** Input **

n   =     21
nxk =      4
m   =      7
itmx=     15

Coordinates (x,yd)

     i      x[i]    yd[i]
```

```
     0        0        1
     1      0.1      0.9
     2      0.2      0.8
     3      0.3      0.7
     4      0.4      0.6
     5      0.5      0.5
     6      0.6      0.6
     7      0.7      0.7
     8      0.8      0.8
     9      0.9      0.9
    10        1        1
    11      1.1      0.9
    12      1.2      0.8
    13      1.3      0.7
    14      1.4      0.6
    15      1.5      0.5
    16      1.6      0.4
    17      1.7      0.3
    18      1.8      0.2
    19      1.9      0.1
    20        2        0

Specified Points

     j     xl[j]
     0     0.25
     1      0.5
     2     0.75
     3        1
     4     1.25
     5      1.5
     6     1.75

Locations of Knots

     k     xk[k]
     0        0
     1     0.33
     2     1.33
     3        2

 ** Output **

ierr =      0

The value of the first derivative

   dl[     0]=   -1.24
   dl[     1]= -0.0544
   dl[     2]=    1.27
   dl[     3]= -0.0606
   dl[     4]=  -0.899
   dl[     5]=   -1.22
   dl[     6]=   -1.01


The value of the second derivative

  ddl[     0]=   0.582
  ddl[     1]=    8.88
  ddl[     2]=   -4.26
  ddl[     3]=   -4.39
  ddl[     4]=   -2.31
  ddl[     5]=  -0.225
  ddl[     6]=    1.86

    y[     0]=   0.984
    y[     1]=   0.558
    y[     2]=   0.801
    y[     3]=  0.0485

c[i,j]
            i=0               i=1               i=2
j=     0      -0.345             -3.86              5.53
j=     1       0.648              5.6             -14.3
j=     2        1.15             -3.14              1.39

Optimal Location of Knots

   xk[     0]=        0
   xk[     1]=     0.57
   xk[     2]=    0.774
   xk[     3]=        2

Number of Iterations
  itmx=     3

Least Squares error
  s=   0.0289
```

6.2.7 ASL_dgiipc, ASL_rgiipc
積分値と 3 次スプライン係数

(1) **機 能**

端条件入力不要とした “not-a-knot” 条件の 3 次スプライン係数を求め, 指定範囲の積分値を計算する. なお, 節点位置は標本点と一致させる.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dgiipc  (x, y, n, a, b, & q, c);
```

単精度関数:

```
ierr = ASL_rgiipc  (x, y, n, a, b, & q, c);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	x	{D∗ / R∗}	n	入 力	標本点 $(x[i-1], y[i-1]), i = 1, \cdots, n$ の横座標値 $x[i-1]$ $(x[i-1] < x[i], i \neq n)$
2	y	{D∗ / R∗}	n	入 力	標本点の縦座標値, あるいは, 関数値 $y[i-1]$
				出 力	3 次スプライン係数の 0 次項 (入力値と同じ)
3	n	I	1	入 力	標本点の数
4	a	{D / R}	1	入 力	積分範囲下端 (横座標値)
5	b	{D / R}	1	入 力	積分範囲上端 (横座標値)
6	q	{D∗ / R∗}	1	出 力	横座標区間 $[a, b]$ での 3 次スプライン関数の積分値
7	c	{D∗ / R∗}	$3\times(n-1)$	出 力	3 次スプライン係数の k 次項 (k=1, 2, 3): $c[(k-1)+3\times(j-1)]$ 横座標値 $t(x[j-1] \le t < x[j])$ における 3 次スプライン関数の値 $f(t)$ は $f(t) = ((c[2+3\times(j-1)]\times d + c[1+3\times(j-1)])\times d + c[3\times(j-1)])\times d + y[j-1]$ ただし, $d = t - x[j-1]$
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) $n \ge 2$

(b) $x[0] < x[1] < \cdots < x[n-1]$ (昇順)

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	a または b が補間区間の範囲外	端点での 3 次スプライン係数により補外した関数について積分する.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	

(6) **注意事項**

(a) 補間値, 微分値, または積分値をさらに続けて求めたい場合は, この関数を使用した後, 6.2.18 $\left\{\begin{matrix}\text{ASL\_dgiscx}\\ \text{ASL\_rgiscx}\end{matrix}\right\}$, 6.2.19 $\left\{\begin{matrix}\text{ASL\_dgidcy}\\ \text{ASL\_rgidcy}\end{matrix}\right\}$ または 6.2.20 $\left\{\begin{matrix}\text{ASL\_dgiicz}\\ \text{ASL\_rgiicz}\end{matrix}\right\}$ をそれぞれ呼び出せばよい. この時, 配列 x, y, c および変数 n の内容はそのまま後続の関数の対応する引数の入力とする. ただし, 微分値を求めたい場合は, 配列 y の内容を渡す必要がない. このようにすれば, 3 次スプライン係数の計算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.

(7) **使用例**

(a) 問 題

$y_i = x_i e^{-40x_i}$ の式より $\{x_i\} = \{0.0, 0.1, 0.23, 0.34, 0.47, 0.59, 0.73, 0.92, 1.0\}$ における各点の y_i $(i = 1, 2, \cdots, 9)$ をとり出し, これらを標本点として 3 次スプライン関数で補間する. さらに横座標に対応する区間 $[0.2, 0.5]$ での 3 次スプライン関数の積分値を求める.

(b) 入力データ

x[i − 1]=x_i, y[i − 1]=y_i (i = 1, ⋯ , n), a=0.2, b=0.5, n=9

(c) 主プログラム

```
/*      C interface example for ASL_dgiipc */

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

int main()
{
    double *x;
    double *y;
    int nx;
    double a;
    double b;
    double q;
    double *c;
    int ierr;
    int i,j;
    FILE *fp;

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

    printf( "    *** ASL_dgiipc ***\n" );
    printf( "\n     ** Input **\n\n" );
    nx=9;
    a=0.2;
    b=0.5;

    c = ( double * )malloc((size_t)( sizeof(double) * (3*(nx-1)) ));
    if( c == NULL )
    {
        printf( "no enough memory for array c\n" );
        return -1;
    }

    x = ( double * )malloc((size_t)( sizeof(double) * nx ));
```

```
    if( x == NULL )
    {
        printf( "no enough memory for array x\n" );
        return -1;
    }

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

    printf( "\tn = %6d\n", nx );
    printf( "\n\tLimits of Integration a=%8.3g\n", a );
    printf( "\t                       b=%8.3g\n", b );
    for( i=0 ; i<nx ; i++ )
    {
        fscanf( fp, "%lf", &x[i] );
    }
    for( i=0 ; i<nx ; i++ )
    {
        fscanf( fp, "%lf", &y[i] );
    }

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

    fclose( fp );

    ierr = ASL_dgiipc(x, y, nx, a, b, &q, c);

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

    printf( "\tIntegral\n\n" );
    printf( "\t  q=%8.3g\n",q );

    printf( "\n" );
    printf( "\tSpline Coefficients\n\n" );
    printf( "\t  c[i,j]\n" );
    printf( "\t                    i=0      i=1      i=2\n" );
    for( j=0 ; j<nx-1 ; j++ )
    {
        printf( "\t j=%6d",j );
        for( i=0 ; i<3 ; i++ )
        {
            printf( " %8.3g", c[i+3*j] );
        }
        printf( "\n" );
    }

    free( x );
    free( y );
    free( c );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dgiipc ***

 ** Input **

n =      9

Limits of Integration a=     0.2
                      b=     0.5

Coordinates (x,y)

     i     x[i]     y[i]
     0        0        0
     1      0.1    0.067
     2     0.23   0.0917
     3     0.34   0.0873
     4     0.47   0.0717
     5     0.59   0.0557
     6     0.73   0.0394
     7     0.92   0.0232
     8        1   0.0183

 ** Output **

ierr =      0

Integral

  q=  0.0252
```

```
Spline Coefficients

  c[i,j]
                 i=0       i=1       i=2
 j=      0     0.963     -3.29      3.66
 j=      1     0.414      -2.2      3.66
 j=      2    0.0281    -0.769      1.36
 j=      3   -0.0917     -0.32     0.783
 j=      4    -0.135   -0.0145     0.245
 j=      5    -0.128    0.0736    0.0659
 j=      6    -0.104     0.101   -0.0269
 j=      7    -0.068     0.086   -0.0269
```

6.2.8 ASL_dgiisc, ASL_rgiisc
平滑化した積分値と 3 次スプライン係数

(1) **機 能**

最適な平滑化 3 次スプライン係数を自動的に求め, 指定範囲の積分値を計算する. なお, 節点の横座標は標本点の横座標と一致させる.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dgiisc  (x, yd, n, a, b, & q, y, c, isw, wk);
```

単精度関数:

```
ierr = ASL_rgiisc  (x, yd, n, a, b, & q, y, c, isw, wk);
```

(3) 引数と戻り値

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	x	{D∗ / R∗}	n	入 力	標本点 $(x[i-1], yd[i-1]), i = 1, \cdots, n$ の横座標値 $x[i-1]$ $(x[i-1] < x[i], i \neq n)$
2	yd	{D∗ / R∗}	n	入 力	標本点の縦座標値, あるいは, 関数値 $yd[i-1]$
3	n	I	1	入 力	標本点の数
4	a	{D / R}	1	入 力	積分範囲下端 (横座標値)
5	b	{D / R}	1	入 力	積分範囲上端 (横座標値)
6	q	{D∗ / R∗}	1	出 力	横座標区間 $[a, b]$ での 3 次スプライン関数の積分値
7	y	{D∗ / R∗}	n	出 力	3 次スプライン係数の 0 次項
8	c	{D∗ / R∗}	$3\times(n-1)$	出 力	3 次スプライン係数の k 次項 (k=1, 2, 3): $c[(k-1)+3\times(j-1)]$ 横座標値 $t(x[j-1] \leq t < x[j])$ における 3 次スプライン関数の値 $f(t)$ は $f(t) = ((c[2+3\times(j-1)]\times d + c[1+3\times(j-1)])\times d + c[3\times(j-1)])\times d + y[j-1]$ ただし, $d = t - x[j-1]$
9	isw	I	1	入 力	処理スイッチ isw=1: 横座標値の間隔が等間隔とは限らないとき, 選択する. ただし, 標本点が無相関に近いときは, n の値を 20 程度までと制限すること. isw=2: 横座標値の間隔が等間隔であるとき, 選択する.
10	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 **大きさ**: $n\times(2\times n+4)$ (isw=1 のとき) $6\times n$ (isw=2 のとき)
11	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

(a) $n \geq 4$

(b) $x[0] < x[1] < \cdots < x[n-1]$ (昇順)

(c) $isw = 2$ のときは, 横座標値が等間隔であること.

(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.	
1000	a または b が補間区間の範囲以外	端点での 3 次スプライン係数により補外した関数について積分する.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	
4000	クロスバリデーションの最小値が見つからない (データが無相関である).	

(6) **注意事項**

(a) 補間値, 微分値, または積分値をさらに続けて求めたい場合は, この関数を使用した後, 6.2.18 $\left\{\begin{matrix}\text{ASL\_dgiscx}\\ \text{ASL\_rgiscx}\end{matrix}\right\}$, 6.2.19 $\left\{\begin{matrix}\text{ASL\_dgidcy}\\ \text{ASL\_rgidcy}\end{matrix}\right\}$ または 6.2.20 $\left\{\begin{matrix}\text{ASL\_dgiicz}\\ \text{ASL\_rgiicz}\end{matrix}\right\}$ をそれぞれ呼び出せばよい. この時, 配列 x, y, c および変数 n の内容はそのまま後続の関数の対応する引数の入力とする. ただし, 微分値を求めたい場合は, 配列 y の内容を渡す必要がない. このようにすれば, 3 次スプライン係数の計算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.

(7) **使用例**

(a) 問 題

$y_i = \sin(3\pi x_i/2) + e_i$ (e_i : $[-0.2, 0.2]$ の間の一様乱数) の式より $x_i = (i-1)/24$ $(i = 1, 2, \cdots, 25)$ における各点の y_i をとり出し, これらを標本点として平滑化した 3 次スプライン関数で近似する. さらに横座標に対応する区間 $[0.2, 0.5]$ での 3 次スプライン関数の積分値を求める.

(b) 入力データ

x[i − 1]=x_i, yd[i − 1]=y_i (i = 1, ⋯ , n), a = 0.2, b = 0.5, n = 25, isw = 2, m = 10

(c) 主プログラム

```
/*      C interface example for ASL_dgiisc */

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

int main()
{
    double *x;
    double *f;
    int nx;
    double a;
    double b;
    double q;
    double *y;
    double *c;
    int isw;
    double *wk;
    int ierr;
    int i,j;
    FILE *fp;

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

    printf( "    *** ASL_dgiisc ***\n" );
    printf( "\n     ** Input **\n\n" );
    nx=25;
    isw=2;
```

```
a=0.2;
b=0.5;

c = ( double * )malloc((size_t)( sizeof(double) * (3*(nx-1)) ));
if( c == NULL )
{
    printf( "no enough memory for array c\n" );
    return -1;
}

wk = ( double * )malloc((size_t)( sizeof(double) * (6*nx) ));
if( wk == NULL )
{
    printf( "no enough memory for array wk\n" );
    return -1;
}

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


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


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

printf( "\tn  = %6d\n", nx );
printf( "\tisw= %6d\n", isw );
printf( "\n\tLimits of Integration a=%8.3g\n", a );
printf( "\t                      b=%8.3g\n", b );
for( i=0 ; i<nx ; i++ )
{
    fscanf( fp, "%lf", &x[i] );
}
for( i=0 ; i<nx ; i++ )
{
    fscanf( fp, "%lf", &f[i] );
}

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

fclose( fp );

ierr = ASL_dgiisc(x, f, nx, a, b, &q, y, c, isw, wk);

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

printf( "\tIntegral" );
printf( "\tq=%8.3g\n",q );

printf( "\n" );
printf( "\tSpline Coefficients\n\n" );
for( i=0 ; i<nx ; i++ )
{
    printf( "\t  y[%6d]=%8.3g\n", i,y[i] );
}
printf( "\n" );
printf( "\tc[i,j]\n\n" );
printf( "\t               i=0      i=1      i=2\n" );
for( j=0 ; j<nx-1 ; j++ )
{
    printf( "\t j=%6d",j );
    for( i=0 ; i<3 ; i++ )
    {
        printf( " %8.3g", c[i+3*j] );
    }
    printf( "\n" );
}

free( x );
free( f );
free( y );
free( c );
free( wk );
```

```
    return 0;
}
```

(d) 出力結果

```
*** ASL_dgiisc ***

 ** Input **

n  =     25
isw=      2

Limits of Integration a=     0.2
                      b=     0.5

Coordinates (x,yd)

     i     x[i]    yd[i]
     0        0   0.0481
     1   0.0416    0.147
     2   0.0833    0.442
     3    0.125    0.579
     4    0.167    0.641
     5    0.208     0.74
     6     0.25    0.726
     7    0.292     1.09
     8    0.333    0.804
     9    0.375     1.02
    10    0.417    0.851
    11    0.458        1
    12      0.5    0.517
    13    0.542    0.692
    14    0.583    0.348
    15    0.625    0.385
    16    0.667   -0.102
    17    0.708   -0.135
    18     0.75   -0.339
    19    0.792   -0.571
    20    0.833   -0.525
    21    0.875   -0.841
    22    0.917   -0.795
    23    0.958    -1.08
    24        1    -1.18

 ** Output **

ierr =      0

Integral    q=   0.255

Spline Coefficients

  y[     0]=   0.095
  y[     1]=   0.245
  y[     2]=   0.391
  y[     3]=   0.527
  y[     4]=   0.648
  y[     5]=   0.751
  y[     6]=   0.834
  y[     7]=   0.893
  y[     8]=    0.92
  y[     9]=   0.915
  y[    10]=   0.876
  y[    11]=   0.802
  y[    12]=   0.695
  y[    13]=   0.563
  y[    14]=   0.408
  y[    15]=   0.237
  y[    16]=  0.0578
  y[    17]=  -0.121
  y[    18]=  -0.294
  y[    19]=  -0.458
  y[    20]=  -0.613
  y[    21]=  -0.761
  y[    22]=  -0.907
  y[    23]=   -1.05
  y[    24]=    -1.2

c[i,j]

                i=0      i=1      i=2
 j=     0      3.6 2.13e-14     -5.9
 j=     1     3.57   -0.737    -18.1
 j=     2     3.41       -3    -11.7
 j=     3      3.1    -4.47    -5.13
 j=     4      2.7    -5.11    -5.98
 j=     5     2.25    -5.86    -7.46
 j=     6     1.72    -6.79    -21.1
 j=     7     1.05    -9.42     3.88
 j=     8     0.28    -8.94    -10.6
 j=     9    -0.52    -10.3     2.62
 j=    10    -1.36    -9.94    -0.59
 j=    11    -2.19      -10     24.5
 j=    12     -2.9    -6.95      2.1
```

```
j=    13    -3.47    -6.69     18.3
j=    14    -3.93     -4.4     10.8
j=    15    -4.24    -3.05     29.4
j=    16    -4.34    0.628     9.28
j=    17    -4.24     1.79     7.52
j=    18    -4.05     2.73     1.89
j=    19    -3.81     2.96    -12.3
j=    20    -3.63     1.42    -1.37
j=    21    -3.52     1.25    -11.3
j=    22    -3.47   -0.161     2.67
j=    23    -3.47    0.173    -1.39
```

6.2.9 ASL_dgiimc, ASL_rgiimc
最小二乗積分値と 3 次スプライン係数

(1) **機 能**

最小二乗近似 3 次スプライン係数を求め, 指定範囲の積分値を計算する. また, 最適節点位置も求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dgiimc (x, yd, n, xk, nxk, & itmx, a, b, & q, & s, y, c, iwk, wk1, wk2);

単精度関数:

ierr = ASL_rgiimc (x, yd, n, xk, nxk, & itmx, a, b, & q, & s, y, c, iwk, wk1, wk2);

(3) 引数と戻り値

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	x	{D*, R*}	n	入 力	標本点 $(x[i-1], yd[i-1]), i = 1, \cdots, n$ の横座標値 $x[i-1]$ $(x[i-1] < x[i], i \neq n)$
2	yd	{D*, R*}	n	入 力	標本点の縦座標値, あるいは, 関数値 $yd[i-1]$
3	n	I	1	入 力	標本点の数
4	xk	{D*, R*}	nxk	入 力	節点の位置の初期推定値 $xk[i-1] < xk[i], i = 1, \cdots, nxk-1$ $xk[0] \leq x[0]$ $xk[nxk-1] \geq x[n-1]$
				出 力	最適節点位置
5	nxk	I	1	入 力	節点の数
6	itmx	I*	1	入 力	最大反復回数 (15 回程度が適当)
				出 力	実際の反復回数
7	a	{D, R}	1	入 力	積分範囲下端 (横座標値)
8	b	{D, R}	1	入 力	積分範囲上端 (横座標値)
9	q	{D*, R*}	1	出 力	横座標区間 $[a, b]$ での 3 次スプライン関数の積分値
10	s	{D*, R*}	1	出 力	3 次スプライン近似の最小二乗誤差
11	y	{D*, R*}	nxk	出 力	3 次スプライン係数の 0 次項
12	c	{D*, R*}	内容参照	出 力	3 次スプライン係数の k 次項 (k=1, 2, 3): $c[(k-1)+3\times(j-1)]$ 横座標値 $t(xk[j-1] \leq t < xk[j])$ における 3 次スプライン関数の値 $f(t)$ は $f(t) = ((c[2+3\times(j-1)]\times d + c[1+3\times(j-1)])\times d + c[3\times(j-1)])\times d + y[j-1]$ ただし, $d = t - xk[j-1]$ **大きさ:** $3\times(nxk-1)$
13	iwk	I*	75	ワーク	作業領域
14	wk1	I*	内容参照	ワーク	作業領域 **大きさ:** $n \times (nxk+6)$
15	wk2	{D*, R*}	3811	ワーク	作業領域
16	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $n \geq 2, nxk \leq 28, itmx \leq 20$

(b) $x[0] < x[1] < \cdots < x[n-1]$ (昇順)

(c) 両端節点の範囲内に標本点が分布している.

(d) $xk[0] < xk[1] < \cdots < xk[nxk-1]$ (昇順)

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	a または b が補間区間の範囲以外	端点での 3 次スプライン係数により補外した関数について積分する.
1500	最小二乗誤差 (s) の極小が得られないまま, itmx 回の反復回数で計算が終了した (データが無相関に近い).	そのときの 3 次スプライン係数, 最小二乗誤差により補間する.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	
3030	制限条件 (d) を満足しなかった.	

(6) **注意事項**

(a) 節点位置の初期推定値の値によっては, 異なった 3 次スプライン係数が得られる場合や収束状況が変化する場合がある.

(b) y[0], y[nxk − 1] の値は, 通常, 3 次スプライン係数を使って補外した値となる.

(c) 補間値, 微分値, または積分値をさらに続けて求めたい場合は, この関数を使用した後, 6.2.18 $\left\{\begin{matrix}\text{ASL\_dgiscx}\\ \text{ASL\_rgiscx}\end{matrix}\right\}$, 6.2.19 $\left\{\begin{matrix}\text{ASL\_dgidcy}\\ \text{ASL\_rgidcy}\end{matrix}\right\}$ または 6.2.20 $\left\{\begin{matrix}\text{ASL\_dgiicz}\\ \text{ASL\_rgiicz}\end{matrix}\right\}$ をそれぞれ呼び出せばよい. この時, 配列 xk, y, c および変数 nxk の内容はそのまま後続の関数の対応する引数の入力とする. ただし, 微分値を求めたい場合は, 配列 y の内容を渡す必要がない. このようにすれば, 3 次スプライン係数の計算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.

(7) 使用例

(a) 問 題

$$y_i = \begin{cases} 1.0 - x_i & (0.0 \le x_i \le 0.5) \\ x_i & (0.5 < x_i \le 1.0) \\ 2.0 - x_i & (1.0 < x_i \le 2.0) \end{cases}$$

の式より $x_i = 0.1 \times (i-1)$ $(i = 1, 2, \cdots, 21)$ における各点の y_i をとり出し, これらを標本点として 4 つの節点 $\{\xi_j\} = \{0.0, 0.33, 1.33, 2.0\}$ を使って最小二乗近似 3 次スプライン補間を行う. さらに横座標に対応する区間 $[0.5, 1.5]$ での 3 次スプライン関数の積分値を求める.

