

FUJITSU Software

A horizontal band featuring a red abstract graphic with flowing, curved lines and bright light spots, creating a sense of motion and energy.

FUJITSU c-SSL II 使用手引書

(科学用関数ライブラリ)

J2UL-1907-01Z0(01)

2015年 9月

まえがき

本マニュアルは、C 言語から利用できる科学用関数ライブラリ C-SSLII (C-Scientific Subroutine Library II) の使用方法について記述しています。

C-SSL II は、パーソナルコンピュータからベクトル計算機にわたる種々のシステムで利用できるよう設計されています。利用者プログラムと C-SSL II のインタフェースは、システムが異なってもすべて同一になっています。したがって、本マニュアルは C-SSL II が提供される各種システムで共通に利用されることを意図して記述しています。

本書の構成は以下のとおりです。

本マニュアルの読み方

本マニュアルで使用しているフォントや記号の意味を説明します。

C-SSL II 一覧表

C-SSL II で提供される関数を機能ごとに分類した一覧表です。この中に書かれているページ番号から、個々のルーチンの使用方法のページを簡単に見つけることができます。オンライン化されたマニュアルの利用者は、一覧表のリンクをたどるだけで、関数の使用方法のページを開くことができます。

第 1 章 概説

C-SSL II を使用するために必要な次の内容について記述しています。C-SSL II を最初に使う前に本章をよく読んで使い方を理解してからお使いください。

- C-SSL II の一般規約
- データの格納方法
- 関数の使用方法の読み方
- 数値計算の分野ごとの C-SSL II ルーチンの概要

第 2 章 関数の使用方法

個々の関数の使い方について記述しています。使用方法の説明は、関数名のアルファベット順に編集されています。C-SSL II の関数ごとに概要と引数の説明、サンプルプログラム、使用上の注意について述べています。

付録 A 補助ルーチン

補助ルーチンの使い方について記述しています。

付録 B 参考文献一覧

SSL II で使用している理論において重要な文献を載せています。本文の中で参照する場合には [10] のように文献の番号を記述しています。

C-SSL II の中で使用されている手法の詳細については以下の Fortran 用 SSL II の使用手引書を参照してください。

“富士通 SSL II 使用手引書”

“FUJITSU SSL II 拡張機能使用手引書”

“FUJITSU SSL II 拡張機能使用手引書 II”

本書では C 言語用の SSL II を C-SSL II と呼び, Fortran 言語用の SSL II は区別のため Fortran SSL II と呼んでいます.

C-SSL II は最新の技術を維持するために, 改良並びに新規追加がなされることがあります. また, 改良若しくは新規追加した関数が, 既存の関数の機能を包含し, 性能的にまさる場合には, 一定の猶予期間をおいて, 既存の関数を削除することがありますので, あらかじめご承知おきください.

Fortran SSL II の開発にあたっては, 下記に示した関係各所の深大な御協力, 御指導をいただきました. ここに深く感謝致します.

北海道大学大型計算機センタ
名古屋大学大型計算機センタ
京都大学大型計算機センタ
九州大学大型計算機センタ
日本原子力研究所核設計研究室
Australian National University
(順不同, 敬称略)

Fortran SSL II 著作者氏名

田 中 正 次	Professor Richard Peirce Brent
二 宮 市 三	Dr Andrew James Cleary
鳥 居 達 生	Dr Murray Leslie Dow
長谷川 武光	Mr Christopher Robert Dum
秦 野 和 郎	Dr Lutz Grosz
秦 野 甯 世	Dr David Lawrence Harrar II
吉 田 年 雄	Dr Markus Hegland
刀 根 薫	Ms Judith Helen Jenkinson
古 林 隆	Dr Margaret Helen Kahn
細 野 敏 夫	Mr Geoffrey Keating
	Dr Zbigniew Leyk
	Mr Gavin John Mercer
	Mr David John Miron
	Mr Ole Møller Nielsen
	Professor Michael Robert Osborne
	Dr Peter Frederick Price
	Dr Stephen Gwyn Roberts
	Dr David Barry Singleton
	Dr David Edward Stewart

(順不同, 敬称略)

C-SSL II は Fortran SSL II を基に, FECIT (Fujitsu European Centre for Information Technology, England)との協力の
のもと, 開発されました.

輸出管理規制について

本ドキュメントを輸出または第三者へ提供する場合は、お客様が居住する国および米国輸出管理関連法規等の規制をご確認のうえ、必要な手続きをおとりください。

出版年月および版数

版数	マニュアルコード
2015 年 9 月 第 10 版	J2UL-1907-01Z0(01)
2014 年 10 月 第 9 版	J2UL-1907-01Z0(00)
2013 年 6 月 第 8 版	—
2013 年 3 月 第 7 版	—
2006 年 3 月 第 6 版	—
2002 年 12 月 第 5 版	—
2001 年 1 月 第 4 版	—
1999 年 9 月 第 3 版	—
1999 年 1 月 第 2 版	—
1997 年 12 月 初版	—

著作権表示

Copyright FUJITSU LIMITED 1997-2015

変更履歴

変更内容	変更箇所	版数
以下のルーチンを追加しました ・ c_dvcft3	C-SSLII 一覧表, 1.12, 第 2 章	第 7 版
Neumann 型前処理を利用する場合の使用上の注意を追加	c_dvcgd, c_dvcge	第 8 版
体裁の見直し	表紙, まえがき	第 9 版
作業領域 w についての使用上の注意を追加	c_dvcfm1, c_dvcft3	第 10 版
誤記訂正	c_dvcpf1	

お願い

- 本書を無断で他に転載しないようお願いします。
- 本書は予告なしに変更されることがあります。

本マニュアルの読み方

ここでは本マニュアルで使用しているフォントや記号の意味について説明します。

フォントの意味

本マニュアルの中で使用している英数字は、次の表のように、フォントによって意味が異なります。

フォント	内 容	例
Courier	C 言語プログラムに関するもの、変数名やプログラムリストなど。	<code>a[i][j]</code>
Times	一般の英字。 数学の関数名。	C-SSL II, Fortran <code>exp</code> , <code>max</code>
<i>Times italic</i>	数学の変数や行列、ベクトルの要素。	x , a_{ij}
Times bold	行列やベクトル。	$\mathbf{Ax}=\mathbf{b}$

数学記号について

数学の行列と C 言語の配列は次のように違っているので注意が必要です。

- 数学では、行列やベクトルの要素の添字は 1 から始まります。行列 $\mathbf{A}$ の最初の要素は a_{11} です。
- C 言語の 2 次元配列や 1 次元配列の添字は 0 から始まります。2 次元配列 `a` の最初の要素は `a[0][0]` です。

数式の中で使われる i は、虚数単位を表わします。例えば $z=5+i10$ と書いたときは $i=\sqrt{-1}$ です。

$|x|$ は x の絶対値を表わします。 $\|\mathbf{x}\|$ は $\mathbf{x}$ のノルムを表わします。

C-SSL II 一覧表

C-SSL II で提供される関数を機能ごとに分類した一覧表です。この中に書かれているページ番号から、個々のルーチンの使用方法のページを簡単に見つけることができます。オンライン化されたマニュアルの利用者は、関数名をクリックするだけで、関数の使用方法のページを開くことができます。

A. 線型計算

行列格納方法の変換

関数名	項 目	ページ
c_dcgsm	行列格納方法の変換 (一般行列の格納方法→対称行列用圧縮格納法)	261
c_dcsgm	行列格納方法の変換 (対称行列用圧縮格納法→一般行列の格納方法)	294
c_dcgsgbm	行列格納方法の変換 (一般行列の格納方法→対称バンド行列用圧縮格納法)	259
c_dcsbgm	行列格納方法の変換 (対称バンド行列用圧縮格納法→一般行列の格納方法)	290
c_dcsgbm	行列格納方法の変換 (対称行列用圧縮格納法→対称バンド行列用圧縮格納法)	297
c_dcsbgm	行列格納方法の変換 (対称バンド行列用圧縮格納法→対称行列用圧縮格納法)	292

行列操作

関数名	項 目	ページ
c_daggm	行列の和 (実行列)	84
c_dsggm	行列の差 (実行列)	498
c_dvmggm	行列の積 (実行列)	662
c_dmav	実行列と実ベクトルの積	424
c_dmcv	複素行列と複素ベクトルの積	426
c_dvmvsd	実スパース行列と実ベクトルの積 (対角形式格納法)	675
c_dvmvse	実スパース行列と実ベクトルの積 (ELLPACK 形式格納法)	677
c_dmsgm	行列の積 (実対称行列・実行列)	440
c_dassm	行列の和 (実対称行列)	123
c_dsssm	行列の差 (実対称行列)	511
c_dmssm	行列の積 (実対称行列)	442
c_dmsgm	行列の積 (実行列・実対称行列)	430
c_dmsv	実対称行列と実ベクトルの積	444
c_dmsbv	実対称バンド行列と実ベクトルの積	438
c_dvmbv	実バンド行列と実ベクトルの積	650

連立 1 次方程式

関数名	項 目	ページ
c_dvlax	実行列の連立 1 次方程式 (ブロッキング LU 分解法)	615
c_dl原因	複素行列の連立 1 次方程式 (クラウト法)	391
c_dlsix	実対称行列の連立 1 次方程式 (ブロック対角ピボッティング手法)	413
c_dvl原因	正値対称行列の連立 1 次方程式 (変形コレスキー法)	631
c_dvl原因	実バンド行列の連立 1 次方程式 (ガウス消去法)	618
c_dvl原因	正値対称行列の連立 1 次方程式 (ブロック化されたコレスキー分解法)	628
c_dvl原因	正値対称バンド行列の連立 1 次方程式 (変形コレスキー法)	625
c_dls原因	実対称バンド行列の連立 1 次方程式 (ブロック対角ピボッティング手法)	410

関数名	項 目	ページ
c_dlstx	正値対称 3 項行列の連立 1 次方程式 (変形コレスキー法)	416
c_dltx	実 3 項行列の連立 1 次方程式 (ガウス消去法)	419
c_dvcgd	正値対称スパース行列の連立 1 次方程式 (前処理付き CG 法, 対角形式格納法)	579
c_dvcge	正値対称スパース行列の連立 1 次方程式 (前処理付き CG 法, ELLPACK 形式格納法)	583
c_dvtfqd	非対称又は不定値のスパース実行列の連立 1 次方程式 (TFQMR 法, 対角形式格納法)	738
c_dvtfqe	非対称又は不定値のスパース実行列の連立 1 次方程式 (TFQMR 法, ELLPACK 形式格納法)	741
c_dvqmrld	非対称又は不定値のスパース実行列の連立 1 次方程式 (QMR 法, 対角形式格納法)	679
c_dvqmre	非対称又は不定値のスパース実行列の連立 1 次方程式 (QMR 法, ELLPACK 形式格納法)	682
c_dvcrd	非対称又は不定値のスパース実行列の連立 1 次方程式 (MGCR 法, 対角形式格納法)	597
c_dvcre	非対称又は不定値のスパース実行列の連立 1 次方程式 (MGCR 法, ELLPACK 形式格納法)	600
c_dvbcsd	非対称又は不定値のスパース実行列の連立 1 次方程式 (BICGSTAB(<i>l</i>)法, 対角形式格納法)	544
c_dvbcse	非対称又は不定値のスパース実行列の連立 1 次方程式 (BICGSTAB(<i>l</i>)法, ELLPACK 形式格納法)	547
c_dvltqr	実 3 項行列の連立 1 次方程式 (QR 分解)	634
c_dvltx	実 3 項行列の連立 1 次方程式 (サイクリック・リダクション法)	636
c_dvltx1	定数型実 3 項行列の連立 1 次方程式 (ディリクレ型, サイクリック・リダクション法)	639
c_dvltx2	定数型実 3 項行列の連立 1 次方程式 (ノイマン型, サイクリック・リダクション法)	642
c_dvltx3	定数型実 3 項行列の連立 1 次方程式 (周期型, サイクリック・リダクション法)	645

逆行列

関数名	項 目	ページ
c_dvluiv	LU 分解された実行列の逆行列	648
c_dcluiv	LU 分解された複素行列の逆行列	279
c_dvldiv	LDL ^T 分解された正値対称行列の逆行列	621

行列の三角分解

関数名	項 目	ページ
c_dvalu	実行列の LU 分解 (ブロッキング LU 分解法)	541
c_dclu	複素行列の LU 分解 (クラウト法)	276
c_dsmmdm	実対称行列の MDM ^T 分解 (ブロック対角ピボッティング手法)	504
c_dvslldl	正値対称行列の LDL ^T 分解	721
c_dvblu	実バンド行列の LU 分解 (ガウス消去法)	555
c_dsbmdm	実対称バンド行列の MDM ^T 分解 (ブロック対角ピボッティング手法)	490
c_dvspdll	正値対称行列の LL ^T 分解 (ブロック化されたコレスキー分解法)	724
c_dvbldl	正値対称バンド行列の LDL ^T 分解 (変形コレスキー法)	550

三角分解された連立 1 次方程式

関数名	項 目	ページ
c_dlux	LU 分解された実行列の連立 1 次方程式	422
c_dclux	LU 分解された複素行列の連立 1 次方程式	282
c_dmdmx	MDM ^T 分解された実対称行列の連立 1 次方程式	428
c_dvsplx	LL ^T 分解された正値対称行列の連立 1 次方程式	727
c_dvldlx	LDL ^T 分解された正値対称行列の連立 1 次方程式	623
c_dvblux	LU 分解された実バンド行列の連立 1 次方程式	558
c_dbmdmx	MDM ^T 分解された実対称バンド行列の連立 1 次方程式	192
c_dvbldx	LDL ^T 分解された正値対称バンド行列の連立 1 次方程式	553

最小二乗解

関数名	項 目	ページ
c_dlaxl	実行列の最小二乗解 (ハウスホルダー変換)	385
c_dlaxlm	実行列の最小二乗最小ノルム解 (特異値分解法)	388
c_dginv	実行列の一般逆行列 (特異値分解法)	338
c_dasvd1	実行列の特異値分解 (ハウスホルダー法, QR 法)	125