(b) 入力データ

x[i − 1]=x_i, yd[i − 1]=y_i (i = 1, ⋯ , n), xk[j − 1]=ξ_j (j = 1, ⋯ , nxk), a=0.5, b=1.5, n=21, nxk=4, itmx=15

(c) 主プログラム

```
/*      C interface example for ASL_dgiimc */

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

int main()
{
    double *x;
    double *f;
    int nx;
    double *xk;
    int nxk;
    int itmx;
    double a;
    double b;
    double q;
    double er;
    double *y;
    double *c;
    int iwk[75];
    double *wk1;
    double wk2[3811];
    int ierr;
    int i,j,k,nwk;
    FILE *fp;

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

    printf( "    *** ASL_dgiimc ***\n" );
    printf( "\n     ** Input **\n\n" );
    nx=21;
    nxk=4;
    a=0.5;
    b=1.5;
    itmx=15;

    c = ( double * )malloc((size_t)( sizeof(double) * (3*(nxk-1)) ));
    if( c == NULL )
    {
        printf( "no enough memory for array c\n" );
        return -1;
    }

    nwk=nx*(nxk+6);
    wk1 = ( double * )malloc((size_t)( sizeof(double) * nwk ));
    if( wk1 == NULL )
    {
        printf( "no enough memory for array wk1\n" );
        return -1;
    }

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

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

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

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

printf( "\tn   = %6d\n", nx  );
printf( "\tnxk = %6d\n", nxk );
printf( "\titmx= %6d\n", itmx);
printf( "\n\tLimits of Integration   a=%8.3g\n", a);
printf( "\t                        b=%8.3g\n", b);
for( i=0 ; i<nxk ; i++ )
{
    fscanf( fp, "%lf", &xk[i] );
}
for( i=0 ; i<nx ; i++ )
{
    fscanf( fp, "%lf", &x[i] );
}
for( i=0 ; i<nx ; i++ )
{
    fscanf( fp, "%lf", &f[i] );
}

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

printf( "\n" );
printf( "\tLocations of Knots\n\n" );
printf( "\t     k    xk[k]\n");
for( k=0 ; k<nxk ; k++ )
{
    printf( "\t%6d %8.3g\n", k,xk[k] );
}

fclose( fp );

ierr = ASL_dgiimc(x, f, nx, xk, nxk, &itmx, a, b, &q, &er, y, c, iwk, wk1, wk2);

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

printf( "\tIntegral\n\n" );
printf( "\t  q=%8.3g\n",q );
printf( "\n" );

for( j=0 ; j<nxk ; j++ )
{
    printf( "\ty[%6d]=%8.3g\n", j,y[j] );
}

printf( "\n" );
printf( "\tc[i,j]\n\n" );
printf( "\t               i=0      i=1      i=2\n" );
for( j=0 ; j<nxk-1 ; j++ )
{
    printf( "\t j=%6d",j );
    for( i=0 ; i<3 ; i++ )
    {
        printf( " %8.3g", c[i+3*j] );
    }
    printf( "\n" );
}
printf( "\n" );
printf( "\tOptimal Location of Knots\n\n" );
for( k=0 ; k<nxk ; k++ )
{
    printf( "\t  xk[%6d]= %8.3g\n", k,xk[k] );
}
printf( "\n" );
printf( "\tNumber of Iterations\n\n" );
printf( "\t  itmx=%6d\n",itmx );

printf( "\n" );
printf( "\tLeast Squares Error\n\n" );
printf( "\t  s= %8.3g\n", er );

free( x );
free( f );
```

```
    free( xk );
    free( y );
    free( c );
    free( wk1 );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dgiimc ***

 ** Input **

n   =     21
nxk =      4
itmx=     15

Limits of Integration   a=     0.5
                        b=     1.5

Coordinates (x,yd)

     i     x[i]    yd[i]
     0        0        1
     1      0.1      0.9
     2      0.2      0.8
     3      0.3      0.7
     4      0.4      0.6
     5      0.5      0.5
     6      0.6      0.6
     7      0.7      0.7
     8      0.8      0.8
     9      0.9      0.9
    10        1        1
    11      1.1      0.9
    12      1.2      0.8
    13      1.3      0.7
    14      1.4      0.6
    15      1.5      0.5
    16      1.6      0.4
    17      1.7      0.3
    18      1.8      0.2
    19      1.9      0.1
    20        2        0

Locations of Knots

     k    xk[k]
     0        0
     1     0.33
     2     1.33
     3        2

 ** Output **

ierr =      0

Integral

  q=   0.755

y[     0]=   0.984
y[     1]=   0.558
y[     2]=   0.801
y[     3]=  0.0485

c[i,j]

                i=0      i=1      i=2
 j=     0   -0.345    -3.86     5.53
 j=     1    0.648      5.6    -14.3
 j=     2     1.15    -3.14     1.39

Optimal Location of Knots

  xk[     0]=        0
  xk[     1]=     0.57
  xk[     2]=    0.774
  xk[     3]=        2

Number of Iterations

  itmx=     3

Least Squares Error

  s=   0.0289
```

6.2.10 ASL_dgiccp, ASL_rgiccp
3 次スプライン係数 (端条件入力不要)

(1) **機 能**

端条件入力不要とした “not-a-knot” 条件の 3 次スプライン係数を求める. なお, 節点位置は標本点と一致させる.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dgiccp  (x, y, n, c);
```

単精度関数:

```
ierr = ASL_rgiccp  (x, y, n, c);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	x	{D∗ / R∗}	n	入 力	標本点 $(x[i-1], y[i-1]), i = 1, \cdots, n$ の横座標値 $x[i-1]$ $(x[i-1] < x[i], i \neq n)$
2	y	{D∗ / R∗}	n	入 力	標本点の縦座標値, あるいは関数値 $y[i-1]$
				出 力	3 次スプライン係数の 0 次項 (入力値と同じ)
3	n	I	1	入 力	標本点の数
4	c	{D∗ / R∗}	$3\times(n-1)$	出 力	3 次スプライン係数の k 次項 (k=1, 2, 3): $c[(k-1)+3\times(j-1)]$ 横座標値 $t(x[j-1] \le t < x[j])$ における 3 次スプライン関数の値 $f(t)$ は $f(t) = ((c[2+3\times(j-1)]\times d + c[1+3\times(j-1)])\times d + c[3\times(j-1)])\times d + y[j-1]$ ただし, $d = t - x[j-1]$
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) $n \ge 2$

(b) $x[0] < x[1] < \cdots < x[n-1]$ (昇順)

(5) **エラーインディケータ (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**

なし

6.2.11 ASL_dgiccq, ASL_rgiccq
3 次スプライン係数 (端条件入力)

(1) **機 能**

端条件を指定して, 3 次スプライン係数を求める. なお, 節点位置は標本点と一致させる.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dgiccq  (x, y, n, end, c, isw);
```

単精度関数:

```
ierr = ASL_rgiccq  (x, y, n, end, c, 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	x	{D∗ / R∗}	n	入 力	標本点 $(x[i-1], y[i-1]), i = 1, \cdots, n$ の横座標値 $x[i-1]$ $(x[i-1] < x[i], i \neq n)$
2	y	{D∗ / R∗}	n	入 力	標本点の縦座標値, あるいは, 関数値 $y[i-1]$
				出 力	3 次スプライン係数の 0 次項 (入力値と同じ)
3	n	I	1	入 力	標本点の数
4	end	{D∗ / R∗}	4	入 力	端条件 配列 end に設定する値は, isw の値により以下のようになる (アルゴリズムまたは注意事項参照). **始点** x **[0] での端条件** isw [0] =1:end [0] =始点の y' の値 isw [0] =2:end [0] =始点の y'' の値 isw [0] =3:end [0] =始点の y''' の値 isw [0] =4:end [0] =λ_1, end [1] =d_1 **終点** x[n − 1] **での端条件** isw [2] =1:end [2] =終点の y' の値 isw [2] =2:end [2] =終点の y'' の値 isw [2] =3:end [2] =終点の y''' の値 isw [1] =4:end [2] =λ_{n-1}, end [3] =d_{n-1}
5	c	{D∗ / R∗}	3×(n − 1)	出 力	3 次スプライン係数の k 次項 (k=1, 2, 3): c[(k − 1) + 3 × (j − 1)] 横座標値 $t(x[j-1] \leq t < x[j])$ における 3 次スプライン関数の値 f(t) は $f(t) = ((c[2+3\times(j-1)]\times d + c[1+3\times(j-1)])\times d + c[3\times(j-1)])\times d + y[j-1]$ ただし, $d = t - x[j-1]$
6	isw	I*	2	入 力	端条件配列 end に設定する値に対するスイッチ. isw [0] は始点, isw [1] は終点に対応する (引数 end 参照).
7	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

(a) $n \geq 2$

(b) $x[0] < x[1] < \cdots < x[n-1]$ (昇順)

(5) **エラーインディケータ (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) isw [0] または isw [1] の値が 1, 2, 3 のときでも配列 end の大きさは, 4 とする必要がある. この場合, end [1] と end [3] には値を設定する必要がない.

(b) isw [0] ≥ 4 または isw [1] ≥ 4 のとき配列 end に与える値は 3 次スプライン係数を求める場合に得られる次の連立 1 次方程式の係数 $\lambda_1, d_1, \mu_{\mathrm{n}}, d_{\mathrm{n}}$ の値である. 詳細は 6.1.2 使用しているアルゴリズム参照.

$$\begin{bmatrix} 2 & \lambda_1 & & & & \\ & & & & 0 & \\ \mu_2 & 2 & \lambda_2 & & & \\ & \mu_3 & \ddots & \ddots & & \\ & \ddots & \ddots & \lambda_{\mathrm{n}-2} & & \\ & \mu_{\mathrm{n}-1} & & 2 & \lambda_{\mathrm{n}-1} & \\ 0 & & & \mu_{\mathrm{n}} & 2 & \end{bmatrix} \begin{bmatrix} M_1 \\ M_2 \\ \vdots \\ \vdots \\ \vdots \\ M_{\mathrm{n}-1} \\ M_{\mathrm{n}} \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ \vdots \\ \vdots \\ d_{\mathrm{n}-1} \\ d_{\mathrm{n}} \end{bmatrix}$$

ここで $M_1, \cdots, M_{\mathrm{n}}$ は 3 次スプライン関数の 2 次微係数である.

6.2.12 ASL_dgiccr, ASL_rgiccr
3 次スプライン係数 (周期スプライン)

(1) **機 能**

周期的端条件のもとで 3 次スプライン係数を求める. なお, 節点位置は標本点と一致させる.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dgiccr  (x, y, n, c, wk);
```

単精度関数:

```
ierr = ASL_rgiccr  (x, y, n, c, 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	x	{D∗ / R∗}	n	入 力	標本点 $(x[i-1], y[i-1]), i = 1, \cdots, n$ の横座標値 $x[i-1]$ $(x[i-1] < x[i], i \neq n)$
2	y	{D∗ / R∗}	n	入 力	標本点の縦座標値, あるいは関数値 $y[i-1]$
				出 力	3 次スプライン係数の 0 次項 (入力値と同じ)
3	n	I	1	入 力	標本点の数
4	c	{D∗ / R∗}	$3\times(n-1)$	出 力	3 次スプライン係数の k 次項 (k=1, 2, 3): $c[(k-1)+3\times(j-1)]$ 横座標値 $t(x[j-1] \le t < x[j])$ における 3 次スプライン関数の値 $f(t)$ は $f(t) = ((c[2+3\times(j-1)]\times d + c[1+3\times(j-1)])\times d + c[3\times(j-1)])\times d + y[j-1]$ ただし, $d = t - x[j-1]$
5	wk	{D∗ / R∗}	$5\times n$	ワーク	作業領域
6	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $n \ge 4$

(b) $x[0] < x[1] < \cdots < x[n-1]$ (昇順)

(c) $y[0] = y[n-1]$ (周期的端条件)

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
2000	制限条件 (c) を満足しなかった.	y[n − 1] を y [0] と同じとみなして処理を行う.
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) 周期スプラインは, 横座標に関する補間区間 [x[0], x[n − 1]] の曲線パターンが周期的に続くものとして補間する. したがって, 求めようとする値 (補間値, 微分値, 積分値) が補間区間内にある場合のみ意味ある結果が得られる.

6.2.13 ASL_dgiccs, ASL_rgiccs
3 次スプライン係数 (自動平滑化)

(1) **機 能**

平滑化制御変数を自動的に決めて, 最適な平滑化 3 次スプライン係数を求める. なお, 節点の横座標は標本点の横座標と一致させる.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dgiccs  (x, yd, n, y, c, & s, isw, wk);
```

単精度関数:

```
ierr = ASL_rgiccs  (x, yd, n, y, c, & s, isw, 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	x	{D∗ / R∗}	n	入 力	標本点 $(x[i-1], yd[i-1]), i = 1, \cdots, n$ の横座標値 $x[i-1]$ $(x[i-1] < x[i], i \neq n)$
2	yd	{D∗ / R∗}	n	入 力	標本点の縦座標値, あるいは関数値 $yd[i-1]$
3	n	I	1	入 力	標本点の数
4	y	{D∗ / R∗}	n	出 力	3 次スプライン係数の 0 次項
5	c	{D∗ / R∗}	$3\times(n-1)$	出 力	3 次スプライン係数の k 次項 (k=1, 2, 3): $c[(k-1)+3\times(j-1)]$ 横座標値 $t(x[j-1] \le t < x[j])$ における 3 次スプライン関数の値 f(t) は $f(t) = ((c[2+3\times(j-1)]\times d + c[1+3\times(j-1)])\times d + c[3\times(j-1)])\times d + y[j-1]$ ただし, $d = t - x[j-1]$
6	s	{D∗ / R∗}	1	出 力	誤差の 2 乗和 $(\sum_{i=1}^{n}\{yd[i-1]-y[i-1]\}^2)$
7	isw	I	1	入 力	処理スイッチ isw=1: 横座標値の間隔が等間隔とは限らないとき, 選択する. ただし, 標本点が無相関に近いときは, n の値を 20 程度までと制限すること. isw=2: 横座標値の間隔が等間隔であるとき, 選択する.

項番	引数と戻り値	型	大きさ	入出力	内　　容
8	wk	$\left\{\begin{matrix}\text{D}* \\ \text{R}*\end{matrix}\right\}$	内容参照	ワーク	作業領域 **大きさ**: $\text{n} \times (2 \times \text{n} + 4)$ (isw=1 のとき) $6 \times \text{n}$ (isw=2 のとき)
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) $\text{n} \geq 4$

(b) $\text{x}[0] < \text{x}[1] < \cdots < \text{x}[\text{n}-1]$ (昇順)

(c) $\text{isw} = 2$ のときは, 横座標値が等間隔であること.

(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.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	
4000	クロスバリデーションの最小値が見つからない (データが無相関である).	

(6) **注意事項**

なし

6.2.14 ASL_dgicco, ASL_rgicco
3 次スプライン係数 (自動平滑化周期条件)

(1) **機 能**

周期端条件のもとで平滑化制御変数を自動的に決めて, 最適に平滑化された 3 次スプライン係数を求める. なお, 節点の横座標は標本点の横座標と一致させる.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dgicco  (x, yd, n, y, c, & s, wk);
```

単精度関数:

```
ierr = ASL_rgicco  (x, yd, n, y, c, & s, 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	x	{D∗ R∗}	n	入 力	標本点 $(x[i-1], yd[i-1]), i = 1, \cdots, n$ の横座標値 $x[i-1]$ $(x[i-1] < x[i], i \neq n)$
2	yd	{D∗ R∗}	n	入 力	標本点の縦座標値, あるいは関数値 $yd[i-1]$
3	n	I	1	入 力	標本点の数
4	y	{D∗ R∗}	n	出 力	3 次スプライン係数の 0 次項
5	c	{D∗ R∗}	$3\times(n-1)$	出 力	3 次スプライン係数の k 次項 (k=1, 2, 3): $c[(k-1)+3\times(j-1)]$ 横座標値 $t(x[j-1] \le t < x[j])$ における 3 次スプライン関数の値 $f(t)$ は $f(t) = ((c[2+3\times(j-1)]\times d + c[1+3\times(j-1)])\times d + c[3\times(j-1)])\times d + y[j-1]$ ただし, $d = t - x[j-1]$
6	s	{D∗ R∗}	1	出 力	誤差の 2 乗和 $(\sum_{i=1}^{n}\{yd[i-1]-y[i-1]\}^2)$
7	wk	{D∗ R∗}	内容参照	ワーク	作業領域 **大きさ:** $n\times(2\times n+4)$
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) $n \ge 4$

(b) $x[0] < x[1] < \cdots < x[n-1]$ (昇順)

(5) **エラーインディケータ (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.	
4000	クロスバリデーションの最小値が見つからない (データが無相関である).	

(6) **注意事項**

なし

6.2.15 ASL_dgicct, ASL_rgicct
3 次スプライン係数 (制御変数指定平滑化)

(1) **機 能**

制御変数を指定して, 平滑化 (自然)3 次スプライン係数を求める. なお, 節点位置の横座標は標本点の横座標と一致させる.

(2) **使用法**

倍精度関数:

ierr = ASL_dgicct (x, yd, w, n, sf, y, c, wk);

単精度関数:

ierr = ASL_rgicct (x, yd, w, n, sf, y, c, 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	x	{D∗ / R∗}	n	入 力	標本点 $(x[i-1], yd[i-1]), i = 1, \cdots, n$ の横座標値 $x[i-1]$ $(x[i-1] < x[i], i \neq n)$
2	yd	{D∗ / R∗}	n	入 力	標本点の縦座標値, あるいは関数値 $yd[i-1]$
3	w	{D∗ / R∗}	n	入 力	各標本点の相対的重み $(w[i-1] > 0)$ 重み付けを行わない点については通常 1.0 を設定する.
4	n	I	1	入 力	標本点の数
5	sf	{D / R}	1	入 力	平滑化の程度を制御する制御変数 $(sf > 0)$ 3 次スプライン関数 f(x) は, 次式が満足するように決められる. $\sum_{i=1}^{n}((f(x[i-1]) - yd[i-1])/w[i-1])^2 \leq sf$ ただし, f(x) が 1 次関数でなければ等号が成立する.
6	y	{D∗ / R∗}	n	出 力	3 次スプライン係数の 0 次項
7	c	{D∗ / R∗}	$3\times(n-1)$	出 力	3 次スプライン係数の k 次項 (k=1, 2, 3): $c[(k-1)+3\times(j-1)]$ 横座標値 $t(x[j-1] \leq t < x[j])$ における 3 次スプライン関数の値 f(t) は $f(t) = ((c[2+3\times(j-1)]\times d + c[1+3\times(j-1)])\times d + c[3\times(j-1)])\times d + y[j-1]$ ただし, $d = t - x[j-1]$
8	wk	{D∗ / R∗}	$7\times n+14$	ワーク	作業領域
9	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $n \geq 2$

(b) $x[0] < x[1] < \cdots < x[n-1]$ (昇順)

(5) **エラーインディケータ (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) 入力 sf の値は, 入力データ点と平滑化された曲線との距離の 2 乗和に相当し, 大きな値の時は, 直線に近づき, 小さな値のときは完全補間に近づく.

(b) 得られる 3 次スプライン係数は自然スプラインとなるため, 両端 (x [0] と $x[n-1]$) での 3 次スプライン関数の 2 階微分値は 0 となる.

6.2.16 ASL_dgiccm, ASL_rgiccm
3 次スプライン係数 (節点位置自動最小二乗法)

(1) **機 能**

自動的に節点の最適位置をさがし, 最良の最小二乗近似 3 次スプライン係数を求める. また, 最適節点位置も求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dgiccm (x, yd, n, xk, nxk, & itmx, y, c, & s, iwk, wk1, wk2);

単精度関数:

ierr = ASL_rgiccm (x, yd, n, xk, nxk, & itmx, y, c, & s, iwk, wk1, wk2);

(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∗}	n	入 力	標本点 $(x[i-1], yd[i-1]), i = 1, \cdots, n$ の横座標値 $x[i-1]$ $(x[i-1] < x[i], i \neq n)$
2	yd	{D∗ R∗}	n	入 力	標本点の縦座標値, あるいは, 関数値 $yd[i-1]$
3	n	I	1	入 力	標本点の数
4	xk	{D∗ R∗}	nxk	入 力	節点の位置の初期推定値 $xk[i-1] < xk[i], i = 1, \cdots, nxk-1$ $xk[0] \leq x[0]$ $xk[nxk-1] \geq x[n-1]$
				出 力	最適節点位置
5	nxk	I	1	入 力	節点の数
6	itmx	I*	1	入 力	最大反復回数 (15 回程度が適当)
				出 力	実際の反復回数
7	y	{D∗ R∗}	nxk	出 力	3 次スプライン係数の 0 次項
8	c	{D∗ R∗}	内容参照	出 力	3 次スプライン係数の k 次項 (k=1, 2, 3): $c[(k-1)+3\times(j-1)]$ 横座標値 $t(xk[j-1] \leq t < xk[j])$ における 3 次スプライン関数の値 $f(t)$ は $f(t) = ((c[2+3\times(j-1)]\times d + c[1+3\times(j-1)])\times d + c[3\times(j-1)])\times d + y[j-1]$ ただし, $d = t - xk[j-1]$ **大きさ:** $3\times(nxk-1)$
9	s	{D∗ R∗}	1	出 力	3 次スプライン近似の最小二乗誤差
10	iwk	I*	75	ワーク	作業領域
11	wk1	{D∗ R∗}	内容参照	ワーク	作業領域 **大きさ:** $n \times (nxk+6)$
12	wk2	{D∗ R∗}	3811	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) $n \geq 2, nxk \leq 28, itmx \leq 20$

(b) $x[0] < x[1] < \cdots < x[n-1]$ (昇順)

(c) 両端節点の範囲内に標本点が分布している.

(d) $xk[0] < xk[1] < \cdots < xk[nxk-1]$ (昇順)

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1500	最小二乗誤差 (s) の極小が得られないまま, itmx 回の反復回数で計算が終了した (データが無相関に近い).	そのときの 3 次スプライン係数, 最小二乗誤差により補間する.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	
3030	制限条件 (d) を満足しなかった.	

(6) **注意事項**

(a) 節点位置の初期推定値の値によっては, 異なった 3 次スプライン係数が得られる場合や収束状況が変化する場合がある.

(b) y[0], y[nxk − 1] の値は, 通常, 3 次スプライン係数を使って補外した値となる.

6.2.17 ASL_dgiccn, ASL_rgiccn
3 次スプライン係数 (節点位置指定最小二乗法)

(1) **機 能**

指定された節点で最小二乗近似 3 次スプライン係数を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dgiccn  (x, yd, n, xk, & nxk, y, c, nkm, & s, isw, iwk, wk1, wk2);
```

単精度関数:

```
ierr = ASL_rgiccn  (x, yd, n, xk, & nxk, y, c, nkm, & s, isw, iwk, wk1, wk2);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	x	{ D∗ / R∗ }	n	入 力	標本点 $(x[i-1], yd[i-1]), i = 1, \cdots, n$ の横座標値 $x[i-1]$ $(x[i-1] < x[i], i \neq n)$
2	yd	{ D∗ / R∗ }	n	入 力	標本点の縦座標値, あるいは関数値 $yd[i-1]$
3	n	I	1	入 力	標本点の数
4	xk	{ D∗ / R∗ }	内容参照	入 力	節点横座標値 isw=0: $xk[i-1]$ $i = 1, \cdots, nxk$ には左端節点, 内側節点, 右端節点の順ですべての節点横座標値を入力する. isw = 1 または 2:追加節点横座標値を $xk[i-1]$ $i = 1, \cdots, nxk$ に入力する. isw=3:修正後の節点横座標値を xk[0] に入力する. **大きさ:** $\max(1, nxk)$
				出 力	そのときの全節点横座標値 (isw = 4 のとき)
5	nxk	I*	1	入 力	節点数 isw=0:全節点数を設定する isw = 1 または 2: 節点を追加するときは節点数 (正の値) を, 削除するときは節点数の符号を変えた値 (負の値) を設定する. (注意事項 (d) 参照) isw=3: nxk=1 とする.
				出 力	そのときの全節点数 (isw = 4 のとき)
6	y	{ D∗ / R∗ }	nkm	出 力	3 次スプライン係数の 0 次項

項番	引数と戻り値	型	大きさ	入出力	内　　容
7	c	$\left\{\begin{matrix}\text{D*}\\\text{R*}\end{matrix}\right\}$	内容参照	出 力	3 次スプライン係数の k 次項 (k=1, 2, 3): c[(k − 1) + 3 × (j − 1)] 配列 xk に isw=4 を指定した場合に得られる全節点情報が格納されているとしたとき, 横座標値 t(xk[j−1] ≤ t < xk[j]) における 3 次スプライン関数の値 f(t) は f(t) = ((c[2 + 3 × (j − 1)] × d + c[1 + 3 × (j − 1)]) × d + c[3 × (j − 1)]) × d + y[j − 1] ただし, d = t − xk[j − 1] **大きさ:** 3×(nkm − 1)
8	nkm	I	1	入 力	節点の最大数
9	s	$\left\{\begin{matrix}\text{D*}\\\text{R*}\end{matrix}\right\}$	1	出 力	3 次スプライン近似の最小二乗誤差
10	isw	I	1	入 力	処理スイッチ isw=0: 入力節点に対する最小二乗 3 次スプライン係数を求める (最初は必ず isw = 0 で行う). isw=1: nxk ≥ 0 のときは前回までの節点と xk[i−1] に入力された追加節点により新しい 3 次スプライン係数を計算する. nxk < 0 のときは, 前回までの挿入節点から −nxk 個削除して 3 次スプライン係数を再計算する. isw=2: isw = 1 と同じである. 節点の削除後, 追加し直すようなとき効率良く処理を行う. isw=3: 前回の最後に入力された節点の横座標値を xk[0] に変更する. 最小二乗誤差のみの出力で 3 次スプライン係数は計算しない. isw=4: nxk にそのときの節点数, xk[i − 1] i = 1, ··· , nxk に各節点の横座標値を返す.
11	iwk	I*	75	ワーク	作業領域
12	wk1	$\left\{\begin{matrix}\text{D*}\\\text{R*}\end{matrix}\right\}$	内容参照	ワーク	作業領域 **大きさ:** n × (nkm + 6)
13	wk2	$\left\{\begin{matrix}\text{D*}\\\text{R*}\end{matrix}\right\}$	3811	ワーク	作業領域
14	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $n \geq 2, nxk \leq nkm \leq n, nkm \leq 28, isw = 0$ のときは $nxk \geq 2$

(b) $x[0] < x[1] < \cdots < x[n-1]$ (昇順)

(c) 同じ節点が複数個あってはいけない.

(d) 左端節点, 右端節点の範囲内に節点を指定する. さらに, この範囲内でデータを与える.

(e) isw=4 のとき nxk ≥(そのときの全節点数)

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) または (d) を満足しなかった.	
3030	制限条件 (e) を満足しなかった.	

(6) **注意事項**

(a) 配列 x, yd, iwk, wk1, wk2 は, この関数を使用して最適の 3 次スプライン係数を得るまでの間, 保存しておかなければならない.

(b) isw=0 では, xk [0] に左端節点横座標値, $xk[nxk-1]$ に右端節点横座標値を入力する.

(c) 最初に最小二乗近似 3 次スプライン係数を求めるときは, isw=0 を指定してこの関数実行する. その後の節点の変更は isw=1, 2 または 3 を指定して行う. そのときの節点数や節点位置は, isw=4 を指定してこの関数実行することによって, nxk と $xk[i-1]$(昇順) にそれぞれ出力される.

(d) isw=1 または 2 で, 節点をいくつか削除する場合, 追加したときと逆の順に, 横座標値の大きい方から削除される. ただし, 両端の節点は削除されない.
$nxk < -$(節点の個数 $-2) < 0$ の場合は両端を除く全ての節点が削除される. 例えば,

i. isw=0 で節点横座標値 {1.0, 2.0, 5.0, 7.0, 9.0} を入力

ii. isw=1 で節点横座標値 {1.5, 2.5, 6.0} を追加

iii. isw=1 で節点横座標値 {3.0, 8.0} を追加

という過程を経て, 現在の節点が横座標値が {1.0, 1.5, 2.0, 2.5, 3.0, 5.0, 6.0, 7.0, 8.0, 9.0} となった場合, $nxk = -3$ として 3 個の節点を削除する場合には横座標値が {8.0, 3.0, 6.0} の節点が削除され, $nxk = -6$ として 6 個の節点を削除する場合には横座標値が {8.0, 3.0, 6.0, 2.5, 1.5, 7.0} の節点が削除される. また, $nxk = -9$ として 9 個の節点を削除しようとした場合には, 8 個の節点が削除され横座標値が {1.0, 9.0} である両端の節点だけが残る.