B. 固有値固有ベクトル

固有値固有ベクトル

関数名	項 目	ページ
c_deig1	実行列の固有値及び固有ベクトル (2 段 QR 法)	306
c_dceig2	複素行列の固有値及び固有ベクトル (QR 法)	251
c_dseig1	実対称行列の固有値及び固有ベクトル (QL 法)	494
c_dvseg2	実対称行列の固有値及び固有ベクトル (並列バイセクション法, 逆反復法)	711
c_dvsevp	実対称行列の固有値及び固有ベクトル (三重対角化, マルチセクション法, 逆反復法)	714
c_dheig2	エルミート行列の固有値及び固有ベクトル (ハウスホルダー法, バイセクション法, 逆反復法)	349
c_dvhevp	エルミート行列の固有値及び固有ベクトル (三重対角化, マルチセクション法, 逆反復法)	606
c_dbseg	実対称バンド行列の固有値及び固有ベクトル (ルティスハウザー・シュワルツ法, バイセクション法, 逆反復法)	211
c_dbsegj	実対称バンド行列の固有値及び固有ベクトル (ジェニングス法)	214
c_dvland	実対称スパース行列の固有値及び固有ベクトル (Lanczos 法, 対角形式格納法)	610
c_dvtdev	実 3 重対角行列の固有値及び固有ベクトル	734
c_dteig1	実対称 3 重対角行列の固有値及び固有ベクトル (QL 法)	513
c_dteig2	実対称 3 重対角行列の固有値及び固有ベクトル (バイセクション法, 逆反復法)	515
c_dvgsg2	実対称行列の一般固有値及び固有ベクトル (並列バイセクション法, 逆反復法)	603
c_dgbseg	実対称バンド行列の一般固有値及び固有ベクトル (ジェニングス法)	331

固有値

関数名	項 目	ページ
c_dhsqr	実ヘッセンベルグ行列の固有値 (2 段 QR 法)	355
c_dchsqr	複素ヘッセンベルグ行列の固有値 (QR 法)	269
c_dtrql	実対称 3 重対角行列の固有値 (QL 法)	528
c_dbsct1	実対称 3 重対角行列の固有値 (バイセクション法)	208

固有ベクトル

関数名	項 目	ページ
c_dhvec	実ヘッセンベルグ行列の固有ベクトル (逆反復法)	357
c_dchvec	複素ヘッセンベルグ行列の固有ベクトル (逆反復法)	271
c_dbsvect	実対称バンド行列の固有ベクトル (逆反復法)	224

その他

関数名	項 目	ページ
c_dblnc	実行列の平衡化	189
c_dcblnc	複素行列の平衡化	246
c_dhes1	実行列の実ヘッセンベルグ行列への変換 (ハウスホルダー法)	351
c_dches2	複素行列の複素ヘッセンベルグ行列への変換 (安定化基本相似変換)	267
c_dhbk1	実行列の固有ベクトルへの逆変換と正規化	346
c_dchbk2	複素行列の固有ベクトルへの逆変換	264
c_dtrid1	実対称行列の実対称 3 重対角行列への変換 (ハウスホルダー法)	524
c_dtridh	エルミート行列の実対称 3 重対角行列への変換 (ハウスホルダー法, 対角ユニタリ変換)	526
c_dbtrid	実対称バンド行列の実対称 3 重対角行列への変換 (ルティスハウザー・シュワルツ法)	226
c_dtrbk	実対称行列の固有ベクトルへの逆変換	519
c_dtrbkh	エルミート行列の固有ベクトルへの逆変換	521
c_dnrml	実行列の固有ベクトルの正規化	466
c_dcnrml	複素行列の固有ベクトルの正規化	284
c_dgschl	一般形から標準形への変換 (実対称行列の一般固有値問題)	343
c_dgsbk	一般形の固有ベクトルへの逆変換 (実対称行列の一般固有値問題)	341

C. 非線型計算

関数名	項 目	ページ
c_drqdr	実係数 2 次方程式	488
c_dcqdr	複素係数 2 次方程式	288
c_dlowp	実係数低次代数方程式 (5 次以下)	403
c_drjetr	実係数高次代数方程式 (ジェンキンス・トラウブの方法)	486
c_dcjart	複素係数高次代数方程式 (ヤラット法)	274
c_dtsd1	実超越方程式 $f(x)=0$ (プレント法)	530
c_dtsdm	実超越方程式 $f(x)=0$ (マラー法)	532
c_dctsdm	複素超越方程式 $f(z)=0$ (マラー法)	299
c_dnolbr	連立非線型方程式 (プレント法)	455

D. 極値問題

関数名	項 目	ページ
c_dlminf	1 変数関数の極小化 (微係数不要, 2 次補間法)	397
c_dlmng	1 変数関数の極小化 (微係数要, 3 次補間法)	400
c_dminf1	多変数関数の極小化 (微係数不要, 改訂準ニュートン法)	432
c_dming1	多変数関数の極小化 (微係数要, 準ニュートン法)	435
c_dnolf1	関数二乗和の極小化 (微係数不要, 改訂マルカート法)	459

関数名	項 目	ページ
c_dnolg1	関数二乗和の極小化 (微係数要, 改訂マルカート法)	462
c_dlprs1	線形計画問題 (改訂シンプレックス法)	405
c_dnlpg1	非線形計画問題 (微係数要, パウエル法)	450

E. 補間・近似

補間

関数名	項 目	ページ
c_daklag	エイトケン・ラグランジュ補間	89
c_dakher	エイトケン・エルミート補間	86
c_dbif1	B-spline 補間式 (I) による補間, 数値微分, 数値積分	149
c_dbif2	B-spline 補間式 (II) による補間, 数値微分, 数値積分	152
c_dbif3	B-spline 補間式 (III)による補間, 数値微分, 数値積分	155
c_dbif4	B-spline 補間式 (IV) による補間, 数値微分, 数値積分	158
c_dbifd1	B-spline 2 次元補間式 (I-I) による補間, 数値微分, 数値積分	161
c_dbifd3	B-spline 2 次元補間式 (III-III)による補間, 数値微分, 数値積分	165
c_dakmid	2 次元準エルミート補間式による補間	92
c_dakmin	準エルミート補間式	95
c_dbic1	B-spline 補間式 (I)	132
c_dbic2	B-spline 補間式 (II)	135
c_dbic3	B-spline 補間式 (III)	138
c_dbic4	B-spline 補間式 (IV)	140
c_dbicd1	B-spline 2 次元補間式 (I-I)	142
c_dbicd3	B-spline 2 次元補間式 (III-III)	146

近似

関数名	項 目	ページ
c_dlesq1	最小二乗近似多項式	394

平滑化

関数名	項 目	ページ
c_dsmle1	最小二乗近似多項式による平滑化 (等間隔離散点)	507
c_dsmle2	最小二乗近似多項式による平滑化 (不等間隔離散点)	509
c_dbsf1	B-spline 平滑化式による平滑化, 数値微分, 数値積分	217
c_dbsc1	B-spline 平滑化式 (固定節点)	195
c_dbsc2	B-spline 平滑化式 (節点追加方式)	199
c_dbsfd1	B-Spline 2 次元平滑化式による平滑化, 数値微分, 数値積分	220
c_dbscd2	B-spline 2 次元平滑化式 (節点追加方式)	203

級数

関数名	項 目	ページ
c_dfcheb	実関数のチェビシェフ級数展開 (関数入力, 高速 cosine 変換)	313
c_decheb	チェビシェフ級数の求和	302
c_dgcheb	チェビシェフ級数の導関数	335
c_dicheb	チェビシェフ級数の不定積分	361
c_dfcosf	偶関数の cosine 級数展開 (関数入力, 高速 cosine 変換)	317
c_decosp	cosine 級数の求和	304

関数名	項 目	ページ
c_dfstnf	奇関数の sine 級数展開 (関数入力, 高速 sine 変換)	324
c_desinp	sine 級数の求和	309

F. 変換

関数名	項 目	ページ
c_dvcfm1	1 次元離散型複素フーリエ変換 (2, 3, 5 及び 7 の混合基底)	566
c_dvmcf2	1 次元, 多重, 多次元離散型複素フーリエ変換 (複素配列格納, 混合基底)	652
c_dvmcft	1 次元, 多重, 多次元離散型複素フーリエ変換 (実虚別配列, 混合基底)	655
c_dvmrf2	1 次元, 多重, 多次元離散型実フーリエ変換 (混合基底)	664
c_dvmrft	多重, 多次元離散型実フーリエ変換 (2, 3 及び 5 の混合基底)	668
c_dvsrft	1 次元, 多重離散型実フーリエ変換 (2, 3 及び 5 の混合基底)	730
c_dvcft1	離散型複素フーリエ変換 (2 基底 FFT)	569
c_dvcft2	離散型複素フーリエ変換 (メモリ節約型, 2 基底 FFT)	573
c_dvcft3	1 次元離散型複素フーリエ変換 (2 基底, 一定間隔を持つデータ)	576
c_dvrft1	離散型実フーリエ変換 (2 基底 FFT)	699
c_dvrft2	離散型実フーリエ変換 (メモリ節約型, 2 基底 FFT)	703
c_dvmcst	離散型 cosine 変換	659
c_dvcos1	離散型 cosine 変換 (2 基底 FFT)	587
c_dfcosm	離散型 cosine 変換 (中点公式, 2 基底 FFT)	321
c_dvmsnt	離散型 sine 変換	672
c_dvsin1	離散型 sine 変換 (2 基底 FFT)	718
c_dfsinm	離散型 sine 変換 (中点公式, 2 基底 FFT)	328
c_dvcpf1	1 次元素因子離散型複素フーリエ変換	590
c_dvcpf3	3 次元素因子離散型複素フーリエ変換	593
c_dvrpf3	3 次元素因子離散型実フーリエ変換	707
c_dvccvf	複素データの離散畳み込み及び離散相関	561
c_dvrcvf	実データの離散畳み込み及び離散相関	694
c_dvwflt	ウェーブレットフィルターの生成	744
c_dv1dwt	1 次元ウェーブレット変換	535
c_dv2dwt	2 次元ウェーブレット変換	538
c_dlaps1	ラプラス変換 (複素右半平面で正則な有理関数)	376
c_dlaps2	ラプラス変換 (一般の有理関数)	379
c_dlaps3	ラプラス変換 (一般関数)	382
c_dhrwiz	Hurwitz 多項式の判定	353

G. 数値積分

関数名	項 目	ページ
c_dsimp1	1 次元有限区間積分 (等間隔離散点入力, シンプソン則)	500
c_dtrap	1 次元有限区間積分 (不等間隔離散点入力, 台形則)	517
c_daqn9	1 次元有限区間積分 (関数入力, 適応型ニュートン・コーツ 9 点則)	120
c_daqc8	1 次元有限区間積分 (関数入力, クレンショー・カーチス型積分法)	97
c_daqe	1 次元有限区間積分 (関数入力, 二重指数関数型積分公式)	100
c_daqeh	1 次元半無限区間積分 (関数入力, 二重指数関数型積分公式)	104
c_daqei	1 次元全無限区間積分 (関数入力, 二重指数関数型積分公式)	107
c_daqmc8	多次元有限領域積分 (関数入力, クレンショー・カーチス型積分法)	110

関数名	項 目	ページ
c_daqme	多次元積分 (関数入力, 二重指数関数型積分公式)	115

H. 微分方程式

関数名	項 目	ページ
c_dodge	ステイフ連立 1 階常微分方程式 (ギア法)	475
c_dodam	連立 1 階常微分方程式 (アダムス法)	469
c_dodrk1	連立 1 階常微分方程式 (ルンゲ・クッタ・ヴァーナー法)	482

I. 特殊関数

関数名	項 目	ページ
c_dceli1	第 1 種完全楕円積分 $K(x)$	253
c_dceli2	第 2 種完全楕円積分 $E(x)$	255
c_dexpi	指数積分 $E_i(x)$, $\bar{E}_i(x)$	311
c_dsini	正弦積分 $S_i(x)$	502
c_dcosi	余弦積分 $C_i(x)$	286
c_dsfri	正弦フレネル積分 $S(x)$	496
c_dcfri	余弦フレネル積分 $C(x)$	257
c_digam1	第 1 種不完全ガンマ関数 $\gamma(v, x)$	368
c_digam2	第 2 種不完全ガンマ関数 $\Gamma(v, x)$	370
c_dierf	逆誤差関数 $\operatorname{erf}^{-1}(x)$	364
c_dierfc	逆余誤差関数 $\operatorname{erfc}^{-1}(x)$	366
c_dbj0	第 1 種 0 次ベッセル関数 $J_0(x)$	173
c_dbj1	第 1 種 1 次ベッセル関数 $J_1(x)$	175
c_dby0	第 2 種 0 次ベッセル関数 $Y_0(x)$	228
c_dby1	第 2 種 1 次ベッセル関数 $Y_1(x)$	230
c_dbi0	第 1 種 0 次変形ベッセル関数 $I_0(x)$	128
c_dbi1	第 1 種 1 次変形ベッセル関数 $I_1(x)$	130
c_dbk0	第 2 種 0 次変形ベッセル関数 $K_0(x)$	181
c_dbk1	第 2 種 1 次変形ベッセル関数 $K_1(x)$	183
c_dbjn	第 1 種整数次ベッセル関数 $J_n(x)$	177
c_dbyn	第 2 種整数次ベッセル関数 $Y_n(x)$	232
c_dbin	第 1 種整数次変形ベッセル関数 $I_n(x)$	169
c_dbkn	第 2 種整数次変形ベッセル関数 $K_n(x)$	185
c_dcbn	複素変数第 1 種整数次変形ベッセル関数 $I_n(z)$	236
c_dcbkn	複素変数第 2 種整数次変形ベッセル関数 $K_n(z)$	243
c_dcbjn	複素変数第 1 種整数次ベッセル関数 $J_n(z)$	238
c_dcbyn	複素変数第 2 種整数次ベッセル関数 $Y_n(z)$	249
c_dbjr	第 1 種実数次ベッセル関数 $J_\nu(x)$	179
c_dbyr	第 2 種実数次ベッセル関数 $Y_\nu(x)$	234
c_dbir	第 1 種実数次変形ベッセル関数 $I_\nu(x)$	171
c_dbkr	第 2 種実数次変形ベッセル関数 $K_\nu(x)$	187
c_dcbjr	複素変数第 1 種実数次ベッセル関数 $J_\nu(z)$	240
c_dndf	正規分布関数 $\phi(x)$	446
c_dndfc	余正規分布関数 $\psi(x)$	448
c_dindf	逆正規分布関数 $\phi^{-1}(x)$	372

関数名	項 目	ページ
c_dindfc	逆余正規分布関数 $\psi^{-1}(x)$	374

J. 乱数

関数名	項 目	ページ
c_dvrau4	一様乱数 $[0, 1)$ の生成	691
c_dvran3	正規乱数の生成	685
c_dvran4	正規乱数の生成 (Wallace 法)	688
c_rane2	指数乱数の生成 (単精度)	748
c_ranp2	ポアソン乱数の生成 (単精度)	750
c_ranb2	二項乱数の生成 (単精度)	746

K. 補助ルーチン

関数名	項 目	ページ
c_dmach	丸め誤差の単位 (unit round-off)	758
c_dsum	積和計算 (実ベクトル)	759
c_dcsum	積和計算 (複素ベクトル)	754
c_iradix	浮動小数点演算の基底	761
c_dfmax	浮動小数点数の正の最大値	756
c_dfmin	浮動小数点数の正の最小値	757

目次

第1章 概説	1	1.8.8 実3項行列の固有値固有ベクトル	37
1.1 背景	2	1.8.9 実対称行列の一般固有値固有ベクトル	38
1.2 C-SSL II の一般規約	2	1.8.10 実対称バンド行列の一般固有値固有ベクトル	39
1.2.1 関数名	2	1.9 非線型計算	40
1.2.2 引数	3	1.9.1 概要	40
1.2.3 戻り値	4	1.9.2 代数方程式	40
1.2.4 標準ヘッダファイル	4	1.9.3 超越方程式	41
1.2.5 メインプログラムの関数名	4	1.9.4 連立非線型方程式	42
1.2.6 多次元配列	4	1.10 極値問題	43
1.2.7 ユーザ関数	5	1.10.1 概要	43
1.2.8 複素数	6	1.10.2 1変数関数の極小化	43
1.2.9 コンディションコード	6	1.10.3 制約なし多変数関数の極小化	44
1.2.10 プログラムの例	7	1.10.4 制約なし関数二乗和の極小化(非線型最小二乗解)	46
1.3 データの格納方法	8	1.10.5 線形計画問題	47
1.3.1 一般行列の格納方法	8	1.10.6 非線形計画問題(制約付き多変数関数の極小化)	49
1.3.2 対称行列の格納方法	9	1.11 補間・近似	49
1.3.3 エルミート行列の格納方法	10	1.11.1 概要	49
1.3.4 バンド行列の格納方法	11	1.11.2 スプライン (Spline) 関数について	50
1.3.5 一般スパース行列の格納方法	13	1.11.3 補間	57
1.3.6 正値対称スパース行列の格納方法	14	1.11.4 近似	58
1.4 丸め誤差の単位	16	1.11.5 平滑化	58
1.5 計算機定数	16	1.11.6 級数	59
1.6 関数の使用方法の読み方	16	1.12 変換	60
1.7 線型計算	19	1.12.1 概要	60
1.7.1 概要	19	1.12.2 離散型実フーリエ変換	60
1.7.2 行列格納方法の変換	19	1.12.3 離散型 cosine 変換	60
1.7.3 行列操作	20	1.12.4 離散型 sine 変換	61
1.7.4 連立1次方程式・逆行列(直接法)	21	1.12.5 離散型複素フーリエ変換	62
1.7.5 連立1次方程式(3項行列)	24	1.12.6 ラプラス変換	64
1.7.6 連立1次方程式の反復法と収束性	25	1.13 数値微積分	69
1.7.7 最小二乗解	27	1.13.1 概要	69
1.8 固有値固有ベクトル	30	1.13.2 数値微分	69
1.8.1 概要	30	1.13.3 数値積分	70
1.8.2 実行列の固有値固有ベクトル	31	1.14 微分方程式	73
1.8.3 複素行列の固有値固有ベクトル	32	1.14.1 概要	73
1.8.4 実対称行列の固有値固有ベクトル	33	1.14.2 常微分方程式	74
1.8.5 エルミート行列の固有値固有ベクトル	35	1.15 特殊関数	77
1.8.6 実対称バンド行列の固有値固有ベクトル	36	1.15.1 概要	77
1.8.7 対称スパース行列の固有値固有ベクトル	37	1.15.2 楕円積分	78
		1.15.3 指数積分	78
		1.15.4 正弦・余弦積分	78

1.15.5 フレネル積分	78	c_dbtrid	226
1.15.6 ガンマ関数	79	c_dby0	228
1.15.7 誤差関数	79	c_dby1	230
1.15.8 ベッセル関数	79	c_dbyn	232
1.15.9 正規分布関数	80	c_dbyr	234
1.16 擬似乱数	81	c_dcbn	236
1.16.1 概要	81	c_dcbjn	238
1.16.2 擬似乱数の生成	81	c_dcbjr	240
第 2 章 関数の使用方法	83	c_dcbkn	243
c_daggm	84	c_dcbnnc	246
c_dakher	86	c_dcbyn	249
c_daklag	89	c_dceig2	251
c_dakmid	92	c_dceli1	253
c_dakmin	95	c_dceli2	255
c_daqc8	97	c_dcfri	257
c_daqe	100	c_dcgsm	259
c_daqeh	104	c_dcgsm	261
c_daqei	107	c_dchbk2	264
c_daqmc8	110	c_dches2	267
c_daqme	115	c_dchsqr	269
c_daqn9	120	c_dchvec	271
c_dassm	123	c_dcjart	274
c_dasvd1	125	c_dclu	276
c_dbi0	128	c_dcluiv	279
c_dbi1	130	c_dclux	282
c_dbic1	132	c_dcnrml	284
c_dbic2	135	c_dcosi	286
c_dbic3	138	c_dcqdr	288
c_dbic4	140	c_dcsbgm	290
c_dbicd1	142	c_dcsbsm	292
c_dbicd3	146	c_dcsgm	294
c_dbif1	149	c_dcsgbm	297
c_dbif2	152	c_dctsdm	299
c_dbif3	155	c_decheb	302
c_dbif4	158	c_decosp	304
c_dbifd1	161	c_deig1	306
c_dbifd3	165	c_desinp	309
c_dbin	169	c_dexpi	311
c_dbir	171	c_dfcheb	313
c_dbj0	173	c_dfcosf	317
c_dbj1	175	c_dfcosm	321
c_dbjn	177	c_dfsinf	324
c_dbjr	179	c_dfsinm	328
c_dbk0	181	c_dgbseg	331
c_dbk1	183	c_dgcheb	335
c_dbkn	185	c_dginv	338
c_dbkr	187	c_dgsbk	341
c_dblnc	189	c_dgschl	343
c_dbmdmx	192	c_dhbk1	346
c_dbsc1	195	c_dheig2	349
c_dbsc2	199	c_dhes1	351
c_dbscd2	203	c_dhrwiz	353
c_dbst1	208	c_dhsqr	355
c_dbseg	211	c_dhvec	357
c_dbsegj	214	c_dicheb	361
c_dbsf1	217	c_dierf	364
c_dbsfd1	220	c_dierfc	366
c_dbsvec	224	c_digam1	368
		c_digam2	370
		c_dindf	372

c_dindfc	374
c_dlaps1	376
c_dlaps2	379
c_dlaps3	382
c_dlaxl	385
c_dlaxlm	388
c_dlcx	391
c_dlesq1	394
c_dlminf	397
c_dlmng	400
c_dlowp	403
c_dlprs1	405
c_dlsbix	410
c_dlsix	413
c_dlstx	416
c_dltx	419
c_dlux	422
c_dmav	424
c_dmcv	426
c_dmdmx	428
c_dmgsn	430
c_dminf1	432
c_dming1	435
c_dmsbv	438
c_dmsgm	440
c_dmssm	442
c_dmsv	444
c_dndf	446
c_dndfc	448
c_dnlpg1	450
c_dnolbr	455
c_dnolf1	459
c_dnolg1	462
c_dnrml	466
c_dodam	469
c_dodge	475
c_dodrk1	482
c_drjetr	486
c_drqdr	488
c_dsbmdm	490
c_dseig1	494
c_dsfri	496
c_dsggm	498
c_dsimp1	500
c_dsini	502
c_dsmdm	504
c_dsmle1	507
c_dsmle2	509
c_dsssm	511
c_dteig1	513
c_dteig2	515
c_dtrap	517
c_dtrbk	519
c_dtrbkh	521
c_dtrid1	524
c_dtridh	526
c_dtrql	528
c_dtsd1	530
c_dtsdm	532
c_dv1dwt	535

c_dv2dwt	538
c_dvalu	541
c_dvbcsd	544
c_dvbcse	547
c_dvbldl	550
c_dvbldx	553
c_dvblu	555
c_dvblux	558
c_dvccvf	561
c_dvcfm1	566
c_dvcft1	569
c_dvcft2	573
c_dvcft3	576
c_dvcgd	576
c_dvcge	583
c_dvcos1	587
c_dvcpf1	590
c_dvcpf3	593
c_dvcrd	597
c_dvcre	600
c_dvgsg2	603
c_dvhevp	606
c_dvland	610
c_dvlax	615
c_dvlbx	618
c_dvldiv	621
c_dvldlx	623
c_dvlsbx	625
c_dvlspx	628
c_dvlsx	631
c_dvltqr	634
c_dvltx	636
c_dvltx1	639
c_dvltx2	642
c_dvltx3	645
c_dvluiv	648
c_dvmbv	650
c_dvmcf2	652
c_dvmcft	655
c_dvmcst	659
c_dvmggm	662
c_dvmrf2	664
c_dvmrft	668
c_dvmsnt	672
c_dvmvsd	675
c_dvmvse	677
c_dvqmrd	679
c_dvqmre	682
c_dvran3	685
c_dvran4	688
c_dvrav4	691
c_dvrcvf	694
c_dvrft1	699
c_dvrft2	703
c_dvrpf3	707
c_dvseg2	711
c_dvsevp	714
c_dvsin1	718
c_dvsldl	721
c_dvspll	724

c_dvsplx	727
c_dvsrft	730
c_dvtdev	734
c_dvtfqd	738
c_dvtfqe	741
c_dvwflt	744
c_ranb2	746
c_rane2	748
c_ranp2	750
付録 A 補助ルーチン	753
A.1 概要	753
c_dcsum	754
c_dfmax	756
c_dfmin	757
c_dmach	758
c_dsum	759
c_iradix	761
付録 B 参考文献一覧	1
索引	769

第1章 概 説

1.1 背景

C-SSL II の主な目的は, Fortran 言語で利用されている SSL II の豊富なサブルーチンと同じ機能を, 高性能を損なうことなく, C 言語の関数として自然なインタフェースで利用できるようにすることである. そのため C-SSL II では C 言語のためのインタフェースルーチンを作成し, その内部では Fortran SSL II ルーチン呼び出すようにしている. C-SSL II ルーチンの関数名や引数の並び順は Fortran SSL II ルーチンに非常に良く似ており, 既に Fortran SSL II ルーチンを直接呼ぶ C 言語プログラムがある場合には簡単に C-SSL II を呼び出すプログラムに変更できる.

C-SSL II は倍精度実数型(double)の変数を扱う関数が用意されている. 単精度実数型(float) 変数を扱う関数としては幾つかの乱数ルーチンのみが用意されている. 複素数のルーチンについては, 専用の複素数型(dcomplex)を定義し, dcomplex 型の変数を扱う関数が用意されている. また引数や関数の復帰値として使用する整数は int 型を採用している.

C-SSL II では Fortran SSL II のほとんどのルーチンを提供している. ただし, Fortran SSL IIの中には, 今や陳腐であり, それに替わって推奨できるルーチンが提供されている場合は, 陳腐なルーチンは C-SSL II では除外した.

C-SSL II では数値計算の分野を次のように区分して, ルーチンを分類している.

- A. 線型計算
- B. 固有値固有ベクトル
- C. 非線型計算
- D. 極値問題
- E. 補間・近似
- F. 変換
- G. 数値積分
- H. 微分方程式
- I. 特殊関数
- J. 乱数
- K. 補助ルーチン

1.2 C-SSL II の一般規約

1.2.1 関数名

C-SSL II の各関数名は次のような規則に基づいてつけられている.

- 関数名の先頭の 2 文字は必ず `c_` である.
- 一部の例外を除いて, 3 文字目には倍精度ルーチンであることを示す `d` の文字がつく.
- Fortran SSL II の拡張機能にあたるルーチンは次に `v` の文字がつく.

図 1.1 に関数名のつけ方の規則を示す。

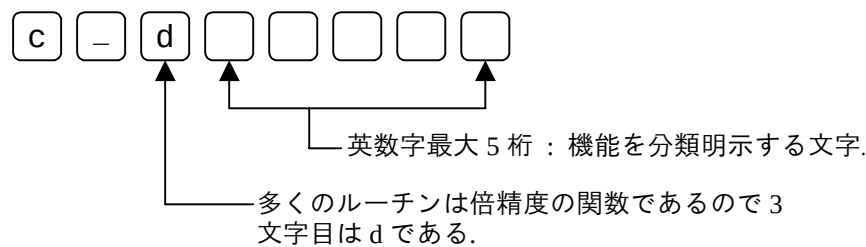


図 1.1 関数名

例えば, `c_dclu` は倍精度複素数の行列に対して LU 分解を行うルーチンである。

1.2.2 引数

C-SSL II ルーチンと利用者のプログラムとのデータの受け渡しは, すべて引数渡しにより行う。

引数の受け渡し方法は次のように一般的な C 言語の引数の受け渡し方法になっている。

- 入力のみの変数は値渡しにより渡す。
- 入出力又は出力を行う変数は, 変数のアドレスを渡す。

各ルーチンの使用方法の説明の中で使用する引数の名前の先頭文字は, 引数の型によって決められている。基本的には Fortran の変数名の規則に従ってつけられている。すなわち, `int` 型の変数は変数名が `i~n` で始まり, `double` 型の変数は変数名が `a~h` 又は `o~y` で始まる。`z` は特別に `dcomplex` 型の変数名の最初に使用している。

引数は利用者側からみたときの入出力関係に応じて, 次の種類がある。

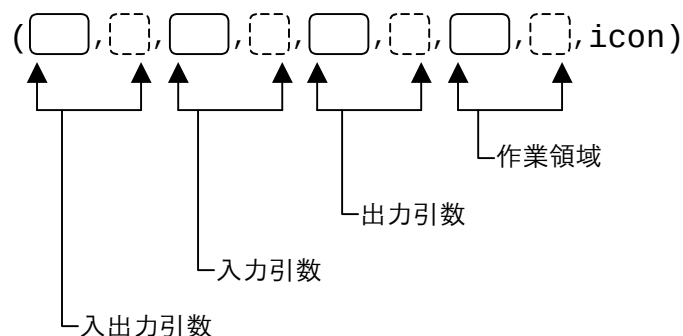
- 入出力引数 : 引数に値を入力する。また, 演算後, 結果が同一領域上に出力される。
- 入力引数 : 引数に値を入力する。演算後内容は保存される。なお, 保存されない場合には, “内容は保存されない”と各関数の説明の項で明記される。
- 出力引数 : 引数に値が出力される。
- 作業領域 : 作業領域として利用される。演算後, 内容は原則として意味を持たない。連続呼び出しが可能な一部のルーチンでは, 2 回目以降に呼び出す場合に前回の作業領域の内容をそのままにしておく必要がある。

さらに, 引数の内容に応じて, 次の種類を定義する。

- 主部 : 数値計算の実質的な対象となるデータを格納する。例えば行列の要素など。
- 制御部 : 数値計算の実質的な対象とならないデータを格納する。例えば行列の次数, 判定値など。

引数の並び順は原則的には, 入出力引数, 入力引数, 出力引数, 作業領域の順で最後に必ず `icon` がある。`icon` とは実行後の状態を示す引数である。各引数の中では主部が先頭にあり, 制御部が続く。C-SSL II ルーチンの引数の並び順は, 基本的には同じ機能を持つ Fortran SSL II と同じ並び順となっている。

図 1.2 に引数の並び順を示す。



□部は主部引数, []部は制御部引数である.
icon は実行後の状態を示す引数である.