(e) isw=3 で節点位置を修正した後, isw=1 または 2 で nxk=0 として実行すれば, その時点の 3 次スプライン係数が得られる.

6.2.18 ASL_dgiscx, ASL_rgiscx
3 次スプライン係数による補間値

(1) **機 能**

与えられた 3 次スプライン係数から 3 次スプライン関数の補間値を計算する.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dgiscx  (x, y, n, c, xl, yl, m);
```

単精度関数:

```
ierr = ASL_rgiscx  (x, y, n, c, xl, yl, 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	x	{D∗ / R∗}	n	入 力	3次スプライン関数の節点横座標値 x[i−1] (x[i−1] < x[i], i ≠ n)
2	y	{D∗ / R∗}	n	入 力	3 次スプライン係数の 0 次項
3	n	I	1	入 力	節点の数
4	c	{D∗ / R∗}	3×(n − 1)	入 力	3 次スプライン係数の k 次項 (k=1, 2, 3): c[(k − 1) + 3 × (j − 1)] 横座標値 t(x[j − 1] ≤ t < x[j]) における 3 次スプライン関数の値 f(t) は f(t) = ((c[2 + 3 × (j − 1)] × d + c[1 + 3 × (j − 1)]) × d + c[3 × (j − 1)]) × d + y[j − 1] ただし, d = t − x[j − 1]
5	xl	{D∗ / R∗}	m	入 力	補間値を計算する点の横座標値
6	yl	{D∗ / R∗}	m	出 力	xl[i − 1] での 3 次スプライン補間値
7	m	I	1	入 力	補間点の数
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) $n \geq 2, m > 0$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	xl[i − 1] が補間区間の範囲外であった (xl[i − 1] < x[0] or xl[i − 1] > x[n − 1]).	端点での 3 次スプライン係数を利用し, 補外した値を出力する.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.

(6) **注意事項**

なし

6.2.19 ASL_dgidcy, ASL_rgidcy
3 次スプライン係数による微分値

(1) **機 能**

与えられた 3 次スプライン係数から 3 次スプライン関数の 1 階, 2 階微分値を計算する.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dgidcy  (x, n, c, xl, dl, ddl, m);
```

単精度関数:

```
ierr = ASL_rgidcy  (x, n, c, xl, dl, ddl, 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	x	{D∗ / R∗}	n	入 力	3 次スプライン関数の節点横座標値 x[i−1] (x[i−1] < x[i], i ≠ n)
2	n	I	1	入 力	節点の数
3	c	{D∗ / R∗}	3×(n − 1)	入 力	3 次スプライン係数の k 次項 (k=1, 2, 3): c[(k − 1) + 3 × (j − 1)] 横座標値 t(x[j − 1] ≤ t < x[j]) における 3 次スプライン関数の値 f(t) は f(t) = ((c[2 + 3 × (j − 1)] × d + c[1 + 3 × (j − 1)]) × d + c[3 × (j − 1)]) × d + y[j − 1] ただし, d = t − x[j − 1], y[j − 1] は 3 次スプライン係数の 0 次項
4	xl	{D∗ / R∗}	m	入 力	3 次スプライン関数の微分値を計算する点の横座標値
5	dl	{D∗ / R∗}	m	出 力	xl[i − 1] での 3 次スプライン関数の 1 階微分値 dl[i − 1] = (3.0 × c[2 + 3 × (j − 1)] × d + 2.0 × c[1 + 3 × (j − 1)]) × d + c[3 × (j − 1)] ただし, x[j − 1] ≤ xl[i − 1] < x[j] d = xl[i − 1] − x[j − 1]
6	ddl	{D∗ / R∗}	m	出 力	xl[i − 1] での 3 次スプライン関数の 2 階微分値 ddl[i − 1] = 6.0 × c[2 + 3 × (j − 1)] × d + 2.0 × c[1 + 3 × (j − 1)] ただし, x[j − 1] ≤ xl[i − 1] < x[j] d = xl[i − 1] − x[j − 1]
7	m	I	1	入 力	微分値を計算する点の数
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) $\mathrm{n} \geq 2, \mathrm{m} > 0$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	xl[i − 1] が補間区間の範囲外であった (xl[i − 1] < x[0] or xl[i − 1] > x[n − 1]).	端点での 3 次スプライン係数を利用し, 補外した値を出力する.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.

(6) **注意事項**

なし

6.2.20 ASL_dgiicz, ASL_rgiicz
3 次スプライン係数による積分値

(1) **機 能**

与えられた 3 次スプライン係数から 3 次スプライン関数の積分値を計算する.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dgiicz  (x, y, n, c, a, b, & q);
```

単精度関数:

```
ierr = ASL_rgiicz  (x, y, n, c, a, b, & q);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	x	{D∗ / R∗}	n	入 力	3 次スプライン関数の節点横座標値 x[i−1] (x[i−1] < x[i], i ≠ n)
2	y	{D∗ / R∗}	n	入 力	3 次スプライン係数の 0 次項
3	n	I	1	入 力	節点の数
4	c	{D∗ / R∗}	3×(n − 1)	入 力	3 次スプライン係数の k 次項 (k=1, 2, 3): c[(k − 1) + 3 × (j − 1)] 横座標値 t(x[j − 1] ≤ t < x[j]) における 3 次スプライン関数の値 f(t) は f(t) = ((c[2 + 3 × (j − 1)] × d + c[1 + 3 × (j − 1)]) × d + c[3 × (j − 1)]) × d + y[j − 1] ただし, d = t − x[j − 1]
5	a	{D / R}	1	入 力	積分範囲下端 (横座標値)
6	b	{D / R}	1	入 力	積分範囲上端 (横座標値)
7	q	{D∗ / R∗}	1	出 力	横座標区間 [a, b] での 3 次スプライン関数の積分値
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 ≥ 2

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	a または b が補間区間の範囲外	端点での 3 次スプライン係数により補外した関数について積分する.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.

(6) **注意事項**

なし

6.3 双 3 次スプライン (曲面補間)

6.3.1 ASL_dgisxb, ASL_rgisxb
補間値

(1) **機 能**

格子上のデータより双 3 次スプライン補間を行い, 新たな格子上の点の補間値を求める. なお, 端条件は “not-a-knot” 条件, 節点位置は標本点と一致させる.

(2) **使用法**

倍精度関数:

ierr = ASL_dgisxb (x, nx, y, ny, z, xl, nxl, yl, nyl, fl, wk);

単精度関数:

ierr = ASL_rgisxb (x, nx, y, ny, z, xl, nxl, yl, nyl, fl, 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	x	{D∗ / R∗}	nx	入 力	標本点の X 座標値 $x[i-1]$, $i=1,\cdots,nx$ $(x[i-1]<x[i], i\neq nx)$
2	nx	I	1	入 力	X 方向標本点数
3	y	{D∗ / R∗}	ny	入 力	標本点の Y 座標値 $y[j-1]$, $j=1,\cdots,ny$ $(y[j-1]<y[j], j\neq ny)$
4	ny	I	1	入 力	Y 方向標本点数
5	z	{D∗ / R∗}	nx×ny	入 力	標本点 $(x[i-1], y[j-1])$ での関数値 z_{ij} からなる配列: $z[(i-1)+nx\times(j-1)]=z_{ij}$
6	xl	{D∗ / R∗}	nxl	入 力	補間値を計算する格子点の X 座標値
7	nxl	I	1	入 力	X 方向補間点数
8	yl	{D∗ / R∗}	nyl	入 力	補間値を計算する格子点の Y 座標値
9	nyl	I	1	入 力	Y 方向補間点数
10	fl	{D∗ / R∗}	内容参照	出 力	補間点 $(xl[i-1], yl[j-1])$ での双 3 次スプライン関数値 f_{ij} からなる配列 $fl[(i-1)+nxl\times(j-1)]=f_{ij}$ なお, $ny>nyl$ のとき $fl[(i-1)+nxl\times(j-1)]$ $(i=1,\cdots,nxl; j=nyl+1,\cdots,ny)$ の領域は作業領域としても利用するので, 計算後不定となる. **大きさ:** $nxl\times\max(nyl, ny)$

項番	引数と戻り値	型	大きさ	入出力	内　　容
11	wk	$\left\{\begin{matrix}\text{D}*\\ \text{R}*\end{matrix}\right\}$	内容参照	ワーク	作業領域 **大きさ**: $\max\{3\times(\text{nx}-1), 3\times(\text{ny}-1)+\text{ny}\}$
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) $\text{n} \geq 2, \text{ny} \geq 2$

(b) $\text{x}[0] < \text{x}[1] < \cdots < \text{x}[\text{nx}-1]$ (昇順)

(c) $\text{y}[0] < \text{y}[1] < \cdots < \text{y}[\text{ny}-1]$ (昇順)

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	補間点 $(\text{xl}[\text{i}-1], \text{yl}[\text{j}-1])$ が補間領域外である.	境界での双 3 次スプライン係数を利用し, 補外した値を出力する.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) または (c) を満足しなかった.	

(6) **注意事項**

(a) この関数では双 3 次スプライン係数の出力は行わない.

(b) この関数では与えられた格子点上の全ての点に対する双 3 次スプライン補間値を求める. 特定の点での双 3 次スプライン補間値を求めたい場合には, 6.3.4 $\left\{\begin{matrix}\text{ASL\_dgicbp}\\ \text{ASL\_rgicbp}\end{matrix}\right\}$ と 6.3.5 $\left\{\begin{matrix}\text{ASL\_dgisbx}\\ \text{ASL\_rgisbx}\end{matrix}\right\}$ をつづけて使用した方が効率がよい.

(7) **使用例**

(a) 問 題

格子座標値 $(\text{x}_\text{i}, \text{y}_\text{j}) = (2\times\text{i}-1, 2\times\text{j}-1)$ $(\text{i}=1,2,3;\ \text{j}=1,2,3)$ での関数値 z_ij からなる行列 Z が

$$Z = \begin{bmatrix} 2.0 & 0.5 & 1.0 \\ 4.0 & 1.0 & 2.0 \\ 3.0 & 0.75 & 1.5 \end{bmatrix}$$

と与えられたとき, これらを標本点として双 3 次スプライン関数で近似し, 格子点 $(\text{xl}_\text{i}, \text{yl}_\text{j}) = (0.5\times(\text{i}+1), 0.5\times(\text{j}+2))$ $(\text{i}=1,2;\ \text{j}=1,2)$ での補間値を求める.

(b) 入力データ

$\text{x}[\text{i}-1] = \text{x}_\text{i}(\text{i}=1,\cdots,\text{nx})$,
$\text{y}[\text{j}-1] = \text{y}_\text{j}(\text{j}=1,\cdots,\text{ny})$,
$\text{z}[(\text{i}-1)+\text{nx}\times(\text{j}-1)] = \text{z}_\text{ij}$
$\text{xl}[\text{i}-1] = \text{xl}_\text{i}(\text{i}=1,\cdots,\text{nxl})$,
$\text{yl}[\text{j}-1] = \text{yl}_\text{j}(\text{j}=1,\cdots,\text{nyl})$,
nx=3, ny=3, nxl=2, nyl=2

(c) 主プログラム

```
/*      C interface example for ASL_dgisxb */

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

int main()
{
    double *x;
    int nx;
    double *y;
    int ny;
    double *f;
    double *xl;
    int nxl;
    double *yl;
    int nyl;
    double *fl;
    double *wk;
    int ierr;
    int i,j,nwk;
    FILE *fp;

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

    printf( "    *** ASL_dgisxb ***\n" );
    printf( "\n     ** Input **\n\n" );
    nx=3;
    ny=3;
    nxl=2;
    nyl=2;

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

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

    nwk=( (3*nx-3>4*ny-3) ? 3*nx-3 : 4*ny-3 );
    wk = ( double * )malloc((size_t)( sizeof(double) * nwk ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

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

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

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

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

    printf( "\tn   = %6d\n", nx  );
    printf( "\tn   = %6d\n", ny  );
    printf( "\tnxl = %6d\n", nxl );
    printf( "\tnyl = %6d\n", nyl );
    for( i=0 ; i<nx ; i++ )
    {
        fscanf( fp, "%lf", &x[i] );
    }
```

```
    for( i=0 ; i<nx ; i++ )
    {
        fscanf( fp, "%lf", &y[i] );
    }
    for( j=0 ; j<ny ; j++ )
    {
        for( i=0 ; i<nx ; i++ )
        {
            fscanf( fp, "%lf", &f[i+nx*j] );
        }
    }
    for( i=0 ; i<nxl ; i++ )
    {
        fscanf( fp, "%lf", &xl[i] );
    }
    for( i=0 ; i<nyl ; i++ )
    {
        fscanf( fp, "%lf", &yl[i] );
    }

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

    printf( "\n" );
    printf( "\tSpecified Points\n\n" );
    printf( "\t     j    xl[j]    yl[j]\n");
    for( j=0 ; j<nxl ; j++ )
    {
        printf( "\t%6d %8.3g %8.3g\n", j,xl[j],yl[j] );
    }

    printf( "\n" );
    printf( "\tFunction Values\n\n" );
    printf( "\t  z[i,j]\n");
    printf( "\t                j=0      j=1      j=2\n" );
    for( i=0 ; i<nx ; i++ )
    {
        printf( "\t i=%6d",i );
        for( j=0 ; j<ny ; j++ )
        {
            printf( " %8.3g", f[i+nx*j] );
        }
        printf( "\n" );
    }

    fclose( fp );

    ierr = ASL_dgisxb(x, nx, y, ny, f, xl, nxl, yl, nyl, fl, wk);

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

    printf( "\tEstimated Function Values in (x,y)\n\n" );
    printf( "\t  fl[i,j]\n" );
    printf( "\t                j=0      j=1\n" );
    for( i=0 ; i<nxl ; i++ )
    {
        printf( "\t i=%6d",i );
        for( j=0 ; j<nyl ; j++ )
        {
            printf( " %8.3g", fl[i+nxl*j] );
        }
        printf( "\n" );
    }

    free( x );
    free( y );
    free( f );
    free( xl );
    free( yl );
    free( fl );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dgisxb ***

 ** Input **

n   =      3
n   =      3
nxl =      2
nyl =      2
```

```
Coordinates (x,y)

     i     x[i]     y[i]
     0        1        1
     1        3        3
     2        5        5

Specified Points

     j    xl[j]    yl[j]
     0        1      1.5
     1      1.5        2

Function Values

  z[i,j]
               j=0      j=1      j=2
 i=     0        2      0.5        1
 i=     1        4        1        2
 i=     2        3     0.75      1.5

 ** Output **

ierr =      0

Estimated Function Values in (x,y)

  fl[i,j]
               j=0      j=1
 i=     0     1.44        1
 i=     1        2     1.39
```

6.3.2 ASL_dgidyb, ASL_rgidyb
混合偏微分値と双 3 次スプライン係数

(1) **機 能**

格子上のデータより双 3 次スプライン係数を求め, 任意点での混合偏微分値を計算する. なお, 端条件は “not-a-knot” 条件, 節点位置は標本点と一致させる.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dgidyb  (x, nx, y, ny, z, xl, yl, dl, c, wk);
```

単精度関数:

```
ierr = ASL_rgidyb  (x, nx, y, ny, z, xl, yl, dl, c, 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	x	{D∗ / R∗}	nx	入 力	標本点の X 座標値 x[i − 1], i = 1, ··· , nx (x[i − 1] < x[i], i ≠ nx)
2	nx	I	1	入 力	X 方向標本点数
3	y	{D∗ / R∗}	ny	入 力	標本点の Y 座標値 y[j − 1], j = 1, ··· , ny (y[j − 1] < y[j], j ≠ ny)
4	ny	I	1	入 力	Y 方向標本点数
5	z	{D∗ / R∗}	nx×ny	入 力	標本点 (x[i − 1], y[j − 1]) での関数値 z_{ij} からなる配列: z[(i − 1) + nx × (j − 1)] = z_{ij}
6	xl	{D / R}	1	入 力	混合偏微分値を計算する点の X 座標値
7	yl	{D / R}	1	入 力	混合偏微分値を計算する点の Y 座標値
8	dl	{D∗ / R∗}	6	出 力	求められた双 3 次スプライン関数 $f(x, y)$ の 点 (x, y)=(xl, yl) での値と混合偏微分値 dl [0] =$f(x, y)$, dl [1] =$\partial f/\partial x$, dl [2] =$\partial f/\partial y$, dl [3] =$\partial^2 f/(\partial x \partial y)$ dl [4] =$\partial^2 f/(\partial x)^2$ dl [5] =$\partial^2 f/(\partial y)^2$
9	c	{D∗ / R∗}	4×nx×ny	出 力	双 3 次スプライン係数 (注意事項 (a) 参照)
10	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 **大きさ**: 2 × nx × ny + 2 × max(nx, ny)
11	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $\mathrm{nx} \geq 4, \mathrm{ny} \geq 4$

(b) $\mathrm{x}[0] < \mathrm{x}[1] < \cdots < \mathrm{x}[\mathrm{nx}-1]$ (昇順)

(c) $\mathrm{y}[0] < \mathrm{y}[1] < \cdots < \mathrm{y}[\mathrm{ny}-1]$ (昇順)

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	偏微分値を計算する点 (xl, yl) が補間領域外である.	境界での双 3 次スプライン係数を利用し, 補外した値を出力する.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	

(6) **注意事項**

(a) この関数で求める双 3 次スプライン係数は, $\alpha_{e,r}^{i,j}$ の値でなく, z(i, j), $\mathrm{z_x}$(i, j), $\mathrm{z_y}$(i, j), $\mathrm{z_{xy}}$(i, j) の値である (6.1.2 参照).

(b) 補間値, 偏微分値, または 2 重積分値をさらに続けて求めたい場合は, この関数を使用した後, 6.3.5 $\left\{\begin{matrix}\mathrm{ASL\_dgisbx}\\ \mathrm{ASL\_rgisbx}\end{matrix}\right\}$, 6.3.6 $\left\{\begin{matrix}\mathrm{ASL\_dgidby}\\ \mathrm{ASL\_rgidby}\end{matrix}\right\}$ または 6.3.7 $\left\{\begin{matrix}\mathrm{ASL\_dgiibz}\\ \mathrm{ASL\_rgiibz}\end{matrix}\right\}$ をそれぞれ呼び出せばよい. この時, 配列 x, y, c および変数 nx, ny の内容はそのまま後続の関数の対応する引数の入力とする. このようにすれば, 双 3 次スプライン係数の計算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.

(7) **使用例**

(a) 問 題

格子座標値 $(\mathrm{x_i}, \mathrm{y_j}) = (2 \times \mathrm{i} - 1, 2 \times \mathrm{j} - 1)$ $(\mathrm{i} = 1, 2, 3, 4;\ \mathrm{j} = 1, 2, 3, 4)$ での関数値 $\mathrm{z_{ij}}$ からなる行列 Z が

$$Z = \begin{bmatrix} 2.0 & 0.5 & 1.0 & 3.5 \\ 4.0 & 1.0 & 2.0 & 7.0 \\ 3.0 & 0.75 & 1.5 & 5.25 \\ -1.0 & -0.25 & -0.5 & -1.75 \end{bmatrix}$$

と与えられたとき, これらを標本点として双 3 次スプライン関数で近似し, 指定点での双 3 次スプライン関数の偏微分値を求める.

(b) 入力データ

$\mathrm{x}[\mathrm{i}-1] = \mathrm{x_i}(\mathrm{i} = 1, \cdots, \mathrm{nx})$, $\mathrm{y}[\mathrm{j}-1] = \mathrm{y_j}(\mathrm{j} = 1, \cdots, \mathrm{ny})$, $\mathrm{z}[(\mathrm{i}-1) + \mathrm{nx} \times (\mathrm{j}-1)] = \mathrm{z_{ij}}$ nx=4, ny=4, xl=1.5, yl=1.5

(c) 主プログラム

```
/*      C interface example for ASL_dgidyb */

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

int main()
{
    double *x;
    int nx;
    double *y;
    int ny;
    int nwk;
    double *f;
    double xl;
```

```
double yl;
double ds[6];
double *c;
double *wk;
int ierr;
int i,j,k,l;
FILE *fp;

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

printf( "    *** ASL_dgidyb ***\n" );
printf( "\n     ** Input **\n\n" );
nx=4;
ny=4;
xl=1.5;
yl=1.5;

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

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

nwk = 2 * (nx * ny + ( (nx>ny)?nx:ny ));
wk = ( double * )malloc((size_t)( sizeof(double) * nwk ));
if( wk == NULL )
{
    printf( "no enough memory for array wk\n" );
    return -1;
}

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

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


printf( "\tnx  = %6d\n", nx  );
printf( "\tny  = %6d\n", ny  );
printf( "\n\tSpecified Points   xl = %8.3g\n", xl );
printf( "\tSpecified Points   yl = %8.3g\n", yl );
for( i=0 ; i<nx ; i++ )
{
    fscanf( fp, "%lf", &x[i] );
}
for( i=0 ; i<nx ; i++ )
{
    fscanf( fp, "%lf", &y[i] );
}
for( j=0 ; j<ny ; j++ )
{
    for( i=0 ; i<nx ; i++ )
    {
        fscanf( fp, "%lf", &f[i+nx*j] );
    }
}

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

printf( "\n" );
printf( "\tFunction Values\n\n" );
printf( "\t  z[i,j]\n");
printf( "\t\t            j=0          j=1          j=2          j=3\n" );
for( i=0 ; i<nx ; i++ )
{
    printf( "\t  i=%6d",i );
    for( j=0 ; j<ny ; j++ )
    {
```

```
            printf( "     %8.3g", f[i+nx*j] );
        }
        printf( "\n" );
    }

    fclose( fp );

    ierr = ASL_dgidyb(x, nx, y, ny, f, xl, yl, ds, c, wk);

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

    printf( "\tThe Partial Derivatives\n\n" );
    for( i=0 ; i<6 ; i++ )
    {
        printf( "\t  dl[%6d]=%8.3g\n", i,ds[i] );
    }

    printf( "\n" );
    printf( "\tSpline Coefficients\n" );
    printf( "\n" );
    for( l=0 ; l<4 ; l++ )
    {
        for( k=0 ; k<2 ; k++ )
        {
            for( j=0 ; j<4 ; j++ )
            {
    printf( "\t  c[     0,%6d,%6d,%6d]=%8.3g  ",j,k,l,c[  2*(j+nx*(k+2*l))] );
    printf( "  c[     1,%6d,%6d,%6d]=%8.3g\n",j,k,l,c[1+2*(j+nx*(k+2*l))] );
            }
        }
        printf( "\n" );
    }

    free( x );
    free( y );
    free( f );
    free( c );
    free( wk );


    return 0;
}
```

(d) 出力結果

```
*** ASL_dgidyb ***

 ** Input **

nx  =      4
ny  =      4

Specified Points   xl =      1.5
Specified Points   yl =      1.5

Coordinates (x,y)

     i                x[i]               y[i]
     0                  1                  1
     1                  3                  3
     2                  5                  5
     3                  7                  7

Function Values

  z[i,j]
                j=0          j=1          j=2          j=3
  i=     0           2          0.5            1          3.5
  i=     1           4            1            2            7
  i=     2           3         0.75          1.5         5.25
  i=     3          -1        -0.25         -0.5        -1.75

 ** Output **

ierr =      0

The Partial Derivatives

  dl[     0]=        2
  dl[     1]=    0.988
  dl[     2]=    -1.39
  dl[     3]=   -0.688
  dl[     4]=   -0.539
  dl[     5]=    0.695

Spline Coefficients

  c[     0,     0,     0,     0]=        2    c[     1,     0,     0,     0]=     1.75
  c[     0,     1,     0,     0]=        4    c[     1,     1,     0,     0]=     0.25
  c[     0,     2,     0,     0]=        3    c[     1,     2,     0,     0]=    -1.25
  c[     0,     3,     0,     0]=       -1    c[     1,     3,     0,     0]=    -2.75
  c[     0,     0,     1,     0]=    -1.25    c[     1,     0,     1,     0]=    -1.09
  c[     0,     1,     1,     0]=     -2.5    c[     1,     1,     1,     0]=   -0.156
  c[     0,     2,     1,     0]=    -1.88    c[     1,     2,     1,     0]=    0.781
  c[     0,     3,     1,     0]=    0.625    c[     1,     3,     1,     0]=     1.72
```

```
c[     0,     0,     0,     1]=     0.5    c[     1,     0,     0,     1]=   0.438
c[     0,     1,     0,     1]=       1    c[     1,     1,     0,     1]=  0.0625
c[     0,     2,     0,     1]=    0.75    c[     1,     2,     0,     1]=  -0.313
c[     0,     3,     0,     1]=   -0.25    c[     1,     3,     0,     1]=  -0.688
c[     0,     0,     1,     1]=   -0.25    c[     1,     0,     1,     1]=  -0.219
c[     0,     1,     1,     1]=    -0.5    c[     1,     1,     1,     1]= -0.0313
c[     0,     2,     1,     1]=  -0.375    c[     1,     2,     1,     1]=   0.156
c[     0,     3,     1,     1]=   0.125    c[     1,     3,     1,     1]=   0.344

c[     0,     0,     0,     2]=       1    c[     1,     0,     0,     2]=   0.875
c[     0,     1,     0,     2]=       2    c[     1,     1,     0,     2]=   0.125
c[     0,     2,     0,     2]=     1.5    c[     1,     2,     0,     2]=  -0.625
c[     0,     3,     0,     2]=    -0.5    c[     1,     3,     0,     2]=   -1.38
c[     0,     0,     1,     2]=    0.75    c[     1,     0,     1,     2]=   0.656
c[     0,     1,     1,     2]=     1.5    c[     1,     1,     1,     2]=  0.0938
c[     0,     2,     1,     2]=    1.13    c[     1,     2,     1,     2]=  -0.469
c[     0,     3,     1,     2]=  -0.375    c[     1,     3,     1,     2]=   -1.03

c[     0,     0,     0,     3]=     3.5    c[     1,     0,     0,     3]=    3.06
c[     0,     1,     0,     3]=       7    c[     1,     1,     0,     3]=   0.438
c[     0,     2,     0,     3]=    5.25    c[     1,     2,     0,     3]=   -2.19
c[     0,     3,     0,     3]=   -1.75    c[     1,     3,     0,     3]=   -4.81
c[     0,     0,     1,     3]=    1.75    c[     1,     0,     1,     3]=    1.53
c[     0,     1,     1,     3]=     3.5    c[     1,     1,     1,     3]=   0.219
c[     0,     2,     1,     3]=    2.63    c[     1,     2,     1,     3]=   -1.09
c[     0,     3,     1,     3]=  -0.875    c[     1,     3,     1,     3]=   -2.41
```

6.3.3 ASL_dgiizb, ASL_rgiizb
2 重積分値

(1) **機 能**

格子上のデータより得られた双 3 次スプライン関数の 2 重積分値を求める. なお, 節点位置は標本点と一致させる.

(2) **使用法**

倍精度関数:

ierr = ASL_dgiizb (x, nx, y, ny, z, ax, bx, cy, dy, & q, wk);

単精度関数:

ierr = ASL_rgiizb (x, nx, y, ny, z, ax, bx, cy, dy, & q, 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	x	{D∗ / R∗}	nx	入 力	標本点の X 座標値 $x[i-1]$, $i=1,\cdots,nx$ $(x[i-1]<x[i], i\neq nx)$
2	nx	I	1	入 力	X 方向標本点数
3	y	{D∗ / R∗}	ny	入 力	標本点の Y 座標値 $y[j-1]$, $j=1,\cdots,ny$ $(y[j-1]<y[j], j\neq ny)$
4	ny	I	1	入 力	Y 方向標本点数
5	z	{D∗ / R∗}	nx×ny	入 力	標本点 $(x[i-1], y[j-1])$ での関数値 z_{ij} からなる配列: $z[(i-1)+nx\times(j-1)]=z_{ij}$
6	ax	{D / R}	1	入 力	X 方向の積分範囲下端
7	bx	{D / R}	1	入 力	X 方向の積分範囲上端
8	cy	{D / R}	1	入 力	Y 方向の積分範囲下端
9	dy	{D / R}	1	入 力	Y 方向の積分範囲上端
10	q	{D∗ / R∗}	1	出 力	直方領域 $[ax,bx]\times[cy,dy]$ での双 3 次スプライン関数の 2 重積分値
11	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 **大きさ**: $(ny+5)\times nx+ny-1+\max(5\times nx-4, 5\times ny-2)$
12	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $\mathrm{nx} \geq 2, \mathrm{ny} \geq 2$

(b) $\mathrm{x}[0] < \mathrm{x}[1] < \cdots < \mathrm{x}[\mathrm{nx}-1]$ (昇順)

(c) $\mathrm{y}[0] < \mathrm{y}[1] < \cdots < \mathrm{y}[\mathrm{ny}-1]$ (昇順)

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	直方領域 $[\mathrm{ax}, \mathrm{bx}] \times [\mathrm{cy}, \mathrm{dy}]$ が補間領域外にわたっている.	境界での双 3 次スプライン係数を利用し, 補外した関数について積分値を出力する.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) または (c) を満足しなかった.	

(6) **注意事項**

(a) この関数では双 3 次スプライン係数の出力は行わない.

(b) 6.3.7 $\left\{\begin{matrix}\mathrm{ASL_dgiibz}\\ \mathrm{ASL_rgiibz}\end{matrix}\right\}$とは, 計算方法が異なるため, 同じ結果が得られるとは限らない.

(7) **使用例**

(a) 問 題

格子座標値 $(\mathrm{x_i}, \mathrm{y_j}) = (\mathrm{i}-1, \mathrm{j}-1)$ $(\mathrm{i}=1,2,3,4;\ \mathrm{j}=1,2,3,4)$ での関数値 $\mathrm{z_{ij}}$ からなる行列 Z が

$$Z = \begin{bmatrix} 8.0 & 7.0 & 6.0 & 5.0 \\ 2.0 & 3.0 & 4.0 & 5.0 \\ -4.0 & -1.0 & 2.0 & 5.0 \\ -10.0 & -5.0 & 0.0 & 5.0 \end{bmatrix}$$

と与えられたとき, これらを標本点として双 3 次スプライン関数で近似し, 直方領域 $[0.5, 2.5] \times [0.5, 2.5]$ での双 3 次スプライン関数の 2 重積分値を求める.

(b) 入力データ

$\mathrm{x}[\mathrm{i}-1] = \mathrm{x_i}(\mathrm{i}=1,\cdots,\mathrm{nx})$, $\mathrm{y}[\mathrm{j}-1] = \mathrm{y_j}(\mathrm{j}=1,\cdots,\mathrm{ny})$, $\mathrm{z}[(\mathrm{i}-1)+\mathrm{nx}\times(\mathrm{j}-1)] = \mathrm{z_{ij}}$ nx=4, ny=4, ax=0.5, bx=2.5, cy=0.5, dy=2.5

(c) 主プログラム

```
/*      C interface example for ASL_dgiizb */

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

int main()
{
    double *x;
    int nx;
    double *y;
    int ny;
    double *f;
    double ax;
    double bx;
    double cy;
    double dy;
    double  q;
    double *wk;
    int ierr;
    int i,j,nwk;
    FILE *fp;

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

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

printf( "    *** ASL_dgiizb ***\n" );
printf( "\n     ** Input **\n\n" );
nx=4;
ny=4;
ax=0.5;
bx=2.5;
cy=0.5;
dy=2.5;

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

nwk=(ny+5)*nx+ny-1+( (5*nx-4>5*ny-2) ? 5*nx-4 : 5*ny-2 );
wk = ( double * )malloc((size_t)( sizeof(double) * nwk ));
if( wk == NULL )
{
    printf( "no enough memory for array wk\n" );
    return -1;
}

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

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

printf( "\tnx  = %6d\n", nx  );
printf( "\tny  = %6d\n", ny  );
printf( "\n\tLimits of Integration   ax=%8.3g\n", ax );
printf( "\t                        bx=%8.3g\n", bx );
printf( "\t                        cy=%8.3g\n", cy );
printf( "\t                        dy=%8.3g\n", dy );
for( i=0 ; i<nx ; i++ )
{
    fscanf( fp, "%lf", &x[i] );
}
for( i=0 ; i<nx ; i++ )
{
    fscanf( fp, "%lf", &y[i] );
}
for( j=0 ; j<ny ; j++ )
{
    for( i=0 ; i<nx ; i++ )
    {
        fscanf( fp, "%lf", &f[i+nx*j] );
    }
}

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

printf( "\n" );
printf( "\tFunction Values\n\n" );
printf( "\t  z[i,j]\n");
printf( "\t                j=0      j=1      j=2      j=3\n" );
for( i=0 ; i<nx ; i++ )
{
    printf( "\t i=%6d ",i );
    for( j=0 ; j<ny ; j++ )
    {
        printf( " %8.3g", f[i+nx*j] );
    }
    printf( "\n" );
}

fclose( fp );

ierr = ASL_dgiizb(x, nx, y, ny, f, ax, bx, cy, dy, &q, wk);

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

```
        printf( "\n" );

        printf( "\tDouble Integral\n\n" );
        printf( "\t  q=%8.3g\n",q );

        free( x );
        free( y );
        free( f );
        free( wk );

        return 0;
    }
```

(d) 出力結果

```
        *** ASL_dgiizb ***

         ** Input **

        nx  =      4
        ny  =      4

        Limits of Integration   ax=     0.5
                                bx=     2.5
                                cy=     0.5
                                dy=     2.5

        Coordinates (x,y)

             i      x[i]    y[i]
             0        0       0
             1        1       1
             2        2       2
             3        3       3

        Function Values

          z[i,j]
                          j=0      j=1      j=2      j=3
         i=     0          8        7        6        5
         i=     1          2        3        4        5
         i=     2         -4       -1        2        5
         i=     3        -10       -5        0        5

         ** Output **

        ierr =      0

        Double Integral

          q=       8
```

6.3.4 ASL_dgicbp, ASL_rgicbp
双 3 次スプライン係数

(1) **機 能**

格子上のデータより双 3 次スプライン係数を求める．端条件は “not-a-knot” 条件とする．なお, 節点位置は標本点と一致させる．

(2) **使用法**

倍精度関数:

```
ierr = ASL_dgicbp  (x, nx, y, ny, z, c, wk);
```

単精度関数:

```
ierr = ASL_rgicbp  (x, nx, y, ny, z, c, 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	x	{D∗ / R∗}	nx	入 力	標本点の X 座標値 $x[i-1]$, $i = 1, \cdots, nx$ $(x[i-1] < x[i], i \neq nx)$
2	nx	I	1	入 力	X 方向標本点数
3	y	{D∗ / R∗}	ny	入 力	標本点の Y 座標値 $y[j-1]$, $j = 1, \cdots, ny$ $(y[j-1] < y[j], j \neq ny)$
4	ny	I	1	入 力	Y 方向標本点数
5	z	{D∗ / R∗}	nx×ny	入 力	標本点 $(x[i-1], y[j-1])$ での関数値 z_{ij} からなる配列: $z[(i-1) + nx \times (j-1)] = z_{ij}$
6	c	{D∗ / R∗}	4×nx×ny	出 力	双 3 次スプライン係数 (注意事項 (a) 参照)
7	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 **大きさ**: $2 \times nx \times ny + 2 \times \max(nx, ny)$
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) $nx \geq 4, ny \geq 4$

(b) $x[0] < x[1] < \cdots < x[nx-1]$ (昇順)

(c) $y[0] < y[1] < \cdots < y[ny-1]$ (昇順)

(5) **エラーインディケータ (戻り値)**

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

(6) **注意事項**

(a) この関数で求める双 3 次スプライン係数は, $\alpha_{e,r}^{i,j}$ の値でなく, z(i, j), z_x(i, j), z_y(i, j), z_{xy}(i, j) の値である (6.1.2 参照).

6.3.5 ASL_dgisbx, ASL_rgisbx
双 3 次スプライン係数による補間値

(1) **機 能**

与えられた双 3 次スプライン係数から双 3 次スプライン関数の補間値を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dgisbx  (x, nx, y, ny, c, xl, yl, & fl);
```

単精度関数:

```
ierr = ASL_rgisbx  (x, nx, y, ny, c, xl, yl, & fl);
```

(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∗}	nx	入 力	節点の X 座標値 $x[i-1], i = 1, \cdots, nx$ $(x[i-1] < x[i], i \neq nx)$
2	nx	I	1	入 力	X 方向節点数
3	y	{D∗ / R∗}	ny	入 力	節点の Y 座標値 $y[j-1], j = 1, \cdots, ny$ $(y[j-1] < y[j], j \neq ny)$
4	ny	I	1	入 力	Y 方向節点数
5	c	{D∗ / R∗}	4×nx×ny	入 力	双 3 次スプライン係数 (注意事項 (a) 参照)
6	xl	{D / R}	1	入 力	補間値を計算する点の X 座標値
7	yl	{D / R}	1	入 力	補間値を計算する点の Y 座標値
8	fl	{D∗ / R∗}	1	出 力	点 (xl, yl) での双 3 次スプライン関数の値
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) $nx \geq 4, ny \geq 4$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	補間点 (xl, yl) が補間領域外である.	境界での双 3 次スプライン係数を利用し, 補外した値を出力する.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.

(6) **注意事項**

(a) この関数に入力する双 3 次スプライン係数は, $\alpha_{e,r}^{i,j}$ の値でなく, z(i, j), z_x(i, j), z_y(i, j), z_{xy}(i, j) の値である (6.1.2 参照).

(b) この関数は特定の点での双 3 次スプライン補間値を求める. 与えられた格子点上の全ての点に対する双 3 次スプライン補間値を求めたい場合には, 6.3.1 $\left\{\begin{matrix}\text{ASL_dgisxb}\\ \text{ASL_rgisxb}\end{matrix}\right\}$を使用した方が効率がよい.

6.3.6 ASL_dgidby, ASL_rgidby
双 3 次スプライン係数による混合偏微分値

(1) **機 能**

与えられた双 3 次スプライン係数から双 3 次スプライン関数の混合偏微分値を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dgidby  (x, nx, y, ny, c, xl, yl, dl);
```

単精度関数:

```
ierr = ASL_rgidby  (x, nx, y, ny, c, xl, yl, dl);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	x	{D∗ / R∗}	nx	入 力	節点の X 座標値 $x[i-1], i = 1, \cdots, nx$ $(x[i-1] < x[i], i \neq nx)$
2	nx	I	1	入 力	X 方向節点数
3	y	{D∗ / R∗}	ny	入 力	節点の Y 座標値 $y[j-1], j = 1, \cdots, ny$ $(y[j-1] < y[j], j \neq ny)$
4	ny	I	1	入 力	Y 方向節点数
5	c	{D∗ / R∗}	4×nx×ny	入 力	双 3 次スプライン係数 (注意事項 (a) 参照)
6	xl	{D / R}	1	入 力	混合偏微分値を計算する点の X 座標値
7	yl	{D / R}	1	入 力	混合偏微分値を計算する点の Y 座標値
8	dl	{D∗ / R∗}	6	出 力	双 3 次スプライン関数 $f(x, y)$ の点 (x, y)=(xl, yl) での値と混合偏微分値 dl [0] $=f$, dl [1] $=\partial f/\partial x$, dl [2] $=\partial f/\partial y$, dl [3] $=\partial^2 f/(\partial x \partial y)$ dl [4] $=\partial^2 f/(\partial x)^2$ dl [5] $=\partial^2 f/(\partial y)^2$
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) $nx \geq 4, ny \geq 4$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	偏微分値を計算する点 (xl, yl) が補間領域外である.	境界での双 3 次スプライン係数を利用し, 補外した値を出力する.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.

(6) **注意事項**

(a) この関数に入力する双 3 次スプライン係数は, $\alpha_{e,r}^{i,j}$ の値でなく, z(i, j), z_x(i, j), z_y(i, j), z_{xy}(i, j) の値である (6.1.2 参照).

6.3.7 ASL_dgiibz, ASL_rgiibz
双 3 次スプライン係数による 2 重積分値

(1) **機 能**

与えられた双 3 次スプライン係数から双 3 次スプライン関数の直方領域での積分値を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dgiibz  (x, nx, y, ny, c, ax, bx, cy, dy, & q);
```

単精度関数:

```
ierr = ASL_rgiibz  (x, nx, y, ny, c, ax, bx, cy, dy, & q);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	x	{D∗ / R∗}	nx	入 力	節点の X 座標値 $x[i-1], i = 1, \cdots, nx$ $(x[i-1] < x[i], i \neq nx)$
2	nx	I	1	入 力	X 方向節点数
3	y	{D∗ / R∗}	ny	入 力	節点の Y 座標値 $y[j-1], j = 1, \cdots, ny$ $(y[j-1] < y[j], j \neq ny)$
4	ny	I	1	入 力	Y 方向節点数
5	c	{D∗ / R∗}	4×nx×ny	入 力	双 3 次スプライン係数 (注意事項 (a) 参照)
6	ax	{D / R}	1	入 力	X 方向の積分範囲下端
7	bx	{D / R}	1	入 力	X 方向の積分範囲上端
8	cy	{D / R}	1	入 力	Y 方向の積分範囲下端
9	dy	{D / R}	1	入 力	Y 方向の積分範囲上端
10	q	{D∗ / R∗}	1	出 力	直方領域 $[ax, bx] \times [cy, dy]$ での双 3 次スプライン関数の 2 重積分値
11	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $nx \geq 4, ny \geq 4$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	直方領域 $[\mathrm{ax}, \mathrm{bx}] \times [\mathrm{cy}, \mathrm{dy}]$ が補間領域外にわたっている.	境界での双 3 次スプライン係数を利用し, 補外した関数について積分値を出力する.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.

(6) **注意事項**

(a) この関数に入力する双 3 次スプライン係数は, $\alpha_{e,r}^{i,j}$ の値でなく, z(i, j), $\mathrm{z_x}(\mathrm{i,j})$, $\mathrm{z_y}(\mathrm{i,j})$, $\mathrm{z_{xy}}(\mathrm{i,j})$ の値である (6.1.2 参照).

(b) 6.3.3 $\left\{\begin{matrix}\mathrm{ASL_dgiizb}\\ \mathrm{ASL_rgiizb}\end{matrix}\right\}$とは, 計算方法が異なるため, 同じ結果が得られるとは限らない.

6.4 平面データの補間

6.4.1 ASL_dgispo, ASL_rgispo 開曲線補間

(1) **機 能**

入力データ点列が開曲線 (open curve) の形をしている場合の平面データの補間を行う. 節点位置は標本点と一致させる.

(2) **使用法**

倍精度関数:

ierr = ASL_dgispo (x, y, n, is, ie, m, xo, yo, wk);

単精度関数:

ierr = ASL_rgispo (x, y, n, is, ie, m, xo, yo, 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	x	{D∗ / R∗}	n	入 力	標本点 $(x[i-1], y[i-1]), i = 1, \cdots, n$ の横座標値 $x[i-1]$ (補間する順に入力)
2	y	{D∗ / R∗}	n	入 力	標本点の縦座標値, あるいは, 関数値 $y[i-1]$
3	n	I	1	入 力	標本点の数
4	is	I	1	入 力	補間始点の標本点番号 補間始点は $(x[is-1], y[is-1])$
5	ie	I	1	入 力	補間終点の標本点番号 補間終点は $(x[ie-1], y[ie-1])$
6	m	I	1	入 力	補間データ数 (=分割数+1)
7	xo	{D∗ / R∗}	m	出 力	得られた補間点の横座標値 $xo[i-1]$
8	yo	{D∗ / R∗}	m	出 力	得られた補間点の縦座標値 $yo[i-1]$
9	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 **大きさ:** $4 \times n + m - 3$
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) $m \geq 2$

(c) $1 \leq is \leq n$

(d) $1 \leq ie \leq n$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	補間点の X 座標が補間区間外となった.	区間外については補外を行う.
3000	制限条件 (a), (b), (c) または (d) を満足しなかった.	処理を打ち切る.

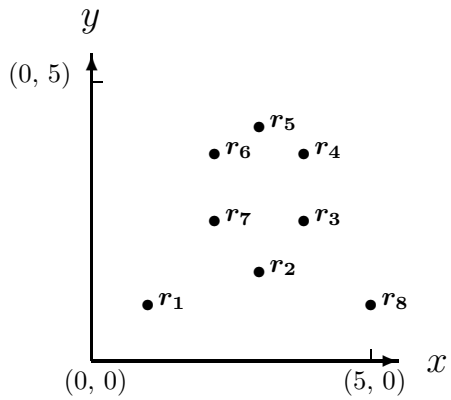
(6) **注意事項**

(a) 補間点は等間隔になるとは限らない.

(7) **使用例**

(a) 問 題

下図のようなデータ点が与えられたとき $\boldsymbol{r_1} \to \boldsymbol{r_2} \to \boldsymbol{r_3} \to \boldsymbol{r_4} \to \boldsymbol{r_5} \to \boldsymbol{r_6} \to \boldsymbol{r_7} \to \boldsymbol{r_2} \to \boldsymbol{r_8}$ 間を 20 点で補間する. ここで, $\boldsymbol{r_1} = (1.0, 1.0), \boldsymbol{r_2} = (3.0, 1.6), \boldsymbol{r_3} = (3.8, 2.5), \boldsymbol{r_4} = (3.8, 3.7), \boldsymbol{r_5} = (3.0, 4.2), \boldsymbol{r_6} = (2.2, 3.7), \boldsymbol{r_7} = (2.2, 2.5), \boldsymbol{r_8} = (5.0, 1.0)$



(b) 入力データ

配列 x, y, n = 9, is = 1, ie = 9, m = 20

(c) 主プログラム

```
/*      C interface example for ASL_dgispo */

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

int main()
{
    double *x;
    double *y;
    int nx;
    int is;
    int ie;
    int m;
    double *xo;
    double *yo;
    double *wk;
    int ierr;
    int i,nwk;
    FILE *fp;

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

    printf( "    *** ASL_dgispo ***\n" );
    printf( "\n     ** Input **\n\n" );
    nx=9;
    m=20;
```

```
    is=1;
    ie=9;

    nwk=4*nx+m-3;
    wk = ( double * )malloc((size_t)( sizeof(double) * nwk ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

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

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

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

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

    printf( "\tn   = %6d\n", nx  );
    printf( "\tis  = %6d\n", is  );
    printf( "\tie  = %6d\n", ie  );
    printf( "\tm   = %6d\n", m   );
    for( i=0 ; i<nx ; i++ )
    {
        fscanf( fp, "%lf", &x[i] );
    }
    for( i=0 ; i<nx ; i++ )
    {
        fscanf( fp, "%lf", &y[i] );
    }

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

    fclose( fp );

    ierr = ASL_dgispo(x, y, nx, is, ie, m, xo, yo, wk);

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

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

    free( x );
    free( y );
    free( xo );
    free( yo );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dgispo ***

 ** Input **

n   =      9
is  =      1
ie  =      9
m   =     20

Coordinates (x,y)
```

```
      i                x[i]               y[i]
      0                   1                  1
      1                   3                1.6
      2                 3.8                2.5
      3                 3.8                3.7
      4                   3                4.2
      5                 2.2                3.7
      6                 2.2                2.5
      7                   3                1.6
      8                   5                  1

 ** Output **

ierr =      0

      i               xo[i]              yo[i]
      0                   1                  1
      1                1.54               1.06
      2                2.11                1.2
      3                2.66               1.41
      4                3.17               1.72
      5                3.58               2.12
      6                3.85               2.64
      7                3.94               3.22
      8                3.75               3.77
      9                3.29               4.14
     10                2.71               4.14
     11                2.25               3.77
     12                2.06               3.22
     13                2.15               2.64
     14                2.42               2.12
     15                2.83               1.72
     16                3.34               1.41
     17                3.89                1.2
     18                4.46               1.06
     19                   5                  1
```

6.4.2 ASL_dgispr, ASL_rgispr
閉曲線補間

(1) **機 能**

入力データ点列が閉曲線 (closed curve) の形をしている場合の平面データの補間を行う．節点位置は標本点と一致させる．

(2) **使用法**

倍精度関数:

ierr = ASL_dgispr (x, y, n, is, ie, m, xo, yo, wk);

単精度関数:

ierr = ASL_rgispr (x, y, n, is, ie, m, xo, yo, 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	x	{D∗ / R∗}	n	入 力	標本点 $(x[i-1], y[i-1]), i = 1, \cdots, n$ の横座標値 $x[i-1]$ (補間する順に入力)
2	y	{D∗ / R∗}	n	入 力	標本点の縦座標値, あるいは, 関数値 $y[i-1]$
3	n	I	1	入 力	標本点の数
4	is	I	1	入 力	補間始点の標本点番号 補間始点は $(x[is-1], y[is-1])$
5	ie	I	1	入 力	補間終点の標本点番号 補間終点は $(x[ie-1], y[ie-1])$
6	m	I	1	入 力	補間データ数 (=分割数+1)
7	xo	{D∗ / R∗}	m	出 力	得られた補間点の横座標値 $xo[i-1]$
8	yo	{D∗ / R∗}	m	出 力	得られた補間点の縦座標値 $yo[i-1]$
9	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 **大きさ:** $9 \times n + m - 3$
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) $n \geq 4$

(b) $m \geq 2$

(c) $1 \leq is \leq n$

(d) $1 \leq ie \leq n$

(e) $x[0] = x[n-1], y[0] = y[n-1]$

(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	補間点の X 座標が補間区間外となった.	区間外については補外を行う.
2000	制限条件 (e) を満足しなかった.	(x[n − 1], y[n − 1]) を (x [0] , y [0]) と同じとみなして処理を行う.
3000	制限条件 (a), (b), (c) または (d) を満足しなかった.	処理を打ち切る.

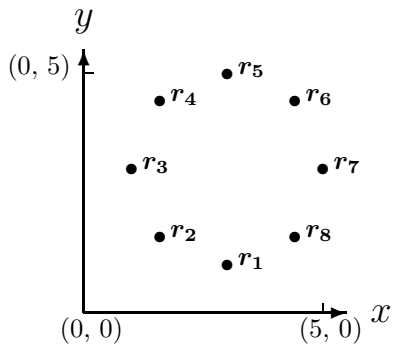
(6) **注意事項**

(a) 補間点は等間隔になるとは限らない.

(7) **使用例**

(a) 問 題

下図のようなデータ点が与えられたとき $\boldsymbol{r_1} \to \boldsymbol{r_2} \to \boldsymbol{r_3} \to \boldsymbol{r_4} \to \boldsymbol{r_5} \to \boldsymbol{r_6} \to \boldsymbol{r_7} \to \boldsymbol{r_8} \to \boldsymbol{r_1}$ 間を 20 点で補間する. ここで, $\boldsymbol{r_1} = (3.0, 1.0), \boldsymbol{r_2} = (1.5858, 1.5858), \boldsymbol{r_3} = (1.0, 3.0), \boldsymbol{r_4} = (1.5858, 4.4142), \boldsymbol{r_5} = (3.0, 5.0), \boldsymbol{r_6} = (4.4142, 4.4142), \boldsymbol{r_7} = (5.0, 3.0), \boldsymbol{r_8} = (4.4142, 1.5858)$



(b) 入力データ

配列 x, y, n = 9, is = 1, ie = 9, m = 20

(c) 主プログラム