図 1.2 引数の並び順

C-SSL II では作業域はユーザが用意した領域を引数に与えるようにしている. 幾つかのルーチンでは何度か同じルーチンを呼び出す場合にこの作業域に格納されたデータを再利用する場合がある.

1.2.3 戻り値

すべての C-SSL II 関数は `int` 型の値を返す. 戻り値は以下のような意味を持つ.

- 0 : 計算が正常に終了した.
- 1 : 演算の途中でエラーが発生したか, 正しい計算結果が得られなかった.

ユーザはプログラムの中で関数の戻り値をチェックし, エラーが起こった場合には, 引数 `icon` を参照すると, より詳細な情報を得ることができる.

1.2.4 標準ヘッダファイル

C-SSL II では標準ヘッダファイルとして `cssl.h` をライブラリと一緒に提供している. このヘッダファイルの中では, すべての関数のプロトタイプ宣言や `dcomplex` の型などが宣言されている. C-SSL II を使用する C 言語プログラムは必ずこのヘッダファイルをインクルードしなければならない.

1.2.5 メインプログラムの関数名

メイン関数の名前は `main` または `MAIN__` (大文字の `MAIN` に続けて 2 つのアンダースコア) を使用する. 一部のシステムでは `main` を使用することができない. これは, C-SSL II は内部で Fortran SSL II を呼び出しており, C 言語と Fortran の結合規則に従う必要があるからである. 結合規則については“C 言語使用手引書”参照.

1.2.6 多次元配列

C-SSL II では行列などの演算において, 配列上でそのデータを処理することが多い. C-SSL II を利用するプログラムでは行列は一般に 2 次元配列に格納する.

2 次元配列を引数として渡す場合, 次のような点に注意する必要がある.

- 配列 `a` は倍精度のポインタに型変換して渡す必要がある. すなわち `(double *) a` と書く.
- `a` の列の数を別の引数として渡す. この値のことを“整合寸法”と呼ぶ.

C-SSL II では行列を配列に格納するための格納方法を幾つか用意している。格納方法の詳細については“1.3 データの格納方法”を参照のこと。

1.2.7 ユーザ関数

C-SSL II のルーチンの中には引数としてユーザ関数を必要とするものがある。

ユーザ関数は C 言語の関数として一般的なインタフェースで作成する。ユーザ関数の結果がスカラー値であれば関数値として返し、結果が 1 次元配列であれば関数は void 型とし、その配列を引数の一つとして返す。

次の例ではユーザ関数 func を c_daqn9 の引数に与えている。

```
/* include C SSL header file */
#include "cssl.h"

/* user function prototype */
double func(double x);

/* user function */
double func(double x)
{
    double res;
    res = x*sin(x);
    return res;
}

/* user's main program */
MAIN__()
{
    int ierr, nmin, nmax, n, icon;
    double epsa, epsr, a, b, s, err;
    ...
    /* call C library routine */
    ierr = c_daqn9(a, b, func, epsa, epsr, nmin, nmax,
                  &s, &err, &n, &icon);
    ...
}
```

返すべき値が 2 次元配列である場合、そのための引数は倍精度へのポインタとして宣言する。この場合の結果の格納方法は次の例のようになる。

```
#include <stdlib.h>
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

#define N 2 /* order of system */

/* user function prototypes */
void fun(double x, double y[], double yp[]);
void jac(double x, double y[], double *pd, int k);

MAIN__()
{
    int ierr, icon;
    int i, n, isw, mf, ivw[N+25];
    double x, y[N], xend, epsv[N], epsr, h, vw[N*(N+17)+70];

    /* Define the input to the routine */
    ...
    /* solve system */
    ierr = c_dodge(&x, y, fun, n, xend, &isw, epsv, &epsr,
                  mf, &h, jac, vw, ivw, &icon);
    /* Check for errors, print results etc. */
    ...
}

/* user function */
void fun(double x, double y[], double yp[])
{
    yp[0] = y[1];
    yp[1] = -11*y[1]-10*y[0];
}
```

```
    return;
}

/* user Jacobian function */
void jac(double x, double y[], double *pd, int k)
{ /* [i][j] -> [i*k+j] */
    pd[0*k+0] = 0; /* [0][0] */
    pd[0*k+1] = 1; /* [0][1] */
    pd[1*k+0] = -10; /* [1][0] */
    pd[1*k+1] = -11; /* [1][1] */
    return;
}
```

1.2.8 複素数

ANSI C には複素数型はサポートされていない. C-SSL II では複素数型を構造体を使って表わし, 次のように typedef を使って dcomplex という名前の複素数型を宣言している.

```
typedef struct {
    double re, im;
} dcomplex;
```

上記の宣言は標準ヘッダファイル cssl.h の中で宣言してあるので, このファイルをソースプログラムの先頭でインクルードしておけば, dcomplex 型を使用することができる. dcomplex は倍精度実数型を実部(re)と虚部(im)に持つ.

複素数を利用したプログラムの例を示す.

```
/* include C-SSL II header file */
#include "cssl.h"
#define NMAX 1000

MAIN__()
{
    dcomplex za[NMAX][NMAX];
    dcomplex zvw[NMAX];
    ...
    /* initialize matrix from file */
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            fscanf(in, "%le, %le", &za[i][j].re, &za[i][j].im);
    ...
    ierr = c_dclu((dcomplex *)za, k, n, epsz, ip, &is, zvw, &icon);
    ...
}
```

1.2.9 コンディションコード

C-SSL II のルーチンには, 実行後の状態を示す引数 icon が用意されている. icon には実行後のコンディションコードがセットされているので, その値を判定した後に出力結果を用いる必要がある.

コードは 0 から 39999 の値をとるが, 結果が保障されるか否かに応じて, 表 1.1 のように区分されている.

表 1.1 コンディションコード

コード	意味	結果の保障	区分
0	正常に処理が終了している.	結果は保障される.	正常
1～9999	正常に処理が終了しているが, なんらかの補助的な情報を含んでいる.		
10000～19999	処理の過程で, 内部的な制限を加えることにより, 一応の処理は終了している.	制限付きで結果は保障される.	警告
20000～29999	処理の過程で異常が生じ, 処理が打ち切られた.	結果は保障されない.	異常
30000～39999	入力引数にエラーがあったため, 処理が打ち切られた.		

1.2.10 プログラムの例

次のプログラムは `c_dvlax` を使って LU 分解により連立一次方程式を解くプログラムである. また, 行列-ベクトル積のルーチン `c_dmav` も使用している.

配列 `a` は実際に解きたい行列よりも大きめに宣言している. そうすることにより, ユーザは C-SSL II のルーチンを次数の異なる幾つかの問題を同じ配列 `a` を使って解くことができる.

また, 近年のアーキテクチャ, 特にベクトル-スーパーコンピュータでは整合寸法 COLS をキャッシュやバンクの衝突が起こらないよう適切な値を選ぶことにより, より高いパフォーマンスで計算を行うことができる. ベクトル計算機では奇数が良いとされている. また, 一般に 2 のべきの値は避けた方がよい.

```

#include <stdio.h>
#include "cssl.h"          ← C-SSL II 用ヘッダファイル

#define ROWS 100           ← 定数は#define で定義するとよい
#define COLS 101

MAIN__()                  ← メインプログラムの関数名は大文字の MAIN に続けて 2 つ
{                          のアンダースコア
    int ierr, icon;
    int n, i, j, isw, is;
    double epsz, eps;
    double a[ROWS][COLS], b[ROWS], x[ROWS], vw[ROWS];
    int ip[ROWS];

    n = 50;
    /* Initialize matrix a */
    ...

    /* Initialize solution vector x */
    ...

    /* Initialize constant vector b = a*x */
    ierr = c_dmav((double*)a, COLS, n, n, x, b, &icon);
    ↑                               整合寸法を引数に与える

    epsz = 0.0;
    isw = 1;

    /* solve system of equations */
    ierr = c_dvlax((double*)a, COLS, n, b, epsz, isw, &is, vw, ip, &icon);
    ↑                               型変換が必要!

    if (icon != 0) {        ← icon で実行後の状態を確認する
        printf("ERROR: c_dvlax failed with icon = %d\n", icon);
        exit(1);
    }
}

```

```
    }  
    /* check solution vector */  
    ...  
}
```

1.3 データの格納方法

本節では, データとして行列を扱う場合の格納方法について述べる.

C 言語の配列に行列を格納する場合の方法は, その構造と形式に応じて異なる. C-SSL II が使用する行列は図 1.3 のようなクラスに分類することができる.

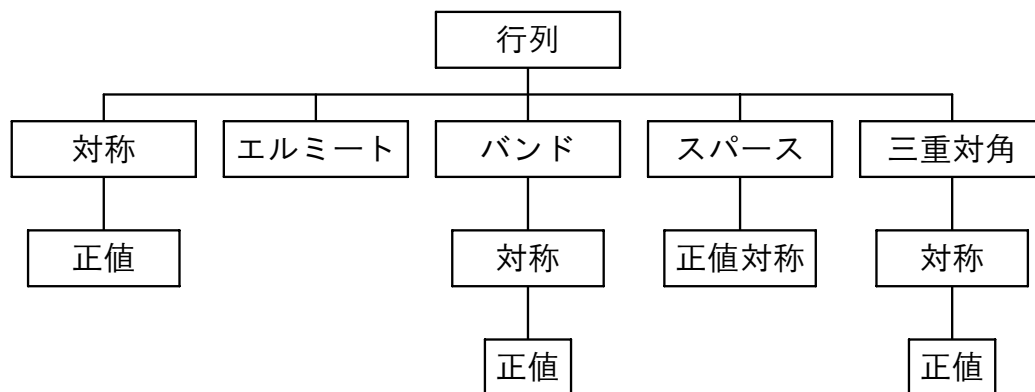


図 1.3 行列の分類

各クラスの中には 1 種類又は複数種類の行列の格納方法が用意されている. 1~2 つのルーチンでしか使用しない格納方法もあるが, それらはベクトル計算機でより良い性能になるよう, 用意されているものである.

1.3.1 一般行列の格納方法

一般の密行列は図 1.4 に示すように 2 次元配列に格納する.

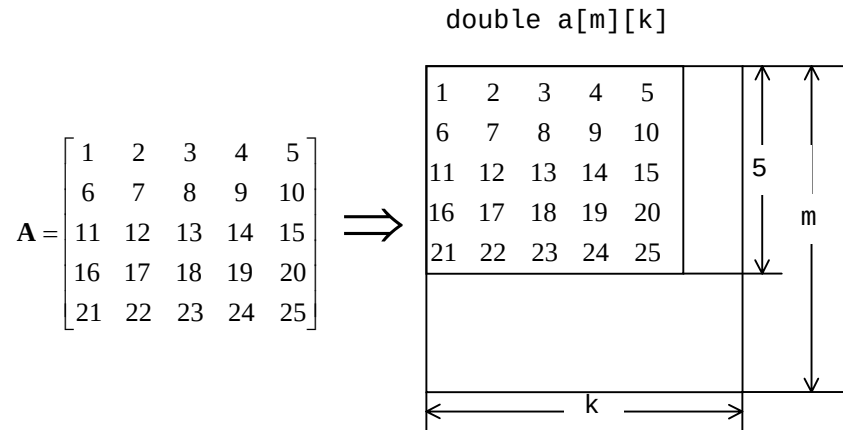


図 1.4 一般行列の格納方法

行列の要素 a_{ij} は配列の要素 $a[i-1][j-1]$ に格納する. 行列の要素の添字は 1 から始まるが, C 言語の配列の添字は 0 から始まることに注意すること. このことは, ベクトルを 1 次元配列に格納する場合も同様で, ベクトルの要素 y_i は配列の要素 $y[i-1]$ に格納する.

C-SSL II のルーチンでは, 問題のサイズよりも大きめに宣言した 2 次元配列を使用することができる. 例として $a[m][k]$ なる 2 次元配列に 5×5 の行列 A を格納する場合を考える. このような行列を引数に持つ関数を使用する場合, 配列 a , 次数(この例では 5)の他に, 整合寸法 k を入力するようになっている. C-SSL II で用いるところの“整合寸法”とは, 2 次元配列 a の列の数 k のことを意味する.

次数の異なる幾つかの行列を扱う場合には, そのうちの最も大きな次数を $NMAX$ とすれば, 利用者プログラムでは $a[NMAX][NMAX]$ なる 2 次元配列を 1 つ用意しておけば, その領域を逐次利用することができる. このときは, 整合寸法として常に, $NMAX$ の値を与えなければならない.

1.3.2 対称行列の格納方法

対称行列用圧縮格納法

図 1.5 のように対称密行列 A の対角部分と下三角部分の要素を, 1 次元配列 a 上に行ごとに格納することを対称行列用圧縮格納法という. nt の値は n 次の場合 $nt = n(n+1)/2$.

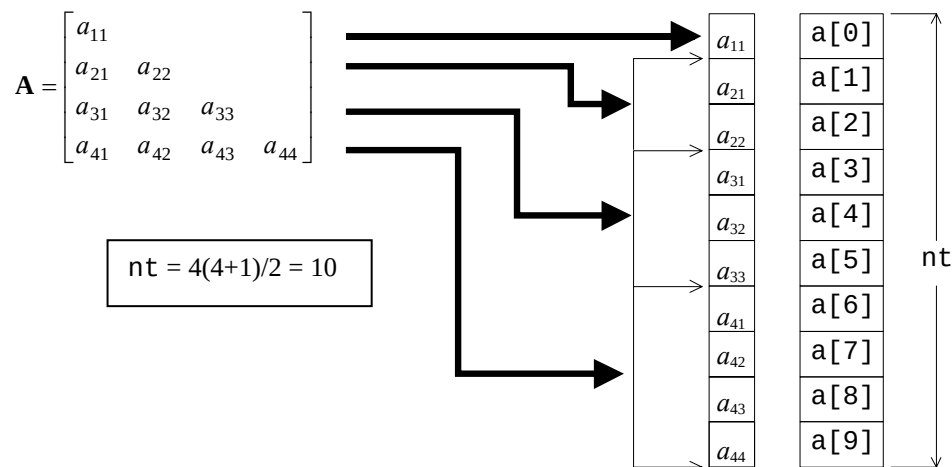


図 1.5 対称行列用圧縮格納法

正値対称行列用圧縮格納法

図 1.6 のように正値対称行列 $\mathbf{A}$ の対角部分と下三角部分を, 1 次元配列 $\mathbf{a}$ 上に列ごとに格納することを正値対称行列用圧縮格納法という. $\mathbf{nt}$ の値は n 次の場合 $\mathbf{nt} = n(n+1)/2$.

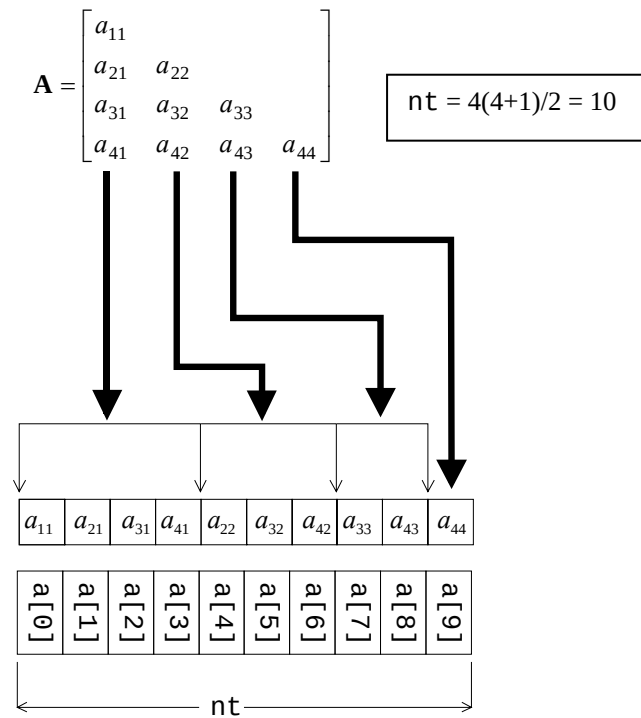


図 1.6 正値対称行列用圧縮格納法

1.3.3 エルミート行列の格納方法

エルミート行列用圧縮格納法

図 1.7 のように, エルミート行列 $\mathbf{A}$ の対角要素と下三角要素の実部を 2 次元配列の対角部分と下三角部分に格納し, 下三角要素の虚部を 2 次元配列の上三角部分に格納することをエルミート行列用圧縮格納法という.

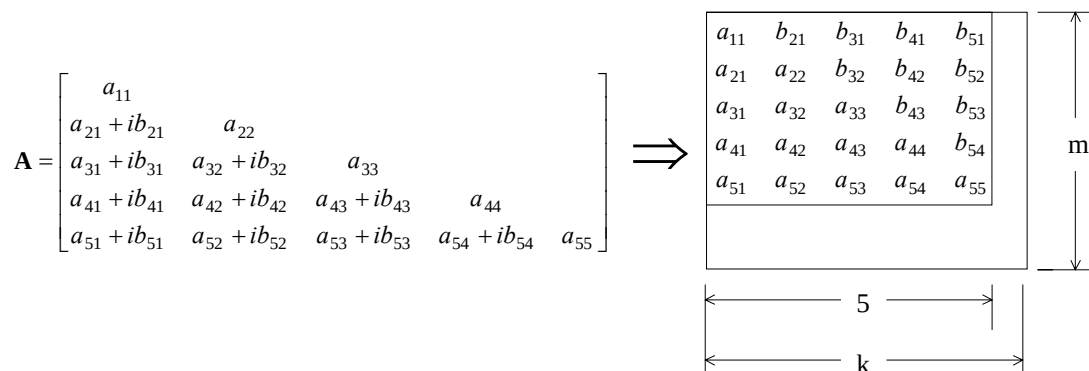


図 1.7 エルミート行列用圧縮格納法

1.3.4 バンド行列の格納方法

バンド行列とは対角部分及び上, 下バンド部分を除きすべて零要素であるような行列である. 以降の説明では 下バンド幅 h_1 , 上バンド幅 h_2 となっている. バンド行列のバンド幅は $w = \min(h_1 + h_2 + 1, n)$ である. 対称バンド行列は, $h = h_1 = h_2$ であるから, $w = 2 \cdot h + 1$ である.

バンド行列用圧縮格納法

図 1.8 のように, バンド行列 $\mathbf{A}$ の要素を 1 次元配列 $\mathbf{a}$ 上に行ごとに格納する方法をバンド行列圧縮格納法という.

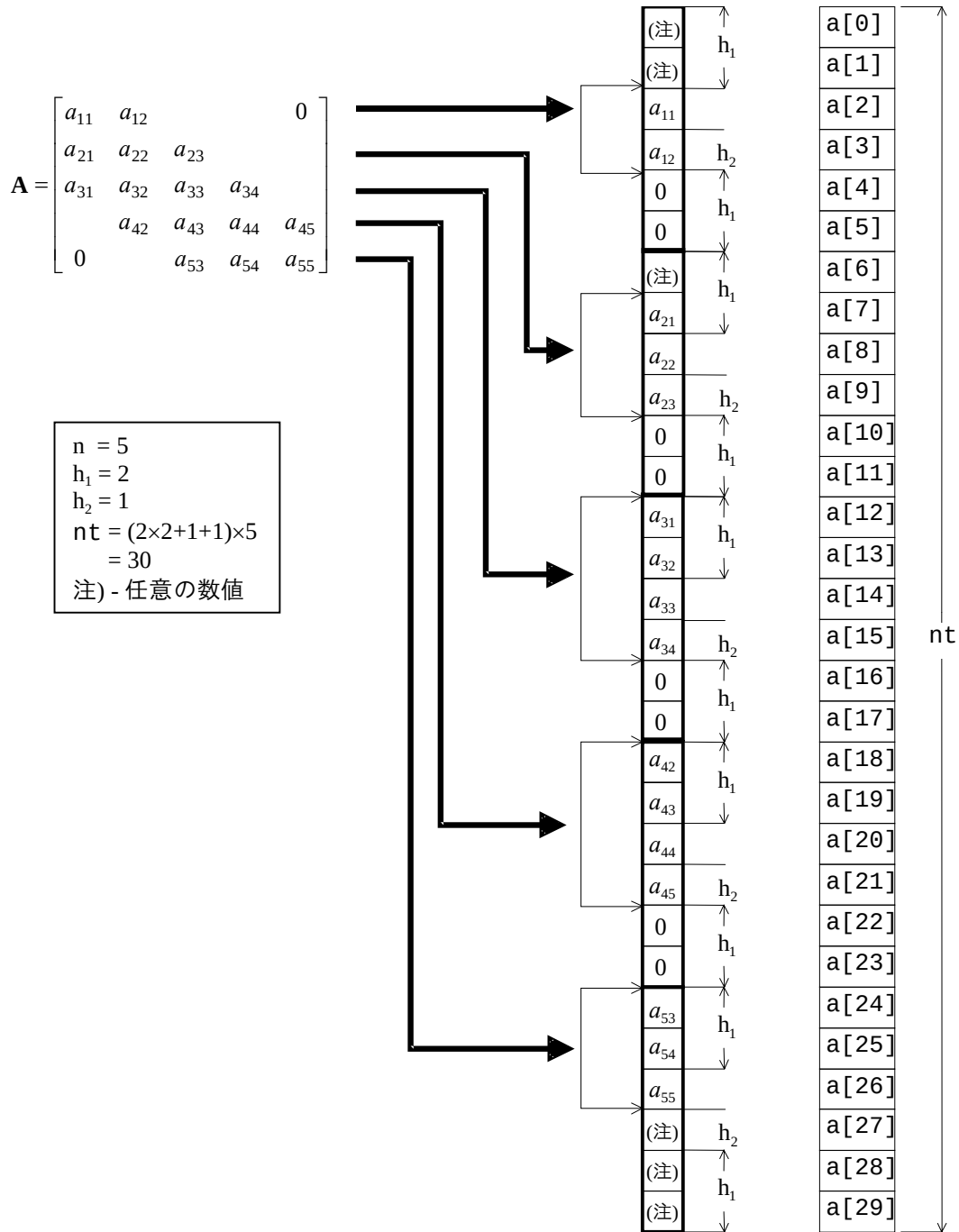


図 1.8 バンド行列用圧縮格納法

この例では下バンド幅 h_1 は 2 であり, 上バンド幅 h_2 は 1 である. 行列の各行の要素は $\mathbf{a}$ の $2 \cdot h_1 + h_2 + 1$ の長さの領域に格納する. 各行の要素の後ろの長さ h_1 の領域は, LU 分解などの処理の過程で使用される領域で

あり, ルーチンを呼び出す前に 0 で初期化しておく必要がある. $2 \cdot h_1 + h_2 + 1$ が行列 $\mathbf{A}$ のサイズ n よりも大きい場合は, バンド行列を使ったルーチンよりも, c_dv1ax などの一般行列を使ったルーチンを使用した方が領域の節約になる.

対称バンド行列用圧縮格納法

図 1.9 のように対称バンド行列 $\mathbf{A}$ の対角部分と下バンド部分の要素を 1 次元配列 $\mathbf{a}$ 上に行ごとに格納することを対称バンド行列圧縮格納法という. 対角要素と幅 h のバンド部分のみを格納するために必要な 1 次元配列のサイズは $nt = n(h+1) - h(h+1)/2$ である.

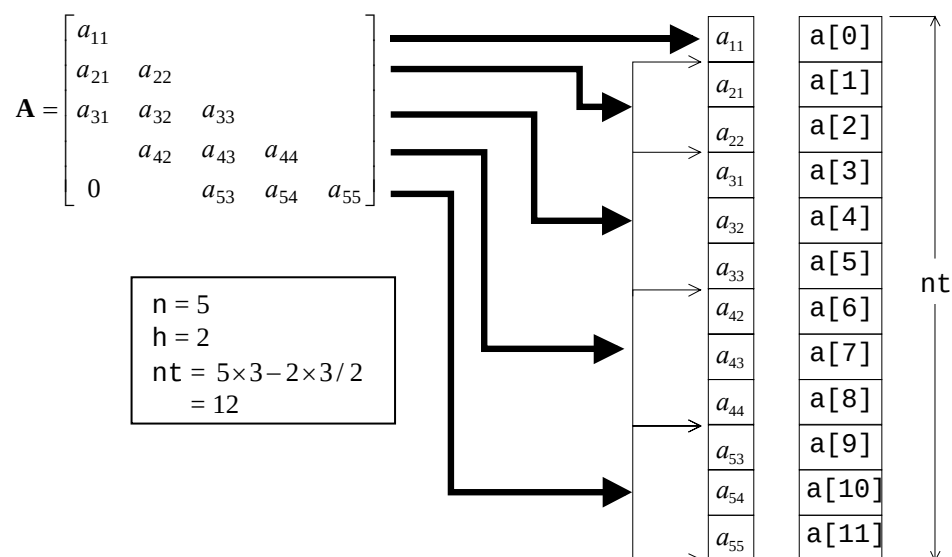


図 1.9 対称バンド行列用圧縮格納法

正値対称バンド行列用圧縮格納法

図 1.10 のように正値対称バンド行列 $\mathbf{A}$ の対角部分と下バンド部分の要素を 1 次元配列 $\mathbf{a}$ 上に行ごとに格納することを正値対称バンド行列用圧縮格納法という. 要素を格納するために必要な 1 次元配列のサイズは $n \times (h+1)$ である.