```
/*      C interface example for ASL_dgispr */

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

int main()
{
    double *x;
    double *y;
    int nx;
    int is;
    int ie;
    int m;
    double *xo;
    double *yo;
    double *wk;
    int ierr;
    int i,nwk;
    FILE *fp;

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

    printf( "    *** ASL_dgispr ***\n" );
    printf( "\n     ** Input **\n\n" );
    nx=9;
```

```
    m=20;
    is=1;
    ie=9;

    nwk=9*nx+m-3;
    wk = ( double * )malloc((size_t)( sizeof(double) * nwk ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

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

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

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

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

    printf( "\tn   = %6d\n", nx  );
    printf( "\tis  = %6d\n", is  );
    printf( "\tie  = %6d\n", ie  );
    printf( "\tm   = %6d\n", m   );
    for( i=0 ; i<nx ; i++ )
    {
        fscanf( fp, "%lf", &x[i] );
    }
    for( i=0 ; i<nx ; i++ )
    {
        fscanf( fp, "%lf", &y[i] );
    }

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

    fclose( fp );

    ierr = ASL_dgispr(x, y, nx, is, ie, m, xo, yo, wk);

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

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

    free( x );
    free( y );
    free( xo );
    free( yo );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dgispr ***

 ** Input **

n   =      9
is  =      1
ie  =      9
m   =     20

Coordinates (x,y)
```

```
      i               x[i]              y[i]
      0                  3                 1
      1               1.59              1.59
      2                  1                 3
      3               1.59              4.41
      4                  3                 5
      5               4.41              4.41
      6                  5                 3
      7               4.41              1.59
      8                  3                 1

 ** Output **

ierr =      0

      i              xo[i]             yo[i]
      0                  3                 1
      1               2.35              1.11
      2               1.77              1.42
      3               1.33              1.91
      4               1.06              2.51
      5               1.01              3.16
      6               1.17               3.8
      7               1.53              4.35
      8               2.05              4.76
      9               2.67              4.97
     10               3.33              4.97
     11               3.95              4.76
     12               4.47              4.35
     13               4.83               3.8
     14               4.99              3.16
     15               4.94              2.51
     16               4.67              1.91
     17               4.23              1.42
     18               3.65              1.11
     19                  3                 1
```

6.4.3 ASL_dgisso, ASL_rgisso
開曲線平滑化補間

(1) **機 能**

入力データ点列が開曲線 (open curve) の形をしている場合の平面データの平滑化補間を行う. 節点の横座標は標本点の横座標と一致させる.

(2) **使用法**

倍精度関数:

ierr = ASL_dgisso (x, y, n, is, ie, m, xo, yo, wk);

単精度関数:

ierr = ASL_rgisso (x, y, n, is, ie, m, xo, yo, 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	x	{D∗ / R∗}	n	入 力	標本点 $(x[i-1], y[i-1]), i = 1, \cdots, n$ の横座標値 $x[i-1]$ (補間する順に入力)
2	y	{D∗ / R∗}	n	入 力	標本点の縦座標値, あるいは, 関数値 $y[i-1]$
3	n	I	1	入 力	標本点の数
4	is	I	1	入 力	補間始点の標本点番号 補間始点は $(x[is-1], y[is-1])$
5	ie	I	1	入 力	補間終点の標本点番号 補間終点は $(x[ie-1], y[ie-1])$
6	m	I	1	入 力	補間データ数 (=分割数+1)
7	xo	{D∗ / R∗}	m	出 力	得られた補間点の横座標値 $xo[i-1]$
8	yo	{D∗ / R∗}	m	出 力	得られた補間点の縦座標値 $yo[i-1]$
9	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 **大きさ:** $2 \times n \times n + 9 \times n + m - 3$
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) $n \geq 4$

(b) $m \geq 2$

(c) $1 \leq is \leq n$

(d) $1 \leq ie \leq n$

(e) $x[0] < x[1] < \cdots < x[n-1]$ (昇順)

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	補間点の X 座標が補間区間外となった.	区間外については補外を行う.
3000	制限条件 (a), (b), (c) または (d) を満足しなかった.	処理を打ち切る.
3010	制限条件 (e) を満足しなかった.	
4000	クロスバリデーションの最小値が見つからない (データが無相関である).	

(6) **注意事項**

(a) 補間点は等間隔になるとは限らない.

(7) **使用例**

(a) 問 題

$$\begin{cases} x_i = 7.5S_i^3 - 11.5S_i^2 + 5S_i + e_{xi} \\ y_i = -0.67S_i^3 - S_i^2 + 2S_i + e_{yi} \end{cases} \quad (i = 1, \cdots, 15)$$

$$S_i = 0.06 \times i$$

e_{xi}, e_{yi} は平均値 0, 標準偏差 0.05 の正規分布乱数とする.

(b) 入力データ

n=15, is=1, ie=10, m=10

(c) 主プログラム

```
/*      C interface example for ASL_dgisso */

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

int main()
{
    double *x;
    double *y;
    int nx;
    int is;
    int ie;
    int m;
    double *xo;
    double *yo;
    double *wk;
    int ierr;
    double *rn;
    double  am;
    double  sg;
    double  si;
    double  esi;
    int ix;
    int iy;
    int i,ni,nwk;

    printf( "    *** ASL_dgisso ***\n" );
    printf( "\n     ** Input **\n\n" );
    nx=15;
    m=10;
    ix=1;
    iy=1;
    am=0.0;
    sg=0.05;

    nwk=(2*nx+9)*nx+m-3;
    wk = ( double * )malloc((size_t)( sizeof(double) * nwk ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }
```

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

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

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

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

    ni=50;
    rn = ( double * )malloc((size_t)( sizeof(double) * ni ));
    if( rn == NULL )
    {
        printf( "no enough memory for array rn\n" );
        return -1;
    }


    ierr = ASL_djdbno( ni, am, sg, &ix, &iy, rn);
    si=0.06;
    esi=0.06;
    for( i=0 ; i<15 ; i++ )
    {
        x[i]=((7.5*si-11.5)*si+5.0)*si+rn[2*i];
        y[i]=((-0.67*si-1.0)*si+2.0)*si+rn[2*i+1];
        si += esi;
    }
    is=1;
    ie=10;

    printf( "\tn   = %6d\n", nx  );
    printf( "\tis  = %6d\n", is  );
    printf( "\tie  = %6d\n", ie  );
    printf( "\tm   = %6d\n", m   );

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

    ierr = ASL_dgisso(x, y, nx, is, ie, m, xo, yo, wk);


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

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

    free( x );
    free( y );
    free( xo );
    free( yo );
    free( wk );
    free( rn );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dgisso ***

 ** Input **

n   =     15
is  =      1
ie  =     10
m   =     10

Coordinates (x,y)

     i               x[i]              y[i]
     0              0.164             0.131
     1              0.517             0.218
     2              0.517             0.336
     3              0.661             0.432
     4              0.574             0.455
     5              0.654             0.501
     6              0.648             0.692
     7              0.625             0.702
     8              0.478             0.761
     9              0.449              0.73
    10              0.505             0.628
    11               0.35             0.539
    12               0.44             0.628
    13              0.564             0.697
    14              0.646             0.439

 ** Output **

ierr =      0

     i              xo[i]             yo[i]
     0              0.181              0.13
     1              0.308             0.129
     2              0.424             0.165
     3              0.516             0.272
     4              0.583             0.389
     5              0.624             0.443
     6              0.636             0.502
     7              0.618             0.638
     8              0.564             0.744
     9              0.495             0.732
```

6.4.4 ASL_dgissr, ASL_rgissr
閉曲線平滑化補間

(1) **機 能**

入力データ点列が閉曲線 (closed curve) の形をしている場合の平面データの平滑化補間を行う．節点の横座標は標本点の横座標と一致させる．

(2) **使用法**

倍精度関数:

ierr = ASL_dgissr (x, y, n, is, ie, m, xo, yo, wk);

単精度関数:

ierr = ASL_rgissr (x, y, n, is, ie, m, xo, yo, 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	x	{D∗ / R∗}	n	入 力	標本点 $(x[i-1], y[i-1]), i = 1, \cdots, n$ の横座標値 $x[i-1]$ (補間する順に入力)
2	y	{D∗ / R∗}	n	入 力	標本点の縦座標値, あるいは, 関数値 $y[i-1]$
3	n	I	1	入 力	標本点の数
4	is	I	1	入 力	補間始点の標本点番号 補間始点は $(x[is-1], y[is-1])$
5	ie	I	1	入 力	補間終点の標本点番号 補間終点は $(x[ie-1], y[ie-1])$
6	m	I	1	入 力	補間データ数 (=分割数+1)
7	xo	{D∗ / R∗}	m	出 力	得られた補間点の横座標値 $xo[i-1]$
8	yo	{D∗ / R∗}	m	出 力	得られた補間点の縦座標値 $yo[i-1]$
9	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 **大きさ**: $2 \times n \times n + 9 \times n + m - 3$
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) $n \geq 4$

(b) $m \geq 2$

(c) $1 \leq is \leq n$

(d) $1 \leq ie \leq n$

(e) $x[0] < x[1] < \cdots < x[n-1]$ (昇順)

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	X 座標が平滑化後補間区間外となった.	区間外については補外を行う.
3000	制限条件 (a), (b), (c) または (d) を満足しなかった.	処理を打ち切る.
3010	制限条件 (e) を満足しなかった.	
4000	クロスバリデーションの最小値が見つからない (データが無相関である).	

(6) **注意事項**

(a) 補間点は等間隔になるとは限らない.

(b) 標本点 (x [0] , y [0]) と標本点 (x[n − 1], y[n − 1]) とは誤差の範囲で一致している必要がある.

(7) **使用例**

(a) 問 題

$$\begin{cases} x_i = \sin(S_i) + e_{xi} \\ y_i = \cos(S_i) + e_{yi} \end{cases} \quad (i = 1, \cdots, 16)$$

$$S_i = 0.418879 \times (i - 1)$$

e_{xi}, e_{yi} は平均値 0, 標準偏差 0.05 の正規分布乱数とする.

(b) 入力データ

n=16, is=1, ie=16, m=12

(c) 主プログラム

```
/*      C interface example for ASL_dgissr */

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

int main()
{
    double *x;
    double *y;
    int nx;
    int is;
    int ie;
    int m;
    double *xo;
    double *yo;
    double *wk;
    int ierr;
    double *rn;
    double  am;
    double  sg;
    double  si;
    double  esi;
    int ix;
    int iy;
    int i,ni,nwk;

    printf( "    *** ASL_dgissr ***\n" );
    printf( "\n     ** Input **\n\n" );
    nx=16;
    m=12;
    ix=1;
    iy=1;
    am=0.0;
    sg=0.05;

    nwk=(2*nx+9)*nx+m-3;
    wk = ( double * )malloc((size_t)( sizeof(double) * nwk ));
    if( wk == NULL )
    {
```

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

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

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

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

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

    ni=40;
    rn = ( double * )malloc((size_t)( sizeof(double) * ni ));
    if( rn == NULL )
    {
        printf( "no enough memory for array rn\n" );
        return -1;
    }


    ierr = ASL_djdbno( ni, am, sg, &ix, &iy, rn);
    si=0.0;
    esi=0.418879;
    for( i=0 ; i<16 ; i++ )
    {
        x[i]=sin(si)+rn[2*i];
        y[i]=cos(si)+rn[2*i+1];
        si += esi;
    }
    is=1;
    ie=16;

    printf( "\tn   = %6d\n", nx  );
    printf( "\tis  = %6d\n", is  );
    printf( "\tie  = %6d\n", ie  );
    printf( "\tm   = %6d\n", m   );

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


    ierr = ASL_dgissr(x, y, nx, is, ie, m, xo, yo, wk);


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

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

    free( x );
    free( y );
    free( xo );
    free( yo );
    free( wk );
    free( rn );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dgissr ***

 ** Input **

n   =     16
is  =      1
ie  =     16
m   =     12

Coordinates (x,y)

     i               x[i]              y[i]
     0            -0.0959              1.01
     1              0.476             0.907
     2              0.689             0.681
     3               0.97             0.328
     4              0.901            -0.141
     5              0.861            -0.558
     6              0.609            -0.731
     7              0.253            -0.932
     8             -0.258              -0.9
     9             -0.618            -0.774
    10             -0.808            -0.563
    11              -1.08            -0.237
    12             -0.973             0.303
    13              -0.71             0.788
    14             -0.413             0.851
    15             0.0551             0.928

 ** Output **

ierr =      0

     i              xo[i]             yo[i]
     0             -0.019             0.981
     1               0.47              0.88
     2               0.86             0.476
     3              0.952           -0.0832
     4              0.754            -0.617
     5              0.279            -0.912
     6             -0.303            -0.896
     7             -0.804              -0.6
     8              -1.05             -0.11
     9             -0.927             0.463
    10             -0.523             0.831
    11             -0.019             0.981
```

6.5 B–スプライン

6.5.1 ASL_dgicbs, ASL_rgicbs
B–スプラインの計算

(1) **機 能**

節点 $\xi_{1-m}, \xi_{2-m}, \cdots, \xi_{n+m}$ および B–スプラインの階数 (次数 +1) m が与えられたとき, $\xi_0 \le x \le \xi_{n+1}$ となる x に対して零でない m 個の B–スプラインの値を計算する.

(2) **使用法**

倍精度関数:

ierr = ASL_dgicbs (xd, xk, n, m, aryn, & knot, wk);

単精度関数:

ierr = ASL_rgicbs (xd, xk, n, m, aryn, & knot, 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	xd	{D / R}	1	入 力	標本点 x.
2	xk	{D∗ / R∗}	n + 2 × m	入 力	節点 (付加節点も含む) ξ_i.
3	n	I	1	入 力	内部の節点の数 n.
4	m	I	1	入 力	B–スプラインの階数 m.
5	aryn	{D∗ / R∗}	m	出 力	正規化された B–スプライン $N_{mi}(x)$.
6	knot	I*	1	出 力	零でない B–スプラインの位置.
7	wk	{D∗ / R∗}	m+1	ワーク	m 階の B–スプライン $M_{mi}(x)$.
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) n ≥ 1

(b) m ≥ 1

(c) xk[m − 1] ≤ xd ≤ xk[m + n]

(d) xk[0] ≤ xk[1] ≤ ⋯ ≤ xk[n + 2 × m − 1]

(e) xk[i − 1] < xk[i + m − 1]　(i = 1, 2, ⋯, 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) 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.	

(6) **注意事項**

(a) 節点の値によっては, 異なった B–スプラインが得られる場合がある.

(7) **使用例**

(a) 問 題

節点 $\xi_i = \{0, 0, 0, 0, 1, 2, 3, 5, 6, 7, 7, 7, 7\}$ $(i = -3, -2, \cdots, 9)$ および B–スプラインの階数 $m = 4$ が与えられたとき, $x = 4$ に対して零でない 4 個の B–スプラインの値を求める.

(b) 入力データ

xd=4, 節点 xk, n=5, m=4.

(c) 主プログラム

```
/*      C interface example for ASL_dgicbs */

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

int main()
{
    double xd;
    double *xk;
    int n;
    int m;
    double *aryn;
    int knot;
    double *arym;
    int ierr;
    int i;
    FILE *fp;

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

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

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

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

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

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

```
    }

    printf( "\tn = %6d m = %6d\n", n, m );
    printf( "\txd = %8.3g\n", xd);

    for( i=0 ; i<n+2*m ; i++ )
    {
        fscanf( fp, "%lf", &xk[i] );
        printf( "\txk[%6d] = %8.3g\n", i, xk[i] );
    }

    fclose( fp );

    ierr = ASL_dgicbs(xd, xk, n, m, aryn, &knot, arym);

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

    printf( "\tValues of normalized B-spline\n\n" );
    for( i=0 ; i<m ; i++ )
    {
        printf( "\taryn[%6d] = %8.3g\n", i, aryn[i] );
    }

    free( xk );
    free( aryn );
    free( arym );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dgicbs ***

 ** Input **

n =      5 m =      4
xd =      0.4
xk[     0] =        0
xk[     1] =        0
xk[     2] =        0
xk[     3] =        0
xk[     4] =      0.1
xk[     5] =      0.2
xk[     6] =      0.3
xk[     7] =      0.5
xk[     8] =      0.6
xk[     9] =      0.7
xk[    10] =      0.7
xk[    11] =      0.7
xk[    12] =      0.7

 ** Output **

ierr =      0

Values of normalized B-spline

aryn[     0] =   0.0417
aryn[     1] =    0.458
aryn[     2] =    0.458
aryn[     3] =   0.0417
```

6.5.2 ASL_dgisi1, ASL_rgisi1
B–スプラインを用いた補間 (1 次元データ)

(1) **機 能**

標本点 x_i $(i = 1, 2, \cdots, N)$ において与えられたデータ f_i $(i = 1, 2, \cdots, N)$ を補間する $(m-1)$ 次のスプライン関数

$$S(x) \quad = \quad \sum_{i=1}^{n+m} c_i N_{mi}(x)$$

を求め, 指定された点における補間値を計算する.

(2) **使用法**

倍精度関数:

ierr = ASL_dgisi1 (xd, nd, fd, xk, m, xx, nn, s, wk, iwk);

単精度関数:

ierr = ASL_rgisi1 (xd, nd, fd, xk, m, xx, nn, s, wk, iwk);

(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	xd	{D∗ / R∗}	nd	入 力	標本点 x_i.
2	nd	I	1	入 力	標本点の数 N.
3	fd	{D∗ / R∗}	nd	入 力	標本点 x_i において与えられたデータ f_i.
4	xk	{D∗ / R∗}	nd + m	入 力	節点 (付加節点も含む) ξ_i.
5	m	I	1	入 力	B–スプラインの階数 m.
6	xx	{D∗ / R∗}	nn	入 力	補間値を計算する x 座標.
7	nn	I	1	入 力	補間値を計算する点の個数.
8	s	{D∗ / R∗}	nn	出 力	$x =$ xx[i − 1] における近似関数の値 $S(x)$ (i = 1, 2, ⋯, nn).
9	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 **大きさ** : $\mathrm{nd}^2 + \mathrm{nd} + 2 \times \mathrm{m} + 1$
10	iwk	I*	nd	ワーク	作業領域.
11	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $\mathrm{nd} \geq 1$

(b) $\mathrm{nd} - \mathrm{m} \geq 1, \mathrm{m} \geq 1$

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

(d) $\mathrm{xd}[0] < \mathrm{xd}[1] < \cdots < \mathrm{xd}[\mathrm{nd}-1]$

(e) $\mathrm{xk}[\mathrm{m}-1] \leq \mathrm{xd}[\mathrm{i}-1] \leq \mathrm{xk}[\mathrm{nd}]$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{nd})$

(f) $\mathrm{xk}[0] \leq \mathrm{xk}[1] \leq \cdots \leq \mathrm{xk}[\mathrm{nd}+\mathrm{m}-1]$

(g) $\mathrm{xk}[\mathrm{i}-1] < \mathrm{xk}[\mathrm{i}+\mathrm{m}-1]$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{nd})$

(h) 標本値 $\mathrm{xd}[\mathrm{i}-1]$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{nd})$ は Schoenberg–Whitney の条件を満たす (6.1.2 アルゴリズム参照).

(i) $\mathrm{xk}[\mathrm{m}-1] \leq \mathrm{xx}[\mathrm{i}-1] \leq \mathrm{xk}[\mathrm{nd}]$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{nn})$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
2100	正規方程式を解く際の LU 分解において, 対角要素が 0 に近いものがあった. 精度の良い結果が得られない場合がある.	処理を続ける.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
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.	
3400	制限条件 (d) を満足しなかった.	
3500	制限条件 (e) を満足しなかった.	
3600	制限条件 (f) を満足しなかった.	
3700	制限条件 (g) を満足しなかった.	
3800	制限条件 (h) を満足しなかった.	
3900	制限条件 (i) を満足しなかった.	
4000	正規方程式が解けない.	

(6) **注意事項**

(a) 節点の値によっては, 異なった B–スプラインが得られる場合がある.

(b) 内部の節点の数は nd − m となる.

(7) **使用例**

(a) 問 題

次のデータ

$$f_i = \frac{1}{0.01 + (x_i - 0.3)^2} \quad (x_i = 0.0, 0.1, \cdots, 1.0)$$

を補間して, $x : 0.05, 0.10, \cdots, 0.95$ に対する補間値を求める.

(b) 入力データ

標本点 (xd, fd), nd=11, 節点 xk, m=4, 補間値を計算する座標 xx, nn=19.

(c) 主プログラム

```
/*      C interface example for ASL_dgisi1 */

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

int main()
{
    double *xd;
    int nd;
    double *fd;
    double *xk;
    int m;
    double *xx;
    int nn;
    double *s;
    double *wk;
    int *iwk;
    int ierr;
    int i;
    FILE *fp;

    fp = fopen( "dgisi1.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dgisi1 ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &nd );
    fscanf( fp, "%d", &m );
    fscanf( fp, "%d", &nn );

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

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

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

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

    xx = ( double * )malloc((size_t)( sizeof(double) * nn ));
    if( xx == NULL )
    {
        printf( "no enough memory for array xx\n" );
```

```
        return -1;
    }

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

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

    printf( "\tnd = %6d m = %6d nn = %6d\n",nd,m,nn );

    printf( "\n\n\txd\n\n");
    for( i=0 ; i<nd ; i++ )
    {
        xd[i] = 0.1*(double)i;
        printf( "\txd[%6d] = %8.3g\n", i, xd[i] );
        fd[i] = 1.0/(0.01+(xd[i]-0.3)*(xd[i]-0.3));
    }

    printf( "\n\n\txk\n\n");
    for( i=0 ; i<nd+m ; i++ )
    {
        fscanf( fp, "%lf", &xk[i] );
        printf( "\txk[%6d] = %8.3g\n", i, xk[i] );
    }

    for( i=0 ; i<nn ; i++ )
    {
        xx[i] = 0.05*(double)(i+1);
    }


    fclose( fp );

    ierr = ASL_dgisi1(xd, nd, fd, xk, m, xx, nn, s, wk, iwk);

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

    printf( "\tValues of approximating spline function on sample points\n\n" );
    printf( "\t       xx  \t       s\n\n" );
    for( i=0 ; i<nn ; i++ )
    {
        printf( "\t%8.3g\t%8.3g\n", xx[i], s[i] );
    }

    free( iwk );
    free( xd );
    free( fd );
    free( xk );
    free( xx );
    free( s );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dgisi1 ***

 ** Input **

nd =     11 m =      4 nn =     19


xd

xd[     0] =        0
xd[     1] =      0.1
xd[     2] =      0.2
xd[     3] =      0.3
xd[     4] =      0.4
xd[     5] =      0.5
xd[     6] =      0.6
xd[     7] =      0.7
xd[     8] =      0.8
xd[     9] =      0.9
xd[    10] =        1


xk

xk[     0] =        0
xk[     1] =        0
xk[     2] =        0
xk[     3] =        0
xk[     4] =      0.2
xk[     5] =      0.3
```

```
xk[     6] =      0.4
xk[     7] =      0.5
xk[     8] =      0.6
xk[     9] =      0.7
xk[    10] =      0.8
xk[    11] =        1
xk[    12] =        1
xk[    13] =        1
xk[    14] =        1

 ** Output **

ierr =      0

Values of approximating spline function on sample points

       xx          s

    0.05        15.7
     0.1          20
    0.15        29.3
     0.2          50
    0.25        82.2
     0.3         100
    0.35          82
     0.4          50
    0.45        29.7
     0.5          20
    0.55          14
     0.6          10
    0.65        7.48
     0.7        5.88
    0.75        4.72
     0.8        3.85
    0.85        3.19
     0.9         2.7
    0.95        2.32
```

6.5.3 ASL_dgisi2, ASL_rgisi2
B–スプラインを用いた補間 (2 次元データ)

(1) **機 能**

標本点 (x_i, y_j) $(i = 1, 2, \cdots, N_x; j = 1, 2, \cdots, N_y)$ において与えられたデータ f_{ij} $(i = 1, 2, \cdots, N_x; j = 1, 2, \cdots, N_y)$ を補間するスプライン関数

$$S(x, y) \quad = \quad \sum_{i=1}^{h+m} \sum_{j=1}^{k+n} c_{ij} N_{mi}(x) N_{nj}(y)$$

を求め, 指定された格子点における補間値を計算する.