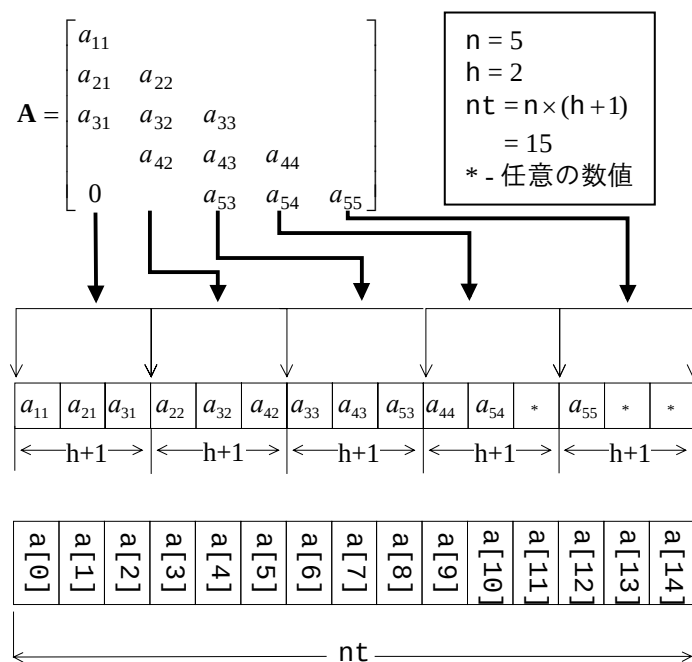


図 1.10 正値対称バンド行列用圧縮格納法

1.3.5 一般スパース行列の格納方法

スパース行列を格納するための方法として、ELLPACK 形式格納法と対角形式格納法を用意している。

ELLPACK 形式格納法は、係数行列の各行ベクトルの非ゼロ要素のみを圧縮して格納する。対角形式格納法は、係数行列を対角方向のベクトルに分解し、非零要素のある対角方向ベクトルのみを格納する。

スパース行列の非零要素が、特定の対角方向のベクトルに集中して存在する構造を持つときは、対角形式の格納法が適している。

ELLPACK 形式格納法

図 1.11 のようにスパース行列を 2 つの 2 次元配列 `coef` 及び `icol` を使用して格納する方法を ELLPACK 形式格納法(詳細については [63], [90]参照)という。

$$\begin{array}{c}
 \mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 3 & 4 & 0 \\ 0 & 0 & 5 & 0 \\ 6 & 0 & 0 & 0 \end{bmatrix} \Rightarrow \begin{array}{l} \mathbf{coef} = \begin{bmatrix} 1 & 3 & 5 & 6 \\ 2 & 4 & 0 & 0 \end{bmatrix} \\ \mathbf{icol} = \begin{bmatrix} 1 & 2 & 3 & 1 \\ 4 & 3 & 3 & 4 \end{bmatrix} \\ \mathbf{iwidth} = 2 \end{array}
 \end{array}$$

図 1.11 一般行列の ELLPACK 形式格納法

係数行列 $\mathbf{A}$ の i 番目の行ベクトルの非零要素を圧縮して `coef` の $i-1$ 列に格納する。行ベクトルを列方向に格納することに注意すること。さらに、`coef` に格納された非零要素が行列 $\mathbf{A}$ の何番目の列にあるかを示す値を、対応する `icol` の配列要素に格納する。`coef` に行列 $\mathbf{A}$ の行ベクトルの非零要素を格納するとき、必ずしも上に詰める必要はない。

この他に、入力の引数として行列 $\mathbf{A}$ の行ベクトルの非零要素の最大値 `iwidth` が必要である。 $\mathbf{A}$ の第 i 行の非零要素の数が `iwidth` より少ないときは、`coef` の余った要素には零を入れ、対応する `icol` の要素には i の値を入れる。

図 1.11 の例では、行列 $\mathbf{A}$ の 1 行目には 1 列目と 4 列目に非零要素がある。 $a_{11}=1$ より、`coef[0][0]` に 1 を格納し、`icol[0][0]` には 1 を格納する。 $a_{14}=2$ より、`coef[1][0]` に 2 を格納し、`icol[1][0]` には 4 を格納する。行列 $\mathbf{A}$ の 3 行目の非零要素の数は `iwidth` よりも少ない。このため `coef[0][2]=5`、`icol[0][2]=3` とするが、`coef[1][2]` には 0 を格納し、`icol[1][2]` には行番号の 3 を格納する。

対角形式格納法

図 1.12 のように、スパース行列を 2 次元配列 `diag` 及び 1 次元配列 `nofst` を使用して格納する方法を対角形式格納法(詳細については [68], [78]参照)という。

$$\begin{array}{c}
 \mathbf{A} = \begin{bmatrix} 0 & 1 & 2 & 3 & 0 \\ 4 & 5 & 0 & 6 & 0 \\ 0 & 7 & 8 & 9 & 0 \\ 0 & 0 & 10 & 11 & 0 \end{bmatrix} \Rightarrow \begin{array}{l} \mathbf{diag} = \begin{bmatrix} 1 & 5 & 8 & 11 \\ 2 & 0 & 9 & 0 \\ 3 & 6 & 0 & 0 \\ 0 & 4 & 7 & 10 \end{bmatrix} \\ \mathbf{nofst} = (0 \ 1 \ 2 \ -1) \end{array}
 \end{array}$$

図 1.12 一般行列の対角形式格納法

行列 $\mathbf{A}$ の非零要素の存在する対角方向のベクトルを配列 `diag` の行方向に格納する. ある整数 k に対して, 次のような対角方向のベクトルを対角ベクトルと, ここでは呼ぶ. 特に, 対角要素からなるベクトルを主対角ベクトルと呼ぶ.

$$(a_{1,1+k}, a_{2,2+k}, \dots, a_{n,n+k}) \quad \text{ただし } i+k < 1 \text{ 又は } i+k > n \text{ のとき } a_{i,i+k} = 0$$

対角ベクトルを配列 `diag` の行ベクトルに格納する順序については自由である. ただし, 次に述べる 一次元配列 `nofst` で対応をとる.

`nofst[i]` には, `diag[i][j]`, $j=0, \dots, n-1$ に格納される対角ベクトルの主対角ベクトルからの距離を格納する. 上記対角ベクトルで k が距離を表す. 主対角要素からなる対角ベクトルは距離が 0, 上三角行列にある対角ベクトルの距離は正の整数, そして下三角行列にある対角ベクトルの距離は負の整数となる.

1.3.6 正値対称スパース行列の格納方法

正値対称スパース行列

正規化された正値対称行列の上三角行列部分と下三角行列部分を `ELLPACK` 形式の格納方法, 又は対角形式の格納方法で, この順番に格納する.

正値対称な行列 $\mathbf{A}$ はその対角要素の平方根の逆数を対角要素に持つ対角行列 $\mathbf{D}^{-1/2}$ によって, 対角要素が 1 の対称行列 $\mathbf{A}^*$ に正規化することができる.

$$\mathbf{A}^* = \mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2}$$

ここで,

$$\begin{aligned} \mathbf{D}^{-1/2} &= \text{diag}(a_{11}^{-1/2}, a_{22}^{-1/2}, \dots, a_{nn}^{-1/2}) \\ &= \text{diag}(d_{11}^{-1/2}, d_{22}^{-1/2}, \dots, d_{nn}^{-1/2}) \end{aligned}$$

次数 n の連立 1 次方程式 $\mathbf{A}\mathbf{x} = \mathbf{b}$ は, 正規化された行列 $\mathbf{A}^*$ の連立 1 次方程式に変換することができる.

$$\begin{aligned} (\mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2})(\mathbf{D}^{1/2} \mathbf{x}) &= \mathbf{D}^{-1/2} \mathbf{b} \\ \mathbf{A}^* \mathbf{y} &= \mathbf{b}^* \end{aligned}$$

ただし, $\mathbf{y} = \mathbf{D}^{1/2} \mathbf{x}$, $\mathbf{b}^* = \mathbf{D}^{-1/2} \mathbf{b}$.

正値対称スパース行列の `ELLPACK` 形式格納法

図 1.13 のように, 対角要素が 1 に正規化された正値対称スパース行列の上三角行列部分及び下三角行列部分を, おのこの一般スパース行列の `ELLPACK` 形式の格納方法で格納したものを 1 つの配列 `coef` にまとめて格納する.

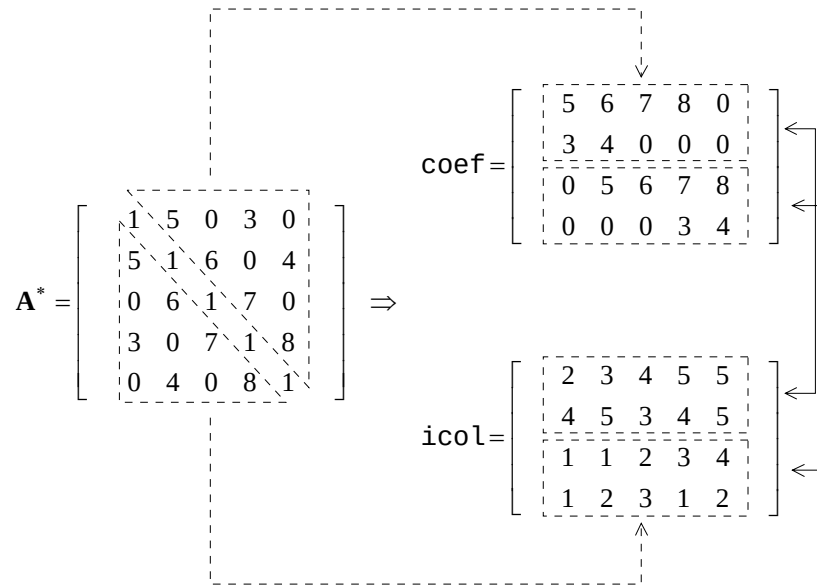


図 1.13 正値対称スパース行列の ELLPACK 形式格納法

上三角行列部分を最初に格納しその後下三角行列部分を格納する。対角要素はすべて1であることがわかっているため格納する必要はない。

上三角行列部分の各行ベクトルでの非零要素の最大値を nsu とし、さらに下三角行列部分の各行ベクトルでの非零要素の最大値を nsL とする。 $nsh = \max(nsu, nsL)$ としたとき上三角行列部分の非零要素を $coef$ の $0 \sim nsh-1$ 行に格納する。下三角行列部分の非零要素は $nsh \sim 2*nsh-1$ 行に格納する。配列 $coef$ の余った要素には零を設定し、対応する $icol$ の配列要素には、何番目の行ベクトルかを示す値を設定する。

図 1.13 の例では、 $a_{32} = 6$ より $coef[2][2]$ に 6 を格納し $icol[2][2]$ に 2 を格納する。同様に、 $a_{34} = 7$ より $coef[0][2]$ に 7 を格納し $icol[0][2]$ に 4 を格納する。

正値対称スパース行列の対角形式格納法

図 1.14 のように、対角要素が 1 に正規化された正値対称スパース行列の上三角行列部分及び下三角行列部分を各々一般スパース行列の対角形式の格納法で格納したもの、1つの配列 $diag$ にまとめて格納する。

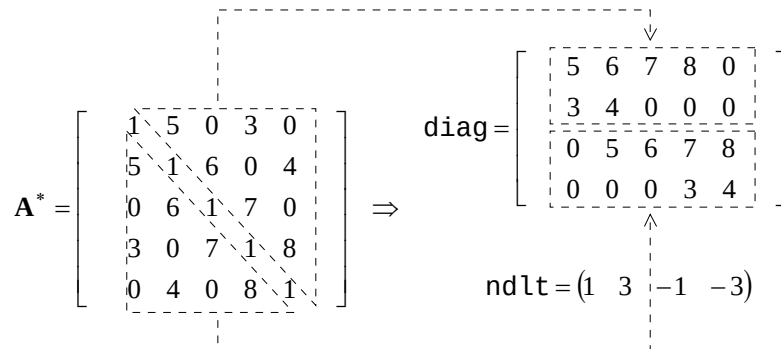


図 1.14 正値対称スパース行列の対角形式の格納法

上三角行列部分(下三角行列部分)の非零要素のある対角ベクトルの本数を ndt とすると、 $diag$ の $0 \sim ndt-1$ 行に上三角行列部分を、 $diag$ の $ndt \sim 2*ndt-1$ 行に下三角行列部分を格納する。このとき、上三角行列部分は距離 ($ndlt$ の値) に関して昇順に、下三角行列部分に関しては、降順に格納しなければならない。

1.4 丸め誤差の単位

C-SSL II においては, 丸め誤差の単位(unit round off)を各所で用いている. 丸め誤差の単位は, 浮動小数点演算の誤差解析における基本的な概念である. 丸め誤差の単位 μ は, 浮動小数点演算において $1+\mu=1$ を満たすような最も大きな値として定義される. C-SSL II においては, この丸め誤差の単位を収束判定, 零判定などに用いている.

浮動小数点演算の誤差解析については, [16], [117], [122]を参照のこと.

1.5 計算機定数

本マニュアルでは, ハードウェアによって決まる計算機定数については, 記号を用いて表現している. 以下に, その記号を定義する.

- fl_{min} — 浮動小数点数の正の最小値. IEEE 形式の浮動小数点数をサポートしているハードウェアの倍精度では約 2.2×10^{-308} .
- fl_{max} — 浮動小数点数の正の最大値. IEEE 形式の浮動小数点数をサポートしているハードウェアの倍精度では約 1.8×10^{308} .
- t_{max} — 三角関数(sin, cos)の引数の上限値. VPP シリーズでは 3.53×10^{15} .

1.6 関数の使用方法の読み方

“第 2 章 関数の使用方法”は次のページような形式で説明している.

c_dmv ← 関数名

実行列と実ベクトルの積

`ierr = c_dmv(a, k, m, n, x, y, &icon);`機能のタイトルと
呼び出し例**1. 機能** ← 機能の詳細な説明 $m \times n$ の実行列 **A** と実ベクトル **x** の積 **y** を求める.

$$\mathbf{y} = \mathbf{Ax} \quad (1)$$

ここで **x** は n 次元ベクトル, **y** は m 次元ベクトルである. $m, n \geq 1$ であること.**2. 引数** ← 呼び出し例と引数の説明

呼び出し形式:

`ierr = c_dmv((double*)a, k, m, n, x, y, &icon);` ← 呼び出し例(型変換を含む)

引数の説明:

引数	型宣言	種類	詳細な説明
a	double a[m][k]	入力	行列 A .
k	int	入力	配列 a の整合寸法 ($\geq n$).
m	int	入力	行列 A の行数 m .
n	int	入力	行列 A の列数 n (“使用上の注意”参照).
x	double x[n]	入力	ベクトル x .
y	double y[m]	入力	ベクトル y . “使用上の注意” (2) の計算を行うときにのみ使用する.
		出力	行列 A とベクトル x の積 y .
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード: ← 値は関数ごとに異なる

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	次のいずれかであった. • $m < 1$ • $n = 0$ • $k < n$	処理を打ち切る.

1. 使用上の注意 ← 関数の使用方法に関する補足説明

a) 残差ベクトルの計算

本関数は、通常(1)の計算を主とするが、 $n = -n$ として任意のベクトル $\mathbf{y}'$ を引数 $\mathbf{y}$ に与えると、

$$\mathbf{y} = \mathbf{y}' - \mathbf{A}\mathbf{x} \quad (2)$$

なる計算ができる。

具体的には、連立1次方程式の残差ベクトル、

$$\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}$$

の計算等を行う場合に利用できる。

2. 使用例 ← 簡単なサンプルプログラム

行列とベクトルの積を計算する。行列は 100×100 、ベクトルは100個の要素を持つ。

```
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 100

MAIN__()
{
    int ierr, icon;
    int m, n, i, j, k;
    double eps;
    double a[NMAX][NMAX], x[NMAX], y[NMAX];

    /* initialize matrix and vector */
    m = NMAX;
    n = NMAX;
    k = NMAX;
    for (i=0; i<n; i++) {
        for (j=0; j<n; j++)
            a[i][j] = 1.0/(j+1);
        x[i] = i+1;
    }
    /* perform matrix vector multiply */
    ierr = c_dmav((double*)a, k, m, n, x, y, &icon);
    if (icon != 0) {
        printf("ERROR: c_dmav failed with icon = %d\n", icon);
        exit(1);
    }
    /* check vector */
    eps = 1e-6;
    for (i=0; i<n; i++)
        if (fabs((y[i]-n)/n) > eps) {
            printf("WARNING: result inaccurate\n");
            exit(1);
        }
    printf("Result OK\n");
    exit(0);
}
```

3. 手法概要 ← 本マニュアルでは詳細な説明は省略し、Fortran SSL II マニュアルを参照している。

本関数は標準的なアルゴリズムを用いている。

この方法の原理及び、本関数の手順については“富士通 SSL II 使用手引書”のMAVの項目の手法概要を参照されたい。

1.7 線型計算

1.7.1 概要

線型計算では, 行列の構造と問題などに従って, 表 1.2 のように体系化されている.