(2) **使用法**

倍精度関数:

ierr = ASL_dgisi2 (xd, nxd, yd, nyd, fd, xk, mx, yk, my, xx, nnx, yy, nny, s, wk, iwk);

単精度関数:

ierr = ASL_rgisi2 (xd, nxd, yd, nyd, fd, xk, mx, yk, my, xx, nnx, yy, nny, s, wk, iwk);

(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	xd	{D∗ / R∗}	nxd	入 力	x 方向の標本点 x_i.
2	nxd	I	1	入 力	x 方向の標本点の数 N_x.
3	yd	{D∗ / R∗}	nyd	入 力	y 方向の標本点 y_j.
4	nyd	I	1	入 力	y 方向の標本点の数 N_y.
5	fd	{D∗ / R∗}	nxd×nyd	入 力	標本点 (x_i, y_j) において与えられたデータ f_{ij}.
6	xk	{D∗ / R∗}	nxd + mx	入 力	x 方向の節点 (付加節点も含む) ξ_i.
7	mx	I	1	入 力	x 方向の B–スプラインの階数 m.
8	yk	{D∗ / R∗}	nyd + my	入 力	y 方向の節点 (付加節点も含む) ζ_j.
9	my	I	1	入 力	y 方向の B–スプラインの階数 n.
10	xx	{D∗ / R∗}	nnx	入 力	補間値を計算する x 座標.
11	nnx	I	1	入 力	補間値を計算する x 方向の点の個数.
12	yy	{D∗ / R∗}	nny	入 力	補間値を計算する y 座標.
13	nny	I	1	入 力	補間値を計算する y 方向の点の個数.
14	s	{D∗ / R∗}	nnx × nny		(x, y) = (xx[i − 1], yy[j − 1]) における近似関数の値 $S(x, y)$ (i = 1, 2, ⋯, nnx ; j = 1, 2, ⋯, nny).
15	wk	{D∗ / R∗}	内容参照	ワーク	作業領域. **大きさ** : $\mathrm{nxd}^2 \times \mathrm{nyd}^2 + \mathrm{nxd} \times (\mathrm{nyd} + \mathrm{mx}) + 2 \times \mathrm{mx} + 2 \times \mathrm{my} + 2$
16	iwk	I*	内容参照	ワーク	作業領域. **大きさ** : nxd × nyd + nxd + nyd
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) $\mathrm{nxd} \geq 1, \mathrm{nyd} \geq 1$

(b) $\mathrm{nxd} - \mathrm{mx} \geq 1, \mathrm{nyd} - \mathrm{my} \geq 1, \mathrm{mx} \geq 1, \mathrm{my} \geq 1$

(c) $\mathrm{nnx} \geq 1, \mathrm{nny} \geq 1$

(d) $\mathrm{xd}[0] < \mathrm{xd}[1] < \cdots < \mathrm{xd}[\mathrm{nxd} - 1]$

(e) $\mathrm{yd}[0] < \mathrm{yd}[1] < \cdots < \mathrm{yd}[\mathrm{nyd} - 1]$

(f) $\mathrm{xk}[\mathrm{mx} - 1] \leq \mathrm{xd}[\mathrm{i} - 1] \leq \mathrm{xk}[\mathrm{nxd}]$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{nxd})$

(g) $\mathrm{yk}[\mathrm{my} - 1] \leq \mathrm{yd}[\mathrm{j} - 1] \leq \mathrm{yk}[\mathrm{nyd}]$ $(\mathrm{j} = 1, 2, \cdots, \mathrm{nyd})$

(h) $\mathrm{xk}[0] \leq \mathrm{xk}[1] \leq \cdots \leq \mathrm{xk}[\mathrm{nxd} + \mathrm{mx} - 1]$

(i) $\mathrm{yk}[0] \leq \mathrm{yk}[1] \leq \cdots \leq \mathrm{yk}[\mathrm{nyd} + \mathrm{my} - 1]$

(j) $\mathrm{xk}[\mathrm{i} - 1] < \mathrm{xk}[\mathrm{i} + \mathrm{mx} - 1]$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{nxd})$

(k) $\mathrm{yk}[\mathrm{j} - 1] < \mathrm{yk}[\mathrm{j} + \mathrm{my} - 1]$ $(\mathrm{j} = 1, 2, \cdots, \mathrm{nyd})$

(l) 標本値 $\mathrm{xd}[\mathrm{i} - 1]$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{nxd})$ は Schoenberg–Whitney の条件を満たす (6.1.2 アルゴリズム参照).

(m) 標本値 $\mathrm{yd}[\mathrm{j} - 1]$ $(\mathrm{j} = 1, 2, \cdots, \mathrm{nyd})$ は Schoenberg–Whitney の条件を満たす (6.1.2 アルゴリズム参照).

(n) $\mathrm{xk}[\mathrm{mx} - 1] \leq \mathrm{xx}[\mathrm{i} - 1] \leq \mathrm{xk}[\mathrm{nxd}]$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{nnx})$

(o) $\mathrm{yk}[\mathrm{my} - 1] \leq \mathrm{yy}[\mathrm{j} - 1] \leq \mathrm{yk}[\mathrm{nyd}]$ $(\mathrm{j} = 1, 2, \cdots, \mathrm{nny})$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
2100	正規方程式を解く際の LU 分解において, 対角要素が 0 に近いものがあった. 精度の良い結果が得られない場合がある.	処理を続ける.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
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.	
3400	制限条件 (d) を満足しなかった.	
3410	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.	
3510	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.	
3610	制限条件 (i) を満足しなかった.	
3700	制限条件 (j) を満足しなかった.	
3710	制限条件 (k) を満足しなかった.	
3800	制限条件 (l) を満足しなかった.	
3810	制限条件 (m) を満足しなかった.	
3900	制限条件 (n) を満足しなかった.	
3910	制限条件 (o) を満足しなかった.	
4000	正規方程式が解けない.	

(6) **注意事項**

(a) 節点の値によっては, 異なった B–スプラインが得られる場合がある.

(b) 内部の節点の数は (nxd − mx, nyd − my) となる

(7) **使用例**

(a) 問 題

次のデータ

$$f_{ij} = \frac{1}{0.01 + (x_i - 0.3)^2} + \frac{1}{0.02 + (y_j - 0.4)^2}$$
$$(x_i = 0.0, 0.1, \cdots, 1.0; y_j = 0.0, 0.1, \cdots, 1.0)$$

を補間して, $x : 0.0, 0.1, \cdots, 0.9; y : 0.0, 0.1, \cdots, 0.9$ に対する補間値を求める.

(b) 入力データ

標本点 (xd, yd, fd), nxd=11, nyd=11, x 方向の節点 xk, mx=4, y 方向の節点 yk, my=4, 補間値を計算する x 座標 xx, nnx=10, 補間値を計算する y 座標 yy, nny=10.

(c) 主プログラム

```
/*      C interface example for ASL_dgisi2 */

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

int main()
{
    double *xd;
    int nxd;
    double *yd;
    int nyd;
    double *fd;
    double *xk;
    int mx;
    double *yk;
    int my;
    double *xx;
    int nnx;
    double *yy;
    int nny;
    double *s;
    double *wk;
    int *iwk;
    int ierr;
    int i,j;
    FILE *fp;

    fp = fopen( "dgisi2.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dgisi2 ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &nxd );
    fscanf( fp, "%d", &mx );
    fscanf( fp, "%d", &nnx );
    fscanf( fp, "%d", &nyd );
    fscanf( fp, "%d", &my );
    fscanf( fp, "%d", &nny );

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

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

    yd = ( double * )malloc((size_t)( sizeof(double) * nyd ));
    if( yd == NULL )
```

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

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

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

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

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

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

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

wk = ( double * )malloc((size_t)( sizeof(double)
    * (nxd*nxd*nyd*nyd+nxd*(nyd+mx)
       +mx*2+my*2+2) ));
if( wk == NULL )
{
    printf( "no enough memory for array wk\n" );
    return -1;
}

printf( "\tnxd = %6d mx = %6d nnx = %6d\n",
    nxd, mx, nnx);
printf( "\tnyd = %6d my = %6d nny = %6d\n",
    nyd, my, nny);

printf( "\n\txd\n\n" );
for( i=0 ; i<nxd ; i++ )
{
    xd[i] = 0.1 * (double)i;
    printf( "\txd[%6d] = %8.3g\n", i, xd[i] );
}

printf( "\n\tyd\n\n" );
for( j=0 ; j<nyd ; j++ )
{
    yd[j] = 0.1 * (double)j;
    printf( "\tyd[%6d] = %8.3g\n", j, yd[j] );
}

for( j=0 ; j<nyd ; j++ )
{
    for( i=0 ; i<nxd ; i++ )
    {
        fd[i+nxd*j]=
         1.0/(0.01+(xd[i]-0.3)*(xd[i]-0.3))
        +1.0/(0.02+(yd[j]-0.4)*(yd[j]-0.4));
    }
}

printf( "\n\txk\n\n" );
for( i=0 ; i<nxd+mx ; i++ )
{
    fscanf( fp, "%lf", &xk[i] );
    printf( "\txk[%6d] = %8.3g\n", i, xk[i] );
}

printf( "\n\tyk\n\n" );
for( j=0 ; j<nyd+my ; j++ )
{
    fscanf( fp, "%lf", &yk[j] );
```

```
        printf( "\tyk[%6d] = %8.3g\n", j, yk[j] );
    }

    for( i=0 ; i<nnx ; i++ )
    {
        xx[i] = 0.1 * (double)i;
    }

    for( j=0 ; j<nny ; j++ )
    {
        yy[j] = 0.1 * (double)j;
    }

    fclose( fp );

    ierr = ASL_dgisi2(xd, nxd, yd, nyd, fd, xk, mx, yk, my, xx, nnx, yy,
             nny, s, wk, iwk);

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

    printf( "\n\n\tValues of an approximating spline function on sample points\n\n" );
    printf( "\n\n\t      xx   \t      yy   \t     s\n\n" );
    for( i=0 ; i<nnx ; i++ )
    {
        for( j=0 ; j<nny ; j++ )
        {
            printf( "\t%8.3g\t%8.3g\t%8.3g\n",
            xx[i], yy[j], s[i+nnx*j] );
        }
    }

    free( iwk );
    free( xd );
    free( yd );
    free( fd );
    free( xk );
    free( yk );
    free( xx );
    free( yy );
    free( s );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dgisi2 ***

 ** Input **

nxd =     11 mx =      4 nnx =     10
nyd =     11 my =      4 nny =     10

xd

xd[     0] =        0
xd[     1] =      0.1
xd[     2] =      0.2
xd[     3] =      0.3
xd[     4] =      0.4
xd[     5] =      0.5
xd[     6] =      0.6
xd[     7] =      0.7
xd[     8] =      0.8
xd[     9] =      0.9
xd[    10] =        1

yd

yd[     0] =        0
yd[     1] =      0.1
yd[     2] =      0.2
yd[     3] =      0.3
yd[     4] =      0.4
yd[     5] =      0.5
yd[     6] =      0.6
yd[     7] =      0.7
yd[     8] =      0.8
yd[     9] =      0.9
yd[    10] =        1

xk

xk[     0] =        0
xk[     1] =        0
xk[     2] =        0
xk[     3] =        0
xk[     4] =      0.2
xk[     5] =      0.3
xk[     6] =      0.4
xk[     7] =      0.5
xk[     8] =      0.6
xk[     9] =      0.7
xk[    10] =      0.8
```

```
xk[    11] =        1
xk[    12] =        1
xk[    13] =        1
xk[    14] =        1

yk

yk[     0] =        0
yk[     1] =        0
yk[     2] =        0
yk[     3] =        0
yk[     4] =      0.2
yk[     5] =      0.3
yk[     6] =      0.4
yk[     7] =      0.5
yk[     8] =      0.6
yk[     9] =      0.7
yk[    10] =      0.8
yk[    11] =        1
yk[    12] =        1
yk[    13] =        1
yk[    14] =        1

 ** Output **

ierr =      0

Values of an approximating spline function on sample points


      xx          yy          s

       0           0        15.6
       0         0.1        19.1
       0         0.2        26.7
       0         0.3        43.3
       0         0.4          60
       0         0.5        43.3
       0         0.6        26.7
       0         0.7        19.1
       0         0.8        15.6
       0         0.9        13.7
     0.1           0        25.6
     0.1         0.1        29.1
     0.1         0.2        36.7
     0.1         0.3        53.3
     0.1         0.4          70
     0.1         0.5        53.3
     0.1         0.6        36.7
     0.1         0.7        29.1
     0.1         0.8        25.6
     0.1         0.9        23.7
     0.2           0        55.6
     0.2         0.1        59.1
     0.2         0.2        66.7
     0.2         0.3        83.3
     0.2         0.4         100
     0.2         0.5        83.3
     0.2         0.6        66.7
     0.2         0.7        59.1
     0.2         0.8        55.6
     0.2         0.9        53.7
     0.3           0         106
     0.3         0.1         109
     0.3         0.2         117
     0.3         0.3         133
     0.3         0.4         150
     0.3         0.5         133
     0.3         0.6         117
     0.3         0.7         109
     0.3         0.8         106
     0.3         0.9         104
     0.4           0        55.6
     0.4         0.1        59.1
     0.4         0.2        66.7
     0.4         0.3        83.3
     0.4         0.4         100
     0.4         0.5        83.3
     0.4         0.6        66.7
     0.4         0.7        59.1
     0.4         0.8        55.6
     0.4         0.9        53.7
     0.5           0        25.6
     0.5         0.1        29.1
     0.5         0.2        36.7
     0.5         0.3        53.3
     0.5         0.4          70
     0.5         0.5        53.3
     0.5         0.6        36.7
     0.5         0.7        29.1
     0.5         0.8        25.6
     0.5         0.9        23.7
     0.6           0        15.6
     0.6         0.1        19.1
     0.6         0.2        26.7
     0.6         0.3        43.3
```

```
0.6         0.4          60
0.6         0.5        43.3
0.6         0.6        26.7
0.6         0.7        19.1
0.6         0.8        15.6
0.6         0.9        13.7
0.7           0        11.4
0.7         0.1          15
0.7         0.2        22.5
0.7         0.3        39.2
0.7         0.4        55.9
0.7         0.5        39.2
0.7         0.6        22.5
0.7         0.7          15
0.7         0.8        11.4
0.7         0.9        9.59
0.8           0         9.4
0.8         0.1        12.9
0.8         0.2        20.5
0.8         0.3        37.2
0.8         0.4        53.8
0.8         0.5        37.2
0.8         0.6        20.5
0.8         0.7        12.9
0.8         0.8         9.4
0.8         0.9        7.55
0.9           0        8.26
0.9         0.1        11.8
0.9         0.2        19.4
0.9         0.3          36
0.9         0.4        52.7
0.9         0.5          36
0.9         0.6        19.4
0.9         0.7        11.8
0.9         0.8        8.26
0.9         0.9        6.41
```

6.5.4 ASL_dgisi3, ASL_rgisi3
B–スプラインを用いた補間 (3 次元データ)

(1) **機 能**

標本点 (x_i, y_j, z_k) $(i = 1, 2, \cdots, N_x; j = 1, 2, \cdots, N_y; k = 1, 2, \cdots, N_z)$ において与えられたデータ f_{ijk} $(i = 1, 2, \cdots, N_x; j = 1, 2, \cdots, N_y; k = 1, 2, \cdots, N_z)$ を補間するスプライン関数

$$S(x, y, z) \quad = \quad \sum_{i=1}^{n_x+m_x} \sum_{j=1}^{n_y+m_y} \sum_{k=1}^{n_z+m_z} c_{ijk} N_{m_x i}(x) N_{m_y j}(y) N_{m_z k}(z)$$

を求め, 指定された格子点における補間値を計算する.

(2) **使用法**

倍精度関数:

ierr = ASL_dgisi3 (xd, nxd, yd, nyd, zd, nzd, fd, xk, mx, yk, my, zk, mz, xx, nnx, yy, nny, zz, nnz, s, wk, iwk);

単精度関数:

ierr = ASL_rgisi3 (xd, nxd, yd, nyd, zd, nzd, fd, xk, mx, yk, my, zk, mz, xx, nnx, yy, nny, zz, nnz, s, 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	xd	{D∗ / R∗}	nxd	入 力	x 方向の標本点 x_i.
2	nxd	I	1	入 力	x 方向の標本点の数 N_x.
3	yd	{D∗ / R∗}	nyd	入 力	y 方向の標本点 y_j.
4	nyd	I	1	入 力	y 方向の標本点の数 N_y.
5	zd	{D∗ / R∗}	nzd	入 力	z 方向の標本点 z_k.
6	nzd	I	1	入 力	z 方向の標本点の数 N_z.
7	fd	{D∗ / R∗}	内容参照	入 力	標本点 (x_i, y_j, z_k) において与えられたデータ f_{ijk}. **大きさ:** nxd×nyd×nzd

項番	引数と戻り値	型	大きさ	入出力	内　　容
8	xk	$\left\{\begin{matrix}\mathrm{D}* \\ \mathrm{R}*\end{matrix}\right\}$	nxd + mx	入 力	x 方向の節点 (付加節点も含む) ξ_i.
9	mx	I	1	入 力	x 方向の B–スプラインの階数 m_x.
10	yk	$\left\{\begin{matrix}\mathrm{D}* \\ \mathrm{R}*\end{matrix}\right\}$	nyd + my	入 力	y 方向の節点 (付加節点も含む) ζ_j.
11	my	I	1	入 力	y 方向の B–スプラインの階数 m_y.
12	zk	$\left\{\begin{matrix}\mathrm{D}* \\ \mathrm{R}*\end{matrix}\right\}$	nzd + mz	入 力	z 方向の節点 (付加節点も含む) η_k.
13	mz	I	1	入 力	z 方向の B–スプラインの階数 m_z.
14	xx	$\left\{\begin{matrix}\mathrm{D}* \\ \mathrm{R}*\end{matrix}\right\}$	nnx	入 力	補間値を計算する x 座標.
15	nnx	I	1	入 力	補間値を計算する x 方向の点の個数.
16	yy	$\left\{\begin{matrix}\mathrm{D}* \\ \mathrm{R}*\end{matrix}\right\}$	nny	入 力	補間値を計算する y 座標.
17	nny	I	1	入 力	補間値を計算する y 方向の点の個数.
18	zz	$\left\{\begin{matrix}\mathrm{D}* \\ \mathrm{R}*\end{matrix}\right\}$	nnz	入 力	補間値を計算する z 座標.
19	nnz	I	1	入 力	補間値を計算する z 方向の点の個数.
20	s	$\left\{\begin{matrix}\mathrm{D}* \\ \mathrm{R}*\end{matrix}\right\}$	内容参照	出 力	$(x, y, z) = (\mathrm{xx[i-1]}, \mathrm{yy[j-1]}, \mathrm{zz[k-1]})$ における近似関数の値 $S(x, y, z)$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{nnx};$ $\mathrm{j} = 1, 2, \cdots, \mathrm{nny}; \mathrm{k} = 1, 2, \cdots, \mathrm{nnz})$. **大きさ:** nnx×nny×nnz
21	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**：$\mathrm{nxd}^2 \times \mathrm{nyd}^2 \times \mathrm{nzd}^2 + \mathrm{nxd} \times \mathrm{nyd} \times \mathrm{nzd} +$ $\mathrm{mx} \times (\mathrm{nxd} + 2) + \mathrm{my} \times (\mathrm{nyd} + 2) + 2 \times \mathrm{mz} + 3$
22	iwk	I*	内容参照	ワーク	作業領域 **大きさ**：nxd × nyd × nzd + nxd + nyd + nzd
23	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $\mathrm{nxd} \geq 1, \mathrm{nyd} \geq 1, \mathrm{nzd} \geq 1$

(b) $\mathrm{nxd} - \mathrm{mx} \geq 1, \mathrm{nyd} - \mathrm{my} \geq 1, \mathrm{nzd} - \mathrm{mz} \geq 1, \mathrm{mx} \geq 1, \mathrm{my} \geq 1, \mathrm{mz} \geq 1$

(c) $\mathrm{nnx} \geq 1, \mathrm{nny} \geq 1, \mathrm{nnz} \geq 1$

(d) $\mathrm{xd}[0] < \mathrm{xd}[1] < \cdots < \mathrm{xd}[\mathrm{nxd} - 1]$

(e) $\mathrm{yd}[0] < \mathrm{yd}[1] < \cdots < \mathrm{yd}[\mathrm{nyd} - 1]$

(f) $\mathrm{zd}[0] < \mathrm{zd}[1] < \cdots < \mathrm{zd}[\mathrm{nzd} - 1]$

(g) $\mathrm{xk}[\mathrm{mx} - 1] \leq \mathrm{xd}[\mathrm{i} - 1] \leq \mathrm{xk}[\mathrm{nxd}] \quad (\mathrm{i} = 1, 2, \cdots, \mathrm{nxd})$

(h) $\mathrm{yk}[\mathrm{my} - 1] \leq \mathrm{yd}[\mathrm{j} - 1] \leq \mathrm{yk}[\mathrm{nyd}] \quad (\mathrm{j} = 1, 2, \cdots, \mathrm{nyd})$

(i) $\mathrm{zk}[\mathrm{mz} - 1] \leq \mathrm{zd}[\mathrm{k} - 1] \leq \mathrm{zk}[\mathrm{nzd}] \quad (\mathrm{k} = 1, 2, \cdots, \mathrm{nzd})$

(j) $\mathrm{xk}[0] \leq \mathrm{xk}[1] \leq \cdots \leq \mathrm{xk}[\mathrm{nxd} + \mathrm{mx} - 1]$

(k) $\mathrm{yk}[0] \leq \mathrm{yk}[1] \leq \cdots \leq \mathrm{yk}[\mathrm{nyd} + \mathrm{my} - 1]$

(l) $\mathrm{zk}[0] \leq \mathrm{zk}[1] \leq \cdots \leq \mathrm{zk}[\mathrm{nzd} + \mathrm{mz} - 1]$

(m) $\mathrm{xk}[\mathrm{i} - 1] < \mathrm{xk}[\mathrm{i} + \mathrm{mx} - 1] \quad (\mathrm{i} = 1, 2, \cdots, \mathrm{nxd})$

(n) $\mathrm{yk}[\mathrm{j} - 1] < \mathrm{yk}[\mathrm{j} + \mathrm{my} - 1] \quad (\mathrm{j} = 1, 2, \cdots, \mathrm{nyd})$

(o) $\mathrm{zk}[\mathrm{k} - 1] < \mathrm{zk}[\mathrm{k} + \mathrm{mz} - 1] \quad (\mathrm{k} = 1, 2, \cdots, \mathrm{nzd})$

(p) 標本値 $\mathrm{xd}[\mathrm{i} - 1] \quad (\mathrm{i} = 1, 2, \cdots, \mathrm{nxd})$ は Schoenberg–Whitney の条件を満たす (6.1.2 アルゴリズム参照).

(q) 標本値 $\mathrm{yd}[\mathrm{j} - 1] \quad (\mathrm{j} = 1, 2, \cdots, \mathrm{nyd})$ は Schoenberg–Whitney の条件を満たす (6.1.2 アルゴリズム参照).

(r) 標本値 $\mathrm{zd}[\mathrm{k} - 1] \quad (\mathrm{k} = 1, 2, \cdots, \mathrm{nzd})$ は Schoenberg–Whitney の条件を満たす (6.1.2 アルゴリズム参照).

(s) $\mathrm{xk}[\mathrm{mx} - 1] \leq \mathrm{xx}[\mathrm{i} - 1] \leq \mathrm{xk}[\mathrm{nxd}] \quad (\mathrm{i} = 1, 2, \cdots, \mathrm{nnx})$

(t) $\mathrm{yk}[\mathrm{my} - 1] \leq \mathrm{yy}[\mathrm{j} - 1] \leq \mathrm{yk}[\mathrm{nyd}] \quad (\mathrm{j} = 1, 2, \cdots, \mathrm{nny})$

(u) $\mathrm{zk}[\mathrm{mz} - 1] \leq \mathrm{zz}[\mathrm{k} - 1] \leq \mathrm{zk}[\mathrm{nzd}] \quad (\mathrm{k} = 1, 2, \cdots, \mathrm{nnz})$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
2100	正規方程式を解く際の LU 分解において, 対角要素が 0 に近いものがあった. 精度の良い結果が得られない場合がある.	処理を続ける.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
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.	
3400	制限条件 (d) を満足しなかった.	
3410	制限条件 (e) を満足しなかった.	
3420	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.	
3510	制限条件 (h) を満足しなかった.	
3520	制限条件 (i) を満足しなかった.	
3600	制限条件 (j) を満足しなかった.	
3610	制限条件 (k) を満足しなかった.	
3620	制限条件 (l) を満足しなかった.	
3700	制限条件 (m) を満足しなかった.	
3710	制限条件 (n) を満足しなかった.	
3720	制限条件 (o) を満足しなかった.	
3800	制限条件 (p) を満足しなかった.	
3810	制限条件 (q) を満足しなかった.	
3820	制限条件 (r) を満足しなかった.	
3900	制限条件 (s) を満足しなかった.	
3910	制限条件 (t) を満足しなかった.	
3920	制限条件 (u) を満足しなかった.	
4000	正規方程式が解けない.	

(6) **注意事項**

(a) 節点の値によっては, 異なった B–スプラインが得られる場合がある.

(b) 内部の節点の数は (nxd − mx, nyd − my, nzd − mz) となる.

(7) **使用例**

(a) 問 題

次のデータ

$$f_{ijk} = 10 + \frac{1}{0.03 + (x_i - 0.5)^2} + \frac{1}{0.04 + (y_j - 0.2)^2} + \frac{1}{0.05 + (z_k - 0.6)^2}$$
$$(x_i = 0.0, 0.1, \cdots, 0.8; y_j = 0.0, 0.1, \cdots, 0.8; z_k = 0.0, 0.1, \cdots, 0.8)$$

を補間して, $x : 0.0, 0.1, \cdots, 0.8; y : 0.0, 0.1, \cdots, 0.8; z : 0.0, 0.1, \cdots, 0.8$ に対する補間値を求める.

(b) 入力データ

標本点 (xd, yd, zd, fd), nxd=9, nyd=9, nzd=9, x 方向の節点 xk, mx=4, y 方向の節点 yk, my=4, z 方向の節点 zk, mz=4, 補間値を計算する x 座標 xx, nnx=9, 補間値を計算する y 座標 yy, nny=9, 補間値を計算する z 座標 zz, nnz=9.

(c) 主プログラム