表 1.2 線型計算の体系

構造	問題	説明箇所
密行列	行列格納方法の変換	1.7.2
	行列操作	1.7.3
	連立 1 次方程式・逆行列	1.7.4
	最小二乗解	1.7.7
バンド行列	行列格納方法の変換	1.7.2
	行列操作	1.7.3
	連立 1 次方程式	1.7.4
3 項行列	連立 1 次方程式	1.7.5
スパース行列	行列操作	1.7.3
	連立 1 次方程式 (反復法)	1.7.6

行列の格納方法は, その構造と形式に応じて異なる. 例えば, 実対称行列では下三角部分並びに対角部分だけ格納するようになっている. したがって, 連立 1 次方程式を解く場合でも, 固有の格納方法に対応して, ルーチンが用意されている. 適した格納方法を選択することによりメモリや計算時間を節約することができる.

1.7.2 行列格納方法の変換

行列格納方法変換では, 図 1.15 に示すとおり格納方法を変換する.

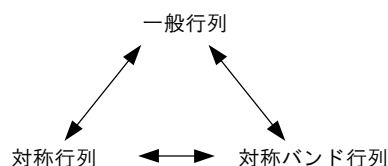


図 1.15 行列格納方法の変換

C-SSL II では, 行列をその構造と形式に応じて異なる格納方法で格納する. ある格納方法で行列が与えられているとき, 異なった格納方法に対応した関数を用いる場合は, 格納方法をあらかじめ変換する必要がある. このために表 1.3 に示す変換用ルーチンが用意されている.

表 1.3 モード変換の関数

変換前	変換後		
	一般行列の 格納方法	対称行列用 圧縮格納法	対称バンド行列 用圧縮格納法
一般行列の 格納方法		c_dcgsm	c_dgsbm
対称行列用 圧縮格納法	c_dcsgm		c_dcssbm
対称バンド行列 用圧縮格納法	c_dcsbgm	c_dcsbsm	

1.7.3 行列操作

行列操作では、次の基本演算を扱う。

- 行列と行列の和・差 $\mathbf{A} \pm \mathbf{B}$
- 行列とベクトルの積 $\mathbf{Ax}$
- 行列と行列の積 $\mathbf{AB}$

C-SSL II では、行列操作のために表 1. 4に示すルーチンが用意されている。スパース行列に関しては、ELLPACK 形式格納法と対角形式格納法の 2 種類の格納方法を利用した行列操作関数がある。

表 1.4 行列操作の関数

A	演算	B 又は x		
		実行例	対称行列	ベクトル
実行例	和	c_daggm		
	差	c_dsggm		
	積	c_dvmggm	c_dmgsbm	c_dmav
複素行列	積			c_dmcv
対称行列	和		c_dassm	
	差		c_dsssm	
	積	c_dmsgm	c_dmssm	c_dmsv
バンド行列	積			c_dvmbv
対称バンド行列	積			c_dmsbv
スパース行列 – 対角形式格納法	積			c_dvmvsd
スパース行列 – ELLPACK 形式格納法	積			c_dvmvse

a) 使用上の注意

行列(スパース行列を除く)とベクトルの積を行う関数は連立 1 次方程式において、近似解に対する残差ベクトル $\mathbf{r} = \mathbf{r} - \mathbf{Ax}$ を求める場合を意図して用意されている。すなわち、近似解の反復改良や、方程式の解を反復解法により求める場合に利用することができる。

1.7.4 連立 1 次方程式・逆行列 (直接法)

ここでは、次の問題を扱う。

- 連立 1 次方程式 $\mathbf{Ax} = \mathbf{b}$ を解く. $\mathbf{A}$ は $n \times n$ の行列. $\mathbf{x}, \mathbf{b}$ は n 次元ベクトル.
- 行列 $\mathbf{A}$ の逆行列 $\mathbf{A}^{-1}$ を求める.
- 行列 $\mathbf{A}$ の行列式 $\det(\mathbf{A})$ を求める.

C-SSL II には、これらの問題を解くために、次のような基本機能を持つ関数が、行列の区分に応じて用意されている(コンポーネントルーチン).

- 係数行列の分解 (numeric decomposition)
- 分解された行列に基づく求解 (solve)
- 分解された行列に基づく行列の逆転

これらを組み合わせて用いれば、連立 1 次方程式を解くこと、逆行列を求めること及び、行列式を求めることができる。

連立 1 次方程式

方程式の解は、次のように順次、コンポーネントルーチン呼び出せば求めることができる。

```
...
/* Decomposition routine */
ierr = c_dvalu((double *)a,k,n,epsz,ip,&is,vw,&icon)
/* Solve routine given a decomposition */
ierr = c_dlux(b,(double *)a,k,n,isw,ip,&icon)
...
```

逆行列

逆行列は、次のように順次、コンポーネントルーチン呼び出せば求めることができる。

```
...
/* Decomposition routine */
ierr = c_dvalu((double *)a,k,n,epsz,ip,&is,vw,&icon);
/* Compute matrix inverse given a decomposition */
ierr = c_dvluiv((double *)a,k,n,ip,(double *)ai,&icon);
...
```

ただし、バンド行列の逆行列は一般には密行列となり、逆行列を求めることは計算上得策ではない。したがって、C-SSL II には、バンド行列の逆行列を求めるコンポーネントルーチンは用意されていない。

行列式

行列式の値を求めるコンポーネントルーチンは用意されていないが、係数行列の分解のための コンポーネントルーチンにより分解された結果を用いて求めることができる。

このように、問題に応じてコンポーネントルーチンを組み合わせればよいが、C-SSL II ではそれを標準的に使う場合も考慮して結合したルーチン(標準ルーチン)も用意されている。通常は、この標準ルーチンを用いればよい。

a) ルーチン一覧

表 1.5に、行列の種類に応じた関数を示す。

表 1.5 連立 1 次方程式・逆行列の関数

行列の種類	標準ルーチン名	係数行列の分解	求解	行列の逆転
実行例	c_dvlax	c_dvalu	c_dlux	c_dvluiv
複素行列	c_dlcx	c_dclu	c_dclux	c_dcluiv
対称行列	c_dlsix	c_dsmdm	c_dmdmx	
正値対称行列 (変形コレスキー法)	c_dvlsx	c_dvsldl	c_dvldlx	c_dvldiv
正値対称行列 (ブロック化された コレスキー分解法)	c_dvlspx	c_dvspll	c_dvsplx	
バンド行列	c_dvlbx	c_dvblu	c_dvblux	
実対称バンド行列	c_dlsbix	c_dsbmdm	c_dbmdmx	
正値対称バンド行列	c_dvlsbx	c_dvbldl	c_dvbldx	

b) 使用上の注意

逆行列について

連立 1 次方程式を解くために、逆行列を求めて計算することは、一般的には得策ではない。すなわち、連立 1 次方程式(1)において、

$$\mathbf{Ax} = \mathbf{b} \quad (1)$$

まず、逆行列 $\mathbf{A}^{-1}$ を求め、次に左側から $\mathbf{b}$ に掛ける (2) なる操作により解を求めることは避けるべきである。

$$\mathbf{x} = \mathbf{A}^{-1} \mathbf{b} \quad (2)$$

通常は、 $\mathbf{A}$ を LU 分解し、(3) なる操作（前進代入と後退代入）により解を求める。

$$\begin{aligned} \mathbf{Ly} &= \mathbf{b} \\ \mathbf{Ux} &= \mathbf{y} \end{aligned} \quad (3)$$

(3) による方が、演算速度・精度の点ですぐれている。ちなみに (2) と (3) の各々の場合における、計算で必要となる乗算回数の概算はそれぞれ $n^3 + n^2$ 及び $n^3/3$ となる。したがって、逆行列は本質的にそれを必要とするときだけ求めること。

同一の係数行列を持つ方程式について

係数行列 $\mathbf{A}$ が同一で、定数ベクトル $\mathbf{b}$ だけ異なる複数組の連立 1 次方程式 (4) を解く場合、

$$\left. \begin{aligned} \mathbf{Ax}_1 &= \mathbf{b}_1 \\ \mathbf{Ax}_2 &= \mathbf{b}_2 \\ &\vdots \\ \mathbf{Ax}_m &= \mathbf{b}_m \end{aligned} \right\} \quad (4)$$

各々の方程式で行列 $\mathbf{A}$ を分解することは得策ではない。この場合は、最初の方程式を解く時にだけ $\mathbf{A}$ を分解し、それ以降の方程式を解く時には、(3) に準じて前進代入、後退代入だけを行えばよい。

標準ルーチンにおいては、 $\mathbf{A}$ の分解から行うか、 $\mathbf{A}$ の分解は省略するかを制御する引数 `isw` が用意されている。

c) 内部処理上の留意点

ここでは、内部処理の立場から、使用上の留意点について述べる。

ガウス消去法とクラウト法

正則な非対称行列の分解 (5) の方法として、通常ガウス消去法とクラウト法が知られている。C-SSL II において、正則な非対称行列の分解の方法として、外積形式ガウスの消去法をブロック化した方法を採用している。

$$\mathbf{A} = \mathbf{LU} \quad (5)$$

ここで、 $\mathbf{L}$ は下三角行列、 $\mathbf{U}$ は上三角行列である。

コレスキー分解法と変形コレスキー法

正値対称行列に対しては、ブロック化されたコレスキー分解法及び変形コレスキー法を採用している。すなわち、(6)なる分解を行う。

$$\begin{aligned} \mathbf{A} &= \mathbf{LL}^T, \\ \mathbf{A} &= \mathbf{LDL}^T \end{aligned} \quad (6)$$

ここで、 $\mathbf{L}$ は下三角行列、 $\mathbf{D}$ は対角行列である。なお、分解された行列は、表 1.6 のように与えられるので注意されたい。

表 1.6 分解された行列

行列の種類	分解された行列の内容
実行例	$\mathbf{PA} = \mathbf{LU}$ $\mathbf{L}$: 下三角行列 $\mathbf{U}$: 単位上三角行列 $\mathbf{P}$ は置換行列
正値対称行列 (コレスキー分解法)	$\mathbf{A} = \mathbf{LL}^T$ $\mathbf{L}$: 下三角行列 なお、下三角行列は計算の手間を少なくするために $\mathbf{L}^T$ として出力される
正値対称行列 (変形コレスキー法)	$\mathbf{A} = \mathbf{LDL}^T$ $\mathbf{L}$: 単位下三角行列 $\mathbf{D}$: 対角行列 なお、対角行列は計算の手間を少なくするために $\mathbf{D}^{-1}$ として出力される

ピボットリングとスケーリング

正則な実行例の分解(5)を数値的に行うことを考える。例えば(7)の行列の分解を考えると、

$$\mathbf{A} = \begin{bmatrix} 0.0 & 1.0 \\ 2.0 & 0.0 \end{bmatrix} \quad (7)$$

このままの状態では、LU 分解は不可能である。また、(8)の場合には、

$$\mathbf{A} = \begin{bmatrix} 0.0001 & 1.0 \\ 1.0 & 1.0 \end{bmatrix} \quad (8)$$

3桁程度の浮動小数点演算では、まったく不安定である。このような事態は、行列の条件が悪い場合にはしばしば発生することであるが、これは、絶対値最大要素をピボットとして選択するピボットリングを行うことにより、回避できることが知られている。(8)の場合は、1行目と2行目の各要素を交換することにより回避できる。

さて、このピボットリングを行うためには、絶対値最大のものを選択する方法に一意性が必要である。ところで、ある行のすべての要素に大きな定数を掛ければ、どの要素でも(それが零でないなら)他の行のどの要素よりも絶対値を大きくすることができる。したがって、ピボットリングで、絶対値最大の大小でピボットを決めることが有効に働くように、行及び列の均衡化 (equilibration) を図る必要がある。

C-SSL II では、“行の均衡化を考慮した部分ピボットリング”を採用している。なお、行の均衡化は、分解する行列の各行において、絶対値最大が 1 となるようなスケーリングにより行うが、実際には各要素の値を変更することなく、ピボット選択時にスケーリングファクターを考慮することで対処している。

トランスポジションベクトル

C-SSL II では、ピボットリングによる行の入換えを行うので、その履歴をトランスポジションベクトルとして保存する。

部分ピボットリングを伴う行列の分解は、(9)により行列表現される。

$$\mathbf{PA} = \mathbf{LU} \quad (9)$$

ここで $\mathbf{P}$ は、行の入換えを行う置換行列である。C-SSL II では、この置換行列 $\mathbf{P}$ を直接保存することなく、トランスポジションベクトルとしてあつかう。すなわち、分解の j 段階 ($j = 1, \dots, n$) において、第 i 行 ($i \geq j$) をピボット行として選択した場合には、分解過程の行列の i 行と j 行とを入れ換え、トランスポジションベクトルの第 j 要素を i とする。

ピボット要素の相対零判定

分解の過程で、ピボット要素が零となった場合、若しくは零に近い値となった場合は、行列が非正則 (singular) である可能性が大きい。この場合、有限の桁数の演算で零に近いピボットを用いて計算を続行することは、解が不正解になることが予想される。C-SSL II では、このような状態になった場合の処理の続行又は、打ち切りを制御するために、引数 epsz を用意している。すなわち、 $\text{epsz} = 10^{-s}$ を設定すると、ピボットとなるべき要素の計算で、10 進 s 桁以上の桁落ちを起こした場合に、そのピボットを相対的に零とみなす。

1.7.5 連立 1 次方程式 (3 項行列)