```
/*      C interface example for ASL_dgisi3 */

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

int main()
{
    double *xd;
    int nxd;
    double *yd;
    int nyd;
    double *zd;
    int nzd;
    double *fd;
    double *xk;
    int mx;
    double *yk;
    int my;
    double *zk;
    int mz;
    double *xx;
    int nnx;
    double *yy;
    int nny;
    double *zz;
    int nnz;
    double *s;
    double *wk;
    int *iwk;
    int ierr;
    double ff,sumt2,sumd2,fit;
    int i,j,k;
    FILE *fp;

    fp = fopen( "dgisi3.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dgisi3 ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &nxd );
    fscanf( fp, "%d", &mx );
    fscanf( fp, "%d", &nnx );
    fscanf( fp, "%d", &nyd );
    fscanf( fp, "%d", &my );
    fscanf( fp, "%d", &nny );
    fscanf( fp, "%d", &nzd );
    fscanf( fp, "%d", &mz );
    fscanf( fp, "%d", &nnz );

    iwk = ( int * )malloc((size_t)( sizeof(int) *
        (nxd*nyd*nzd+nxd+nyd+nzd) ));
    if( iwk == NULL )
```

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

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

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

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

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

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

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

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

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

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

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

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

wk = ( double * )malloc((size_t)( sizeof(double) *
(nxd*nxd*nyd*nyd*nzd*nzd+nxd*nyd*nzd
 +mx*(nxd+2)+my*(nyd+2)+2*mz+3) ));
if( wk == NULL )
{
    printf( "no enough memory for array wk\n" );
    return -1;
}

printf( "\tnxd = %6d mx = %6d nnx = %6d\n",
 nxd, mx, nnx );
printf( "\tnyd = %6d my = %6d nny = %6d\n",
 nyd, my, nny );
printf( "\tnzd = %6d mz = %6d nnz = %6d\n",
 nzd, mz, nnz );
```

```
for( i=0 ; i<nxd ; i++ )
{
    xd[i] = 0.1 * (double)i;
}
for( j=0 ; j<nyd ; j++ )
{
    yd[j] = 0.1 * (double)j;
}
for( k=0 ; k<nzd ; k++ )
{
    zd[k] = 0.1 * (double)k;
}

for( k=0 ; k<nzd ; k++ )
{
for( j=0 ; j<nyd ; j++ )
{
for( i=0 ; i<nxd ; i++ )
{
    fd[i+j*nxd+k*nxd*nyd] = 10.0
    +1.0/(0.03+(xd[i]-0.5)*(xd[i]-0.5))
    +1.0/(0.04+(yd[j]-0.2)*(yd[j]-0.2))
    +1.0/(0.05+(zd[k]-0.6)*(zd[k]-0.6));
}
}
}

for( i=0 ; i<nxd+mx ; i++ )
{
    fscanf( fp, "%lf", &xk[i] );
}
for( j=0 ; j<nyd+my ; j++ )
{
    fscanf( fp, "%lf", &yk[j] );
}
for( k=0 ; k<nzd+mz ; k++ )
{
    fscanf( fp, "%lf", &zk[k] );
}

for( i=0 ; i<nnx ; i++ )
{
    xx[i] = 0.1 * (double)i;
}
for( j=0 ; j<nny ; j++ )
{
    yy[j] = 0.1 * (double)j;
}
for( k=0 ; k<nnz ; k++ )
{
    zz[k] = 0.1 * (double)k;
}

fclose( fp );

ierr = ASL_dgisi3(xd, nxd, yd, nyd, zd, nzd, fd, xk, mx, yk, my, zk,
     mz, xx, nnx, yy, nny, zz, nnz, s, wk, iwk);

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

if( ierr < 3000 )
{
    sumt2 = 0.0;
    sumd2 = 0.0;
    for( i=0 ; i<nnx ; i++ )
    {
    for( j=0 ; j<nny ; j++ )
    {
    for( k=0 ; k<nnz ; k++ )
    {
        ff = 10.0
           +1.0/(0.03+(xx[i]-0.5)*(xx[i]-0.5))
           +1.0/(0.04+(yy[j]-0.2)*(yy[j]-0.2))
           +1.0/(0.05+(zz[k]-0.6)*(zz[k]-0.6));
        sumt2 += ff * ff;
        sumd2 += (ff-s[i+j*nnx+k*nnx*nny])
               * (ff-s[i+j*nnx+k*nnx*nny]);
    }
    }
    }
    sumt2 = sqrt(sumt2);
    sumd2 = sqrt(sumd2);
    fit = 100.0;
    if( ( sumt2 != 0.0 )||( sumd2 != 0.0 ) )
    {
        fit = ( sumt2 / ( sumt2 + sumd2 ) ) * 100.0;
    }
    printf( "\n\tFitting rate = %8.3g\n", fit );
}

free( iwk );
free( xd );
free( yd );
free( zd );
```

```
        free( fd );
        free( xk );
        free( yk );
        free( zk );
        free( xx );
        free( yy );
        free( zz );
        free( s );
        free( wk );

        return 0;
    }
```

(d) 出力結果

```
        *** ASL_dgisi3 ***

         ** Input **

        nxd =      9 mx =      4 nnx =      9
        nyd =      9 my =      4 nny =      9
        nzd =      9 mz =      4 nnz =      9

         ** Output **

        ierr =      0

        Fitting rate =      100
```

6.5.5 ASL_dgiss1, ASL_rgiss1
B–スプラインによる平滑化 (1 次元データ)

(1) **機 能**

標本点 x_i $(i = 1, 2, \cdots, N)$ において与えられたデータ f_i $(i = 1, 2, \cdots, N)$ を平滑化する $(m-1)$ 次のスプライン関数

$$S(x) \quad = \quad \sum_{i=1}^{n+m} c_i N_{mi}(x)$$

を求め, 指定された点における近似関数の値を計算する.

(2) **使用法**

倍精度関数:

ierr = ASL_dgiss1 (xd, nd, fd, xk, m, xx, nn, s, rs, & aic, wk);

単精度関数:

ierr = ASL_rgiss1 (xd, nd, fd, xk, m, xx, nn, s, rs, & aic, 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	xd	{D∗ / R∗}	nd	入 力	標本点 x_i.
2	nd	I	1	入 力	標本点の数 N.
3	fd	{D∗ / R∗}	nd	入 力	標本点 x_i において与えられたデータ f_i.
4	xk	{D∗ / R∗}	nd + m	入 力	節点 (付加節点も含む) ξ_i.
5	m	I	1	入 力	B–スプラインの階数 m.
6	xx	{D∗ / R∗}	nn	入 力	近似関数の値を計算する x 座標.
7	nn	I	1	入 力	近似関数の値を計算する点の個数.
8	s	{D∗ / R∗}	nn	出 力	$x =$ xx[i − 1] における近似関数の値 $S(x)$ (i = 1, 2, ⋯, nn).
9	rs	{D∗ / R∗}	nd	出 力	残差 $S(x_i) - f_i$.
10	aic	{D∗ / R∗}	1	出 力	赤池の情報量規準 AIC.
11	wk	{D∗ / R∗}	内容参照	ワーク	作業領域. **大きさ** : $\mathrm{nd}^2 + \mathrm{nd} + 2 \times \mathrm{m} + 1$
12	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $\mathrm{nd} \geq 1$

(b) $\mathrm{nd} - \mathrm{m} \geq 1, \mathrm{m} \geq 1$

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

(d) $\mathrm{xd}[0] < \mathrm{xd}[1] < \cdots < \mathrm{xd}[\mathrm{nd} - 1]$

(e) $\mathrm{xk}[\mathrm{m} - 1] \leq \mathrm{xd}[\mathrm{i} - 1] \leq \mathrm{xk}[\mathrm{nd}] \quad (\mathrm{i} = 1, 2, \cdots, \mathrm{nd})$

(f) $\mathrm{xk}[0] \leq \mathrm{xk}[1] \leq \cdots \leq \mathrm{xk}[\mathrm{nd} + \mathrm{m} - 1]$

(g) $\mathrm{xk}[\mathrm{i} - 1] < \mathrm{xk}[\mathrm{i} + \mathrm{m} - 1] \quad (\mathrm{i} = 1, 2, \cdots, \mathrm{nd})$

(h) 標本値 $\mathrm{xd}[\mathrm{i} - 1] \quad (\mathrm{i} = 1, 2, \cdots, \mathrm{nd})$ は Schoenberg–Whitney の条件を満たす (6.1.2 アルゴリズム参照).

(i) $\mathrm{xk}[\mathrm{m} - 1] \leq \mathrm{xx}[\mathrm{i} - 1] \leq \mathrm{xk}[\mathrm{nd}] \quad (\mathrm{i} = 1, 2, \cdots, \mathrm{nn})$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
2100	正規方程式を解く際の LU 分解において, 対角要素が 0 に近いものがあった. 精度の良い結果が得られない場合がある.	処理を続ける.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
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.	
3400	制限条件 (d) を満足しなかった.	
3500	制限条件 (e) を満足しなかった.	
3600	制限条件 (f) を満足しなかった.	
3700	制限条件 (g) を満足しなかった.	
3800	制限条件 (h) を満足しなかった.	
3900	制限条件 (i) を満足しなかった.	
4000	正規方程式が解けない.	

(6) **注意事項**

(a) 節点の値によっては, 異なった B–スプラインが得られる場合がある.

(b) 内部の節点の数は nd − m となる.

(7) **使用例**

(a) 問 題

次のデータ

$$f_i = \frac{1}{0.01 + (x_i - 0.3)^2} + e_i \quad (x_i = 0.0, 0.1, \cdots, 1.0)$$

に対して 3 次のスプライン関数を用いてあてはめを行う. ここで e_i は互いに独立で, 平均値 0, 分散 1 の正規分布をする誤差である.

(b) 入力データ

標本点 (xd, fd), nd=11, 節点 xk, m=4, 補間値を計算する座標 xx, nn=19.

(c) 主プログラム

```
/*      C interface example for ASL_dgiss1 */

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

int main()
{
    double *xd;
    int nd;
    double *fd;
    double *xk;
    int m;
    double *xx;
    int nn;
    double *s;
    double *rs;
    double aic;
    double *wk;
    double am,sg;
    int ierr;
    int i,ix,iy;
    double *e;
    FILE *fp;

    fp = fopen( "dgiss1.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dgiss1 ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &nd );
    fscanf( fp, "%d", &m );
    fscanf( fp, "%d", &nn );

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

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

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

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

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

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

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

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

    printf( "\tnd = %6d m = %6d nn = %6d\n", nd,m,nn );

    printf( "\n\n\txd\n\n" );
    for( i=0 ; i<nd ; i++ )
    {
        xd[i] = 0.1 * (double)i;
        printf( "\txd[%6d] = %8.3g\n", i, xd[i] );
    }

    ix=1;
    iy=1;
    am = 0.0;
    sg = 1.0;

    ierr = ASL_djdbno(nd, am, sg, &ix, &iy, e);

    for( i=0 ; i<nd ; i++ )
    {
        fd[i] = 1.0/(0.01+(xd[i]-0.3)*(xd[i]-0.3))+e[i];
    }

    printf( "\n\n\txk\n\n" );
    for( i=0 ; i<nd+m ; i++ )
    {
        fscanf( fp, "%lf", &xk[i] );
        printf( "\txk[%6d] = %8.3g\n", i, xk[i] );
    }

    for( i=0 ; i<nn ; i++ )
    {
        xx[i] = 0.05 * (i+1);
    }

    fclose( fp );

    ierr = ASL_dgiss1(xd, nd, fd, xk, m, xx, nn, s, rs, &aic, wk);

    printf( "\n     ** Output **\n\n" );
    printf( "\n\n\tierr = %6d\n", ierr );
    printf( "\n\n\taic  = %8.3g\n", aic );
    printf( "\n\n\tValues of approximating spline function on sample points\n\n");
    printf( "\t       xx  \t       s\n\n" );
    for( i=0 ; i<nn ; i++ )
    {
        printf( "\t%8.3g\t%8.3g\n", xx[i], s[i] );
    }

    free( xd );
    free( fd );
    free( xk );
    free( xx );
    free( s );
    free( rs );
    free( e );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dgiss1 ***

 ** Input **

nd =     11 m =      4 nn =     19
```

```
xd

xd[     0] =        0
xd[     1] =      0.1
xd[     2] =      0.2
xd[     3] =      0.3
xd[     4] =      0.4
xd[     5] =      0.5
xd[     6] =      0.6
xd[     7] =      0.7
xd[     8] =      0.8
xd[     9] =      0.9
xd[    10] =        1


xk

xk[     0] =        0
xk[     1] =        0
xk[     2] =        0
xk[     3] =        0
xk[     4] =      0.2
xk[     5] =      0.3
xk[     6] =      0.4
xk[     7] =      0.5
xk[     8] =      0.6
xk[     9] =      0.7
xk[    10] =      0.8
xk[    11] =        1
xk[    12] =        1
xk[    13] =        1
xk[    14] =        1

 ** Output **


ierr =      0


aic  =     -677


Values of approximating spline function on sample points

       xx          s

    0.05        14.9
     0.1        20.3
    0.15        30.4
     0.2        51.4
    0.25          83
     0.3        99.9
    0.35        81.1
     0.4        48.9
    0.45        29.3
     0.5        20.2
    0.55        14.4
     0.6        10.4
    0.65        8.09
     0.7        6.26
    0.75        3.91
     0.8        1.97
    0.85        1.51
     0.9        1.97
    0.95        2.42
```

6.5.6 ASL_dgiss2, ASL_rgiss2
B–スプラインによる平滑化 (2 次元データ)

(1) **機 能**

標本点 (x_i, y_j) $(i = 1, 2, \cdots, N_x; j = 1, 2, \cdots, N_y)$ において与えられたデータ f_{ij} $(i = 1, 2, \cdots, N_x; j = 1, 2, \cdots, N_y)$ を平滑化するスプライン関数

$$S(x, y) \quad = \quad \sum_{i=1}^{h+m} \sum_{j=1}^{k+n} c_{ij} N_{mi}(x) N_{nj}(y)$$

を求め, 指定された格子点における近似関数の値を計算する.

(2) **使用法**

倍精度関数:

ierr = ASL_dgiss2 (xd, nxd, yd, nyd, fd, xk, mx, yk, my, xx, nnx, yy, nny, s, rs, & aic, wk, iwk);

単精度関数:

ierr = ASL_rgiss2 (xd, nxd, yd, nyd, fd, xk, mx, yk, my, xx, nnx, yy, nny, s, rs, & aic, wk, iwk);

(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	xd	{D∗ R∗}	nxd	入 力	x 方向の標本点 x_i.
2	nxd	I	1	入 力	x 方向の標本点の数 N_x.
3	yd	{D∗ R∗}	nyd	入 力	y 方向の標本点 y_j.
4	nyd	I	1	入 力	y 方向の標本点の数 N_y.
5	fd	{D∗ R∗}	nxd×nyd	入 力	標本点 (x_i, y_j) において与えられたデータ f_{ij}.
6	xk	{D∗ R∗}	nxd + mx	入 力	x 方向の節点 (付加節点も含む) ξ_i.
7	mx	I	1	入 力	x 方向の B–スプラインの階数 m.
8	yk	{D∗ R∗}	nyd + my	入 力	y 方向の節点 (付加節点も含む) ζ_j.
9	my	I	1	入 力	y 方向の B–スプラインの階数 n.
10	xx	{D∗ R∗}	nnx	入 力	近似関数の値を計算する x 座標.
11	nnx	I	1	入 力	近似関数の値を計算する x 方向の点の個数.
12	yy	{D∗ R∗}	nny	入 力	近似関数の値を計算する y 座標.
13	nny	I	1	入 力	近似関数の値を計算する y 方向の点の個数.
14	s	{D∗ R∗}	nnx × nny	出 力	$(x, y) = (\mathrm{xx[i-1]}, \mathrm{yy[j-1]})$ における近似関数の値 $S(x, y)$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{nnx}\ ;\ \mathrm{j} = 1, 2, \cdots, \mathrm{nny})$.
15	rs	{D∗ R∗}	nxd×nyd	出 力	残差 $S(x_i, y_j) - f_{ij}$.
16	aic	{D∗ R∗}	1	出 力	赤池の情報量規準 AIC.
17	wk	{D∗ R∗}	内容参照	ワーク	作業領域. **大きさ**：$\mathrm{nxd}^2 \times (\mathrm{nyd}^2 + 1) + \mathrm{nxd} \times (\mathrm{nyd} + \mathrm{mx}) + 2 \times \mathrm{mx} + 2 \times \mathrm{my} + 2$
18	iwk	I*	nxd + nyd	ワーク	作業領域
19	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $\mathrm{nxd} \geq 1, \mathrm{nyd} \geq 1$

(b) $\mathrm{nxd} - \mathrm{mx} \geq 1, \mathrm{nyd} - \mathrm{my} \geq 1, \mathrm{mx} \geq 1, \mathrm{my} \geq 1$

(c) $\mathrm{nnx} \geq 1, \mathrm{nny} \geq 1$

(d) $\mathrm{xd}[0] < \mathrm{xd}[1] < \cdots < \mathrm{xd}[\mathrm{nxd} - 1]$

(e) $\mathrm{yd}[0] < \mathrm{yd}[1] < \cdots < \mathrm{yd}[\mathrm{nyd} - 1]$

(f) $\mathrm{xk}[\mathrm{mx} - 1] \leq \mathrm{xd}[\mathrm{i} - 1] \leq \mathrm{xk}[\mathrm{nxd}] \quad (\mathrm{i} = 1, 2, \cdots, \mathrm{nxd})$

(g) $\mathrm{yk}[\mathrm{my} - 1] \leq \mathrm{yd}[\mathrm{j} - 1] \leq \mathrm{yk}[\mathrm{nyd}] \quad (\mathrm{j} = 1, 2, \cdots, \mathrm{nyd})$

(h) $\mathrm{xk}[0] \leq \mathrm{xk}[1] \leq \cdots \leq \mathrm{xk}[\mathrm{nxd} + \mathrm{mx} - 1]$

(i) $\mathrm{yk}[0] \leq \mathrm{yk}[1] \leq \cdots \leq \mathrm{yk}[\mathrm{nyd} + \mathrm{my} - 1]$

(j) $\mathrm{xk}[\mathrm{i} - 1] < \mathrm{xk}[\mathrm{i} + \mathrm{mx} - 1] \quad (\mathrm{i} = 1, 2, \cdots, \mathrm{nxd})$

(k) $\mathrm{yk}[\mathrm{j} - 1] < \mathrm{yk}[\mathrm{j} + \mathrm{my} - 1] \quad (\mathrm{j} = 1, 2, \cdots, \mathrm{nyd})$

(l) 標本値 $\mathrm{xd}[\mathrm{i} - 1] \quad (\mathrm{i} = 1, 2, \cdots, \mathrm{nxd})$ は Schoenberg–Whitney の条件を満たす (6.1.2 アルゴリズム参照).

(m) 標本値 $\mathrm{yd}[\mathrm{j} - 1] \quad (\mathrm{j} = 1, 2, \cdots, \mathrm{nyd})$ は Schoenberg–Whitney の条件を満たす (6.1.2 アルゴリズム参照).

(n) $\mathrm{xk}[\mathrm{mx} - 1] \leq \mathrm{xx}[\mathrm{i} - 1] \leq \mathrm{xk}[\mathrm{nxd}] \quad (\mathrm{i} = 1, 2, \cdots, \mathrm{nnx})$

(o) $\mathrm{yk}[\mathrm{my} - 1] \leq \mathrm{yy}[\mathrm{j} - 1] \leq \mathrm{yk}[\mathrm{nyd}] \quad (\mathrm{j} = 1, 2, \cdots, \mathrm{nny})$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
2100	正規方程式を解く際の LU 分解において, 対角要素が 0 に近いものがあった. 精度の良い結果が得られない場合がある.	処理を続ける.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
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.	
3400	制限条件 (d) を満足しなかった.	
3410	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.	
3510	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.	
3610	制限条件 (i) を満足しなかった.	
3700	制限条件 (j) を満足しなかった.	
3710	制限条件 (k) を満足しなかった.	
3800	制限条件 (l) を満足しなかった.	
3810	制限条件 (m) を満足しなかった.	
3900	制限条件 (n) を満足しなかった.	
3910	制限条件 (o) を満足しなかった.	
4000	正規方程式が解けない.	

(6) **注意事項**

(a) 節点の値によっては, 異なった B–スプラインが得られる場合がある.

(b) 内部の節点の数は $(\mathrm{nxd}-\mathrm{mx}, \mathrm{nyd}-\mathrm{my})$ となる.

(7) 使用例

(a) 問 題

次のデータ

$$f_{ij} = \frac{1}{0.01 + (x_i - 0.3)^2} + \frac{1}{0.02 + (y_j - 0.4)^2} + e_{ij}$$
$$(x_i = 0.0, 0.1, \cdots, 1.0; y_j = 0.0, 0.1, \cdots, 1.0)$$

に対して 3 次のスプライン関数を用いてあてはめを行う. ここで e_{ij} は互いに独立で, 平均値 0, 分散 1 の正規分布をする誤差である.

(b) 入力データ

標本点 (xd, yd, fd), nxd=11, nyd=11, 節点 xk, mx=4, 節点 yk, my=4, 補間値を計算する座標 xx, nnx=11, 補間値を計算する座標 yy, nny=11.

(c) 主プログラム

```
/*      C interface example for ASL_dgiss2 */

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

int main()
{
    double *xd;
    int nxd;
    double *yd;
    int nyd;
    double *fd;
    double *xk;
    int mx;
    double *yk;
    int my;
    double *xx;
    int nnx;
    double *yy;
    int nny;
    double *s;
    double *rs;
    double aic;
    double *wk;
    int *iwk;
    int ierr;
    int i,j,ix,iy;
    double *e,am,sg;
    FILE *fp;

    fp = fopen( "dgiss2.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dgiss2 ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &nxd );
    fscanf( fp, "%d", &mx );
    fscanf( fp, "%d", &nnx );
    fscanf( fp, "%d", &nyd );
    fscanf( fp, "%d", &my );
    fscanf( fp, "%d", &nny );


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

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

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

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

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

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

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

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

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

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

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

wk = ( double * )malloc((size_t)( sizeof(double)*
(nxd*nxd*(nyd*nyd+1)+nxd*(nyd+mx)+mx*mx+my*my+2) ));
if( wk == NULL )
{
    printf( "no enough memory for array wk\n" );
    return -1;
}

printf( "\tnxd = %6d mx = %6d nnx = %6d\n",
    nxd, mx, nnx );
printf( "\tnyd = %6d my = %6d nny = %6d\n",
    nyd, my, nny );

printf( "\n\n\txd\n\n" );
for( i=0 ; i<nxd ; i++ )
{
    xd[i] = 0.1 * (double)i;
    printf( "\txd[%6d] = %8.3g\n", i, xd[i] );
}

printf( "\n\n\tyd\n\n" );
for( j=0 ; j<nyd ; j++ )
{
    yd[j] = 0.1 * (double)j;
    printf( "\tyd[%6d] = %8.3g\n", j, yd[j] );
}

ix=1;
iy=1;
am = 0.0;
sg = 1.0;

ierr = ASL_djdbno(nxd*nyd, am, sg, &ix, &iy, e);

for( j=0 ; j<nyd ; j++ )
{
    for( i=0 ; i<nxd ; i++ )
    {
        fd[i+nxd*j]=
         1.0/(0.01+(xd[i]-0.3)*(xd[i]-0.3))
```

```
            +1.0/(0.02+(yd[j]-0.4)*(yd[j]-0.4))
            +e[j*nxd+i];
        }
    }

    printf( "\n\txk\n\n" );
    for( i=0 ; i<nxd+mx ; i++ )
    {
        fscanf( fp, "%lf", &xk[i] );
        printf( "\txk[%6d] = %8.3g\n", i, xk[i] );
    }

    printf( "\n\tyk\n\n" );
    for( j=0 ; j<nyd+my ; j++ )
    {
        fscanf( fp, "%lf", &yk[j] );
        printf( "\tyk[%6d] = %8.3g\n", j, yk[j] );
    }

    for( i=0 ; i<nnx ; i++ )
    {
        xx[i] = 0.1 * (double)i;
    }

    for( j=0 ; j<nny ; j++ )
    {
        yy[j] = 0.1 * (double)j;
    }

    fclose( fp );

    ierr = ASL_dgiss2(xd, nxd, yd, nyd, fd, xk, mx, yk, my, xx, nnx,
         yy, nny, s, rs, &aic, wk, iwk);

    printf( "\n     ** Output **\n\n" );
    printf( "\n\n\tierr = %6d\n", ierr );
    printf( "\n\n\taic  = %8.3g\n", aic );
    printf( "\n\n\tValues of an approximating spline function on sample points\n\n" );
    printf( "\n\n\t      xx   \t      yy   \t     s\n\n" );
    for( i=0 ; i<nnx ; i++ )
    {
        for( j=0 ; j<nny ; j++ )
        {
            printf( "\t%8.3g\t%8.3g\t%8.3g\n", xx[i], yy[j], s[i+nnx*j] );
        }
    }

    free( iwk );
    free( xd );
    free( yd );
    free( fd );
    free( xk );
    free( yk );
    free( xx );
    free( yy );
    free( s );
    free( rs );
    free( e );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dgiss2 ***

 ** Input **

nxd =     11 mx =      4 nnx =     11
nyd =     11 my =      4 nny =     11


xd

xd[     0] =        0
xd[     1] =      0.1
xd[     2] =      0.2
xd[     3] =      0.3
xd[     4] =      0.4
xd[     5] =      0.5
xd[     6] =      0.6
xd[     7] =      0.7
xd[     8] =      0.8
xd[     9] =      0.9
xd[    10] =        1


yd

yd[     0] =        0
yd[     1] =      0.1
yd[     2] =      0.2
yd[     3] =      0.3
yd[     4] =      0.4
```

```
yd[     5] =      0.5
yd[     6] =      0.6
yd[     7] =      0.7
yd[     8] =      0.8
yd[     9] =      0.9
yd[    10] =        1

xk

xk[     0] =        0
xk[     1] =        0
xk[     2] =        0
xk[     3] =        0
xk[     4] =      0.2
xk[     5] =      0.3
xk[     6] =      0.4
xk[     7] =      0.5
xk[     8] =      0.6
xk[     9] =      0.7
xk[    10] =      0.8
xk[    11] =        1
xk[    12] =        1
xk[    13] =        1
xk[    14] =        1

yk

yk[     0] =        0
yk[     1] =        0
yk[     2] =        0
yk[     3] =        0
yk[     4] =      0.2
yk[     5] =      0.3
yk[     6] =      0.4
yk[     7] =      0.5
yk[     8] =      0.6
yk[     9] =      0.7
yk[    10] =      0.8
yk[    11] =        1
yk[    12] =        1
yk[    13] =        1
yk[    14] =        1

 ** Output **


ierr =      0

aic  = -6.7e+03

Values of an approximating spline function on sample points


      xx          yy         s

       0           0        13.6
       0         0.1        17.9
       0         0.2        24.9
       0         0.3        44.2
       0         0.4        60.6
       0         0.5        43.4
       0         0.6        26.1
       0         0.7        18.4
       0         0.8        15.7
       0         0.9        15.4
       0           1          13
     0.1           0        25.8
     0.1         0.1        29.5
     0.1         0.2          34
     0.1         0.3        52.7
     0.1         0.4          72
     0.1         0.5        53.3
     0.1         0.6        35.1
     0.1         0.7        28.8
     0.1         0.8        25.9
     0.1         0.9        24.2
     0.1           1        21.5
     0.2           0        56.9
     0.2         0.1        60.6
     0.2         0.2        66.2
     0.2         0.3          84
     0.2         0.4         100
     0.2         0.5        84.9
     0.2         0.6        64.9
     0.2         0.7        59.5
     0.2         0.8        55.1
     0.2         0.9        53.5
     0.2           1        51.6
     0.3           0         105
     0.3         0.1         110
     0.3         0.2         117
     0.3         0.3         134
     0.3         0.4         150
```

```
0.3         0.5          134
0.3         0.6          115
0.3         0.7          107
0.3         0.8          108
0.3         0.9          103
0.3           1          104
0.4           0         54.5
0.4         0.1           60
0.4         0.2         67.3
0.4         0.3         83.5
0.4         0.4         99.8
0.4         0.5         83.2
0.4         0.6         64.6
0.4         0.7         58.4
0.4         0.8         56.3
0.4         0.9         53.9
0.4           1           54
0.5           0         25.8
0.5         0.1         28.1
0.5         0.2         39.1
0.5         0.3         53.3
0.5         0.4         71.3
0.5         0.5         52.5
0.5         0.6         35.8
0.5         0.7           28
0.5         0.8         25.3
0.5         0.9         23.1
0.5           1         21.7
0.6           0         15.9
0.6         0.1         20.6
0.6         0.2         26.5
0.6         0.3         42.6
0.6         0.4         59.9
0.6         0.5         43.2
0.6         0.6         26.3
0.6         0.7         17.6
0.6         0.8         16.7
0.6         0.9         12.7
0.6           1         11.8
0.7           0         11.8
0.7         0.1         14.4
0.7         0.2         21.3
0.7         0.3         39.9
0.7         0.4         55.4
0.7         0.5         38.7
0.7         0.6         21.9
0.7         0.7         15.2
0.7         0.8         11.7
0.7         0.9         8.95
0.7           1         9.04
0.8           0         7.53
0.8         0.1         13.6
0.8         0.2         21.6
0.8         0.3         35.7
0.8         0.4         53.6
0.8         0.5         36.9
0.8         0.6         21.5
0.8         0.7         11.6
0.8         0.8         8.22
0.8         0.9         8.12
0.8           1          6.2
0.9           0         7.53
0.9         0.1           13
0.9         0.2         17.9
0.9         0.3         35.8
0.9         0.4         52.5
0.9         0.5         35.1
0.9         0.6         19.2
0.9         0.7         11.2
0.9         0.8          8.4
0.9         0.9         4.64
0.9           1         4.93
  1           0         7.45
  1         0.1         9.82
  1         0.2         18.4
  1         0.3         36.2
  1         0.4         52.4
  1         0.5         35.8
  1         0.6         18.2
  1         0.7         12.2
  1         0.8         7.59
  1         0.9         4.93
  1           1         5.85
```

6.5.7 ASL_dgiss3, ASL_rgiss3
B–スプラインによる平滑化 (3 次元データ)

(1) **機 能**

標本点 (x_i, y_j, z_k) $(i = 1, 2, \cdots, N_x; j = 1, 2, \cdots, N_y; k = 1, 2, \cdots, N_z)$ において与えられたデータ f_{ijk} $(i = 1, 2, \cdots, N_x; j = 1, 2, \cdots, N_y; k = 1, 2, \cdots, N_z)$ を平滑化するスプライン関数

$$S(x, y, z) \quad = \quad \sum_{i=1}^{n_x+m_x} \sum_{j=1}^{n_y+m_y} \sum_{k=1}^{n_z+m_z} c_{ijk} N_{m_x i}(x) N_{m_y j}(y) N_{m_z k}(z)$$

を求め, 指定された格子点における近似関数の値を計算する.

(2) **使用法**

倍精度関数:

ierr = ASL_dgiss3 (xd, nxd, yd, nyd, zd, nzd, fd, xk, mx, yk, my, zk, mz, xx, nnx, yy, nny, zz, nnz, s, rs, & aic, wk, iwk);

単精度関数:

ierr = ASL_rgiss3 (xd, nxd, yd, nyd, zd, nzd, fd, xk, mx, yk, my, zk, mz, xx, nnx, yy, nny, zz, nnz, s, rs, & aic, 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	xd	{D∗ / R∗}	nxd	入 力	x 方向の標本点 x_i.
2	nxd	I	1	入 力	x 方向の標本点の数 N_x.
3	yd	{D∗ / R∗}	nyd	入 力	y 方向の標本点 y_j.
4	nyd	I	1	入 力	y 方向の標本点の数 N_y.
5	zd	{D∗ / R∗}	nzd	入 力	z 方向の標本点 z_k.
6	nzd	I	1	入 力	z 方向の標本点の数 N_z.
7	fd	{D∗ / R∗}	内容参照	入 力	標本点 (x_i, y_j, z_k) において与えられたデータ f_{ijk}. **大きさ:** nxd×nyd×nzd
8	xk	{D∗ / R∗}	nxd + mx	入 力	x 方向の節点 (付加節点も含む) ξ_i.
9	mx	I	1	入 力	x 方向の B–スプラインの階数 m_x.
10	yk	{D∗ / R∗}	nyd + my	入 力	y 方向の節点 (付加節点も含む) ζ_j.
11	my	I	1	入 力	y 方向の B–スプラインの階数 m_y.
12	zk	{D∗ / R∗}	nzd + mz	入 力	z 方向の節点 (付加節点も含む) η_k.

項番	引数と戻り値	型	大きさ	入出力	内　　容
13	mz	I	1	入 力	z 方向の B–スプラインの階数 m_z.
14	xx	$\left\{\begin{matrix}\mathrm{D}* \\ \mathrm{R}*\end{matrix}\right\}$	nnx	入 力	近似関数の値を計算する x 座標.
15	nnx	I	1	入 力	近似関数の値を計算する x 方向の点の個数.
16	yy	$\left\{\begin{matrix}\mathrm{D}* \\ \mathrm{R}*\end{matrix}\right\}$	nny	入 力	近似関数の値を計算する y 座標.
17	nny	I	1	入 力	近似関数の値を計算する y 方向の点の個数.
18	zz	$\left\{\begin{matrix}\mathrm{D}* \\ \mathrm{R}*\end{matrix}\right\}$	nnz	入 力	近似関数の値を計算する z 座標.
19	nnz	I	1	入 力	近似関数の値を計算する z 方向の点の個数.
20	s	$\left\{\begin{matrix}\mathrm{D}* \\ \mathrm{R}*\end{matrix}\right\}$	内容参照	出 力	$(x, y, z) = (\mathrm{xx[i-1]}, \mathrm{yy[j-1]}, \mathrm{zz[k-1]})$ における近似関数の値 $S(x, y, z)$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{nnx};$ $\mathrm{j} = 1, 2, \cdots, \mathrm{nny}; \mathrm{k} = 1, 2, \cdots, \mathrm{nnz})$. **大きさ:** nnx×nny×nnz
21	rs	$\left\{\begin{matrix}\mathrm{D}* \\ \mathrm{R}*\end{matrix}\right\}$	内容参照	出 力	残差 $S(x_i, y_j, z_k) - f_{ijk}$. **大きさ:** nxd×nyd×nzd
22	aic	$\left\{\begin{matrix}\mathrm{D}* \\ \mathrm{R}*\end{matrix}\right\}$	1	出 力	赤池の情報量規準 AIC.
23	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**：$\mathrm{nxd}^2 \times \mathrm{nyd}^2 \times \mathrm{nzd}^2 + \mathrm{nxd} \times \mathrm{nyd} \times \mathrm{nzd} + \mathrm{nxd}^2 \times \mathrm{nyd}^2 + \mathrm{nxd}^2 + \mathrm{mx} \times (\mathrm{nxd}+2) + \mathrm{my} \times (\mathrm{nyd}+2) + 2 \times \mathrm{mz} + 3$
24	iwk	I*	内容参照	ワーク	作業領域 **大きさ**：$\mathrm{nxd} \times \mathrm{nyd} \times \mathrm{nzd} + \mathrm{nxd} + \mathrm{nyd} + \mathrm{nzd}$
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) $\mathrm{nxd} \geq 1, \mathrm{nyd} \geq 1, \mathrm{nzd} \geq 1$

(b) $\mathrm{nxd} - \mathrm{mx} \geq 1, \mathrm{nyd} - \mathrm{my} \geq 1, \mathrm{nzd} - \mathrm{mz} \geq 1, \mathrm{mx} \geq 1, \mathrm{my} \geq 1, \mathrm{mz} \geq 1$

(c) $\mathrm{nnx} \geq 1, \mathrm{nny} \geq 1, \mathrm{nnz} \geq 1$

(d) $\mathrm{xd}[0] < \mathrm{xd}[1] < \cdots < \mathrm{xd}[\mathrm{nxd} - 1]$

(e) $\mathrm{yd}[0] < \mathrm{yd}[1] < \cdots < \mathrm{yd}[\mathrm{nyd} - 1]$

(f) $\mathrm{zd}[0] < \mathrm{zd}[1] < \cdots < \mathrm{zd}[\mathrm{nzd} - 1]$

(g) $\mathrm{xk}[\mathrm{mx} - 1] \leq \mathrm{xd}[\mathrm{i} - 1] \leq \mathrm{xk}[\mathrm{nxd}] \quad (\mathrm{i} = 1, 2, \cdots, \mathrm{nxd})$

(h) $\mathrm{yk}[\mathrm{my} - 1] \leq \mathrm{yd}[\mathrm{j} - 1] \leq \mathrm{yk}[\mathrm{nyd}] \quad (\mathrm{j} = 1, 2, \cdots, \mathrm{nyd})$

(i) $\mathrm{zk}[\mathrm{mz} - 1] \leq \mathrm{zd}[\mathrm{k} - 1] \leq \mathrm{zk}[\mathrm{nzd}] \quad (\mathrm{k} = 1, 2, \cdots, \mathrm{nzd})$

(j) $\mathrm{xk}[0] \leq \mathrm{xk}[1] \leq \cdots \leq \mathrm{xk}[\mathrm{nxd} + \mathrm{mx} - 1]$

(k) $\mathrm{yk}[0] \leq \mathrm{yk}[1] \leq \cdots \leq \mathrm{yk}[\mathrm{nyd} + \mathrm{my} - 1]$

(l) $\mathrm{zk}[0] \leq \mathrm{zk}[1] \leq \cdots \leq \mathrm{zk}[\mathrm{nzd} + \mathrm{mz} - 1]$

(m) $\mathrm{xk}[\mathrm{i} - 1] < \mathrm{xk}[\mathrm{i} + \mathrm{mx} - 1] \quad (\mathrm{i} = 1, 2, \cdots, \mathrm{nxd})$

(n) $\mathrm{yk}[\mathrm{j} - 1] < \mathrm{yk}[\mathrm{j} + \mathrm{my} - 1] \quad (\mathrm{j} = 1, 2, \cdots, \mathrm{nyd})$

(o) $\mathrm{zk}[\mathrm{k} - 1] < \mathrm{zk}[\mathrm{k} + \mathrm{mz} - 1] \quad (\mathrm{k} = 1, 2, \cdots, \mathrm{nzd})$

(p) 標本値 $\mathrm{xd}[\mathrm{i} - 1] \quad (\mathrm{i} = 1, 2, \cdots, \mathrm{nxd})$ は Schoenberg–Whitney の条件を満たす (6.1.2 アルゴリズム参照).

(q) 標本値 $\mathrm{yd}[\mathrm{j} - 1] \quad (\mathrm{j} = 1, 2, \cdots, \mathrm{nyd})$ は Schoenberg–Whitney の条件を満たす (6.1.2 アルゴリズム参照).

(r) 標本値 $\mathrm{zd}[\mathrm{k} - 1] \quad (\mathrm{k} = 1, 2, \cdots, \mathrm{nzd})$ は Schoenberg–Whitney の条件を満たす (6.1.2 アルゴリズム参照).

(s) $\mathrm{xk}[\mathrm{mx} - 1] \leq \mathrm{xx}[\mathrm{i} - 1] \leq \mathrm{xk}[\mathrm{nxd}] \quad (\mathrm{i} = 1, 2, \cdots, \mathrm{nnx})$

(t) $\mathrm{yk}[\mathrm{my} - 1] \leq \mathrm{yy}[\mathrm{j} - 1] \leq \mathrm{yk}[\mathrm{nyd}] \quad (\mathrm{j} = 1, 2, \cdots, \mathrm{nny})$

(u) $\mathrm{zk}[\mathrm{mz} - 1] \leq \mathrm{zz}[\mathrm{k} - 1] \leq \mathrm{zk}[\mathrm{nzd}] \quad (\mathrm{k} = 1, 2, \cdots, \mathrm{nnz})$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
2100	正規方程式を解く際の LU 分解において, 対角要素が 0 に近いものがあった. 精度の良い結果が得られない場合がある.	処理を続ける.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
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.	
3400	制限条件 (d) を満足しなかった.	
3410	制限条件 (e) を満足しなかった.	
3420	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.	
3510	制限条件 (h) を満足しなかった.	
3520	制限条件 (i) を満足しなかった.	
3600	制限条件 (j) を満足しなかった.	
3610	制限条件 (k) を満足しなかった.	
3620	制限条件 (l) を満足しなかった.	
3700	制限条件 (m) を満足しなかった.	
3710	制限条件 (n) を満足しなかった.	
3720	制限条件 (o) を満足しなかった.	
3800	制限条件 (p) を満足しなかった.	
3810	制限条件 (q) を満足しなかった.	
3820	制限条件 (r) を満足しなかった.	
3900	制限条件 (s) を満足しなかった.	
3910	制限条件 (t) を満足しなかった.	
3920	制限条件 (u) を満足しなかった.	
4000	正規方程式が解けない.	

(6) **注意事項**

(a) 節点の値によっては, 異なった B–スプラインが得られる場合がある.

(b) 内部の節点の数は $(\mathrm{nxd}-\mathrm{mx}, \mathrm{nyd}-\mathrm{my}, \mathrm{nzd}-\mathrm{mz})$ となる.

(c) 本ライブラリでは倍精度関数と単精度関数が用意されているが, 解の精度の観点から倍精度関数を使用する事が望ましい.

(7) 使用例

(a) 問 題

次のデータ

$$f_{ijk} = 10 + \frac{1}{0.03 + (x_i - 0.5)^2} + \frac{1}{0.04 + (y_j - 0.2)^2} + \frac{1}{0.05 + (z_k - 0.6)^2} + e_{ij}$$
$$(x_i = 0.0, 0.1, \cdots, 0.8; y_j = 0.0, 0.1, \cdots, 0.8; z_k = 0.0, 0.1, \cdots, 0.8)$$

に対して 3 次のスプライン関数を用いてあてはめを行う. ここで e_{ijk} は互いに独立で, 平均値 0, 分散 0.01 の正規分布をする誤差である.

(b) 入力データ

標本点 (xd, yd, zd, fd), nxd=9, nyd=9, nzd=9, x 方向の節点 xk, mx=4, y 方向の節点 yk, my=4, z 方向の節点 zk, mz=4, 補間値を計算する x 座標 xx, nnx=9, 補間値を計算する y 座標 yy, nny=9, 補間値を計算する z 座標 zz, nnz=9.

(c) 主プログラム

```
/*      C interface example for ASL_dgiss3 */

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

int main()
{
    double *xd;
    int nxd;
    double *yd;
    int nyd;
    double *zd;
    int nzd;
    double *fd;
    double *xk;
    int mx;
    double *yk;
    int my;
    double *zk;
    int mz;
    double *xx;
    int nnx;
    double *yy;
    int nny;
    double *zz;
    int nnz;
    double *s;
    double *rs;
    double aic;
    double *wk;
    double *e,am,sg;
    int *iwk;
    int ierr;
    double ff,sumt2,sumd2,fit;
    int i,j,k,ix,iy;
    FILE *fp;

    fp = fopen( "dgiss3.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dgiss3 ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &nxd );
    fscanf( fp, "%d", &mx );
    fscanf( fp, "%d", &nnx );
    fscanf( fp, "%d", &nyd );
    fscanf( fp, "%d", &my );
    fscanf( fp, "%d", &nny );
    fscanf( fp, "%d", &nzd );
    fscanf( fp, "%d", &mz );
    fscanf( fp, "%d", &nnz );

    iwk = ( int * )malloc((size_t)( sizeof(int) *
        (nxd+nyd+nzd+nxd*nyd*nzd) ));
    if( iwk == NULL )
    {
        printf( "no enough memory for array iwk\n" );
        return -1;
    }
```

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

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

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

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

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

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

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

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

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

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

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

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

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

wk = ( double * )malloc((size_t)( sizeof(double)*
(nxd*nxd*nyd*nyd*nzd*nzd+nxd*nyd*nzd
 +nxd*nxd*nyd*nyd+nxd*nxd+mx*(2+nxd)
 +my*(2+nyd)+2*mz+3) ));
```

```
if( wk == NULL )
{
    printf( "no enough memory for array wk\n" );
    return -1;
}

printf( "\tnxd = %6d mx = %6d nnx = %6d\n",
    nxd, mx, nnx );
printf( "\tnyd = %6d my = %6d nny = %6d\n",
    nyd, my, nny );
printf( "\tnzd = %6d mz = %6d nnz = %6d\n",
    nzd, mz, nnz );

for( i=0 ; i<nxd ; i++ )
{
    xd[i] = 0.1 * (double)i;
}
for( j=0 ; j<nyd ; j++ )
{
    yd[j] = 0.1 * (double)j;
}
for( k=0 ; k<nzd ; k++ )
{
    zd[k] = 0.1 * (double)k;
}

ix=1;
iy=1;
am = 0.0;
sg = 0.1;

ierr = ASL_djdbno(nxd*nyd*nzd, am, sg, &ix, &iy, e);

for( k=0 ; k<nzd ; k++ )
{
for( j=0 ; j<nyd ; j++ )
{
for( i=0 ; i<nxd ; i++ )
{
    fd[i+j*nxd+k*nxd*nyd] = 10.0
    +1.0/(0.03+(xd[i]-0.5)*(xd[i]-0.5))
    +1.0/(0.04+(yd[j]-0.2)*(yd[j]-0.2))
    +1.0/(0.05+(zd[k]-0.6)*(zd[k]-0.6))
    +e[k*nxd*nyd+j*nxd+i];
}
}
}

for( i=0 ; i<nxd+mx ; i++ )
{
    fscanf( fp, "%lf", &xk[i] );
}
for( j=0 ; j<nyd+my ; j++ )
{
    fscanf( fp, "%lf", &yk[j] );
}
for( k=0 ; k<nzd+mz ; k++ )
{
    fscanf( fp, "%lf", &zk[k] );
}

for( i=0 ; i<nnx ; i++ )
{
    xx[i] = 0.1 * (double)i;
}
for( j=0 ; j<nny ; j++ )
{
    yy[j] = 0.1 * (double)j;
}
for( k=0 ; k<nnz ; k++ )
{
    zz[k] = 0.1 * (double)k;
}

fclose( fp );

ierr = ASL_dgiss3(xd, nxd, yd, nyd, zd, nzd, fd, xk, mx, yk, my,
     zk, mz, xx, nnx, yy, nny, zz, nnz, s, rs, &aic, wk, iwk);

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

if( ierr < 3000 )
{
    printf( "\n\taic  = %8.3g\n", aic );
    sumt2 = 0.0;
    sumd2 = 0.0;
    for( i=0 ; i<nnx ; i++ )
    {
    for( j=0 ; j<nny ; j++ )
    {
    for( k=0 ; k<nnz ; k++ )
    {
        ff = 10.0
           +1.0/(0.03+(xd[i]-0.5)*(xd[i]-0.5))
           +1.0/(0.04+(yd[j]-0.2)*(yd[j]-0.2))
```

```
                    +1.0/(0.05+(zd[k]-0.6)*(zd[k]-0.6));
                sumt2 += ff * ff;
                sumd2 += (ff-s[i+j*nnx+k*nnx*nny])
                       * (ff-s[i+j*nnx+k*nnx*nny]);
            }
            }
            }
            sumt2 = sqrt(sumt2);
            sumd2 = sqrt(sumd2);
            fit = 100.0;
            if( ( sumt2 != 0.0 )||( sumd2 != 0.0 ) )
            {
                fit = ( sumt2 / ( sumt2 + sumd2 ) ) * 100.0;
            }
            printf( "\n\tFitting rate = %8.3g\n", fit );
        }

        free( iwk );
        free( xd );
        free( yd );
        free( zd );
        free( fd );
        free( xk );
        free( yk );
        free( zk );
        free( xx );
        free( yy );
        free( zz );
        free( s );
        free( rs );
        free( e );
        free( wk );

        return 0;
    }
```

(d) 出力結果

```
*** ASL_dgiss3 ***

 ** Input **

nxd =      9 mx =      4 nnx =      9
nyd =      9 my =      4 nny =      9
nzd =      9 mz =      4 nnz =      9

 ** Output **

ierr =      0

aic  = -3.98e+03

Fitting rate =     99.8
```

付 録 A　　ASL で使用している計算機依存定数

A.1　誤差判定のための単位

ASL では, 浮動小数点演算における誤差判定のための単位として次の値を設定している. 誤差判定のための単位は, 浮動小数点データの内部表現によって決まる数値であり, ASL C 言語インタフェース ではこの単位を収束判定, 零判定などに用いることがある.

表 A−1　誤差判定のための単位

単精度演算	倍精度演算
$2^{-23}(\simeq 1.19\times 10^{-7})$	$2^{-52}(\simeq 2.22\times 10^{-16})$

備考 誤差判定の単位 ε はマシン ε と呼ばれることもあり, 通常, 対応する浮動小数点形式で $1+\varepsilon$ の計算結果が 1 と異なるような最小の正の定数として定義される. したがって, 誤差判定の単位を見れば, その浮動小数点形式での (仮数部の) 演算の最大有効桁数がわかる.

A.2　浮動小数点データの値の最大値・最小値

ASL の内部で定義している浮動小数点データの値の最大値, 最小値を以下に示す.
なお, 以下の最大値, 最小値はハードウェアが実際に採用している浮動小数点形式のそれとは異なる場合があるので注意されたい.

表 A−2　浮動小数点データの値の最大値・最小値

	単精度演算	倍精度演算
最大値	$2^{127}(2-2^{-23})$ $(\simeq 3.40\times 10^{38})$	$2^{1023}(2-2^{-52})$ $(\simeq 1.80\times 10^{308})$
正の最小値	2^{-126} $(\simeq 1.17\times 10^{-38})$	2^{-1022} $(\simeq 2.23\times 10^{-308})$
負の最大値	-2^{-126} $(\simeq -1.17\times 10^{-38})$	-2^{-1022} $(\simeq -2.23\times 10^{-308})$
最小値	$-2^{127}(2-2^{-23})$ $(\simeq -3.40\times 10^{38})$	$-2^{1023}(2-2^{-52})$ $(\simeq -1.80\times 10^{308})$

索 引

アプリケーションシステム

科学技術計算ライブラリ
ASL C 言語インタフェース
ユーザーズガイド

〈 基本機能編 第 4 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