3 項行列の連立 1 次方程式を解くためのルーチンを表 1.7 に示す。

表 1.7 3 項行列の連立 1 次方程式

行列の種類	計算方法	ルーチン名
実 3 項行列	QR 分解	c_dvltqr
実 3 項行列	ガウス消去法	c_dltx
正値対称 3 項行列	変形コレスキー法	c_dlstx
既約優対角性を満たす実 3 項行列	サイクリック・ リダクション法	c_dvltx
既約優対角性を満たす定数型実 3 項行列 (ディリクレ型)		c_dvltx1
既約優対角性を満たす定数型実 3 項行列 (ノイマン型)		c_dvltx2
既約優対角性を満たす定数型実 3 項行列 (周期型)		c_dvltx3

各々のルーチンは扱う行列の型が違う。一般の正則な 3 項行列の連立 1 次方程式をベクトル計算機で解く場合は、QR 分解により高速に計算できる c_dvltqr を使う。係数行列が既約優対角性を満たすなら、c_dvltx を使ったほうが処理が速い。c_dltx 及び c_dlstx は小規模な連立 1 次方程式を解く場合や、スカラ計算機で解く場合にむいている。

c_dvltx1, c_dvltx2 及び c_dvltx3 も既約優対角性を満たす行列を扱うが、c_dvltx よりも特殊な型の行列を扱う。c_dvltx1 は(10)の行列を扱う。

$$\mathbf{A} = \begin{bmatrix} d & f & & & \\ e & d & f & & 0 \\ & e & \cdot & \cdot & \\ & & \cdot & \cdot & \cdot \\ 0 & & & \cdot & \cdot & f \\ & & & & e & d \end{bmatrix} \quad (10)$$

(10)のように、要素の値が行と列によらず一定である行列を、本マニュアルでは、定数型 3 項行列と呼ぶ。あるいは、この行列が、ある種のディリクレ境界値問題から派生することから、より厳密に、ディリクレ型の定数型 3 項行列と呼ぶ。c_dvltx2, c_dvltx3 の扱う行列は、(10)を若干、変形したものである。c_dvltx2 では(10)における 1 行 2 列の要素、又は n 行 $n-1$ 列の要素が $2e$ となった行列を扱い、これをノイマン型の定数型 3 項行列と呼ぶ。c_dvltx3 では、1 行 n 列の要素と n 行 1 列の要素に e が加わった行列を扱い、これを周期型の定数型 3 項行列と呼ぶ。いずれも微分方程式の境界値問題から派生する行列である。

既約優対角性を満たす行列の連立 1 次方程式を解くためのアルゴリズムとしては、ベクトル計算機向きのアルゴリズムであるサイクリック・リダクション法(Cyclic Reduction Method)を採用している。これはガウス消去法と比較して計算量自体は多いが、ベクトル計算機にとって重要な、計算の並列性が著しく高い。精度については、既約優対角性を満たす行列ならば、ガウス消去法と同等である。

1.7.6 連立 1 次方程式の反復法と収束性

反復法を用いて、スパース行列の連立 1 次方程式を解くためのルーチンを表 1. 8に示す。これらのルーチンではメモリの使用を抑え、大規模問題を、高速に解くことを可能にしている。

表 1. 8 スパース行列の連立 1 次方程式のルーチン

行列の種類	計算方法	対角形式格納法	ELLPACK 形式格納法
正値対称スパース行列	前処理付き CG 法	c_dvcgd	c_dvcge
非対称又は不定値のスパース行列	転置なし準最小残差法 (TFQMR 法)	c_dvtfqd	c_dvtfqe
	準最小残差法(QMR 法)	c_dvqmrq	c_dvqmre
	修正一般化共役残差法 (MGCR 法)	c_dvcrd	c_dvcre
	BiCGSTAB(l)法	c_dvbcsd	c_dvbcse

1.7.6.1 スケーリング

スパース行列の連立 1 次方程式の反復解法を利用するとき、行列の要素を均衡化するためにスケーリングを行なうことをすすめる。行列の正規化は数値的安定性と反復法の収束率を強力に改善する。

正規化された係数行列 $\hat{\mathbf{A}}$ は対角要素が非負であり、かつ各行の要素の絶対値の和がほぼ 1 であるべきである。

$$\mathbf{Ax} = \mathbf{b} \quad (11)$$

連立 1 次方程式(11)の正規化された形は、係数行列に左から対角行列 $\mathbf{L}$ を、右から対角行列 $\mathbf{R}$ を掛けることで構成することができる。新しい変数 $\hat{\mathbf{x}} = \mathbf{R}^{-1}\mathbf{x}$ を導入して方程式(11)は次のようになる。

$$\mathbf{LAR} \hat{\mathbf{x}} = \mathbf{Lb} \Leftrightarrow \hat{\mathbf{A}} \hat{\mathbf{x}} = \hat{\mathbf{b}}$$

ここで、 $\hat{\mathbf{A}} = \mathbf{LAR}$ 、 $\hat{\mathbf{b}} = \mathbf{Lb}$ である。

$\mathbf{A}$ の代わりに正規化された行列 $\hat{\mathbf{A}}$ を反復法で使うことを奨める。このとき、解法を呼び出す前に右辺の $\mathbf{b}$ に $\mathbf{L}$ を掛けること、および返却された近似解に $\mathbf{R}$ を掛けて変換することが必要である。

$s_i = \sum_{j=1}^n |a_{ij}|$ を i 番目の行の要素の絶対値の和とする.

$$L_{ij} = \begin{cases} \frac{\text{sgn}(a_{ii})}{\sqrt{s_i}} & i = j \\ 0 & i \neq j \end{cases} \quad (12)$$

$$R_{ij} = \begin{cases} \frac{1}{\sqrt{s_i}} & i = j \\ 0 & i \neq j \end{cases} \quad (13)$$

反復法の収束率に影響を与える正規化を行なう他の方法があることに注意すること. [116]を参照されたい.

(12), (13) を選ぶと $\mathbf{A}$ が正値対称であれば, $\hat{\mathbf{A}}$ も正値対称であることに注意すること.

1.7.6.2 行列の対称性と反復解法

a) 対称行列

行列 $\mathbf{A}$ が対称, すなわち $a_{ij} = a_{ji}$ ($i, j = 1, \dots, n$) で正値であれば, 古典的共役勾配法([53]参照)が連立一次方程式を解くために使える. 行列が正値でないとブレイクダウンを起こす.

b) 非対称行列

係数行列が非対称又は不定値の場合の解法も使える. 与えられた連立 1 次方程式に対する最適な解法は, 係数行列 $\mathbf{A}$ の特性 (又は正規化された係数行列 $\hat{\mathbf{A}}$ の特性) によって変わる. 異なったクラスの行列に対して以下の解法が利用できる.

1.7.6.3 行列の固有値分布と反復解法の収束

a) MGCR 法

係数行列の固有値の分布が正の実軸に近いとき(図 1. 16参照), MGCR 法 ([66]参照) で検索方向の数を小さく (5~10) して使える (検索方向の意味はここでは省略). 固有値の虚部が大きい固有値があるとき, 収束を良くするために検索方向の数を増やす必要がある. 検索方向の数を増やすことは, 計算過程をより安定にするはたらかがあるが, 計算ステップ毎の計算量とデータ格納量が増える. MGCR 法では, 与えられた問題に対して, まずは検索方向の数を小さくして試行することを奨める.

b) TFQMR 法

係数行列の固有値の実部が正で複素平面の右半分にあり(図 1. 17参照), かつ虚部が大きい場合には, TFQMR 法 ([36]参照)を奨める. また, 解法は固有値の実部の最小値が出来る限り大きいとき最も収束が良くなる. そのため, 例えば, 実部がゼロでない非常に小さな正数値となる固有値があるとき, 収束は悪くなる. TFQMR 法の収束は, 検索方向数に大きな数を指定した MGCR 法よりも悪いかもしれない. しかし, TFQMR 法の反復の各ステップは大きな検索方向数を指定した MGCR 法よりコストは低い. その結果少ない CPU time で計算できる. つまり TFQMR 法は堅固であるが, 小さな検索方向数を指定した MGCR 法より遅いことになる.

c) BICGSTAB(l)

TFQMR 法に似て, BICGSTAB(l)法 ([102]参照)は固有値の分布が実部が正の複素平面の右半分にあるときに適している. また, 解法は固有値の実部の最小値が出来る限り大きいとき最も収束が良くなる. そのため, 例えば, 実部がゼロでない非常に小さな正数値となる固有値があるとき, 収束は悪くなる. 係数行列の固有値の分布が正値の実軸に近いある種の問題で, BICGSTAB(l)法は, 小さな検索方向数を指定した MGCR 法よりもさらに収束率が速い. しかし, BICGSTAB(l)法の各反復ステップは, 行列ベクトル積を 2 回必要なため非常にコストがかかる. このため, ある種の問題では MGCR 法又は TFQMR 法の方が BICGSTAB(l)法より高速である. しかし, BICGSTAB(l)法はより堅固な方法である.

正規化された係数行列の固有値についての情報が無いときは, MGCR 法, TFQMR 法, BICGSTAB(*l*)法をそれぞれ試すことを提案する. 3つの方法を試す順序については, まず, 一般的に堅固でなくても速い方法を第一に選ぶ. その意味で MGCR 法を検索方向の数を 5~10 にして最初に試すことを奨める. そこでうまくいかなければ他の堅固な方法に進む. ある問題に対してどの方法が最も適合しているかを見るには, 例えば収束判定値 eps を 0.1 程度にして各方法の計算時間を測定してみるのが適当である.

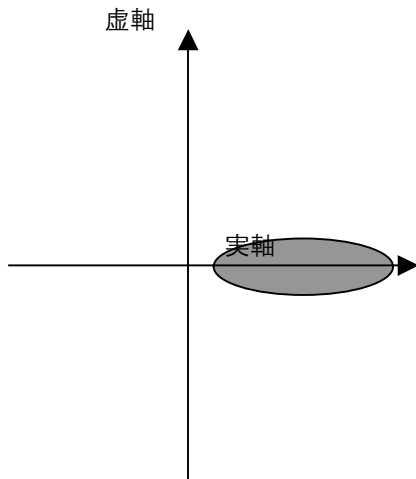
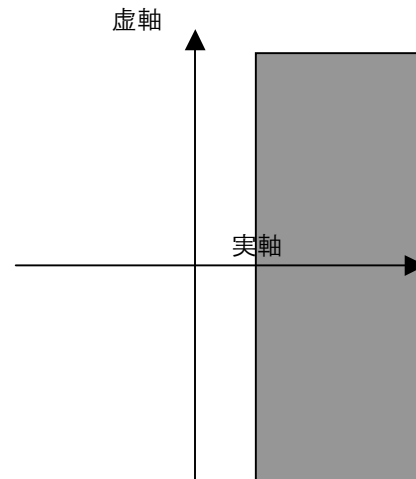


図 1.16 MGCR 法が収束する固有値の分布

図 1.17 TFQMR 法および BICGSTAB(*l*)が収束する固有値の分布

1.7.7 最小二乗解

ここでは, $m \times n$ 行列を扱う次の問題を取りあげる.

- 最小二乗解
- 最小二乗最小ノルム解
- 一般逆行列
- 特異値分解

C-SSL II では, このために, 表 1.9 に示される関数が用意されている.

表 1.9 $m \times n$ 行列を扱う関数

機能	ルーチン名
最小二乗解	c_dlaxl
最小二乗最小ノルム解	c_dlaxlm
一般逆行列	c_dginv
特異値分解	c_dasvd1

最小二乗解

最小二乗解は, 連立 1 次方程式 $\mathbf{Ax} = \mathbf{b}$ において, $\|\mathbf{Ax} - \mathbf{b}\|_2$ が最小となる解 $\tilde{\mathbf{x}}$ である. ここで $\mathbf{A}$ は $m \times n$ 行列 ($m \geq n$, $\text{rank}(\mathbf{A}) = n$), $\mathbf{x}$ は n 次元ベクトル, $\mathbf{b}$ は m 次元ベクトル.

最小二乗最小ノルム解

最小二乗最小ノルム解は, 連立 1 次方程式 $\mathbf{Ax} = \mathbf{b}$ において, $\|\mathbf{Ax} - \mathbf{b}\|_2$ が最小となるベクトル $\mathbf{x}$ の中で, $\|\mathbf{x}\|_2$ が最小となる解 $\mathbf{x}^+$ である. ここで, $\mathbf{A}$ は $m \times n$ 行列, $\mathbf{x}$ は n 次元ベクトル, $\mathbf{b}$ は m 次元ベクトル.

一般逆行列

$m \times n$ 行列 $\mathbf{A}$ に対して, $n \times m$ 行列 $\mathbf{X}$ が (14) を満たすとき, これを $\mathbf{A}$ のムーアペンローズ (Moore-Penrose) の一般逆行列という.

$$\begin{aligned} \mathbf{AXA} &= \mathbf{A} \\ \mathbf{XAX} &= \mathbf{X} \\ (\mathbf{AX})^T &= \mathbf{AX} \\ (\mathbf{XA})^T &= \mathbf{XA} \end{aligned} \quad (14)$$

このような一般逆行列は常に存在し一意である. なお, 行列 $\mathbf{A}$ の一般逆行列 $\mathbf{X}$ を $\mathbf{A}^+$ と表す. C-SSL II では, 以上のような一般逆行列を計算する. C-SSL II で扱う行列は, 縦長行列 ($m > n$), 正方行列 ($m = n$) あるいは横長行列 ($m < n$) のどれであってもよい.

特異値分解

$m \times n$ の実行例 $\mathbf{A}$ が与えられたとき, $\mathbf{A}$ の (15) なる分解を特異値分解という.

$$\mathbf{A} = \mathbf{U}_0 \mathbf{\Sigma}_0 \mathbf{V}^T \quad (15)$$

ここで $\mathbf{U}_0$ 及び $\mathbf{V}$ は, $m \times m$ 及び $n \times n$ の直交行列, $\mathbf{\Sigma}_0$ は $\mathbf{\Sigma}_0 = \text{diag}(\sigma_i)$, $\sigma_i \geq 0$ なる $m \times n$ の対角行列であり, σ_i を $\mathbf{A}$ の特異値という.

いま, $m \geq n$ と仮定する. $\mathbf{\Sigma}_0$ が $m \times n$ の対角行列であることから, (15) の積 $\mathbf{U}_0 \mathbf{\Sigma}_0 \mathbf{V}^T$ の計算値に対して, $\mathbf{U}_0$ の先頭 n 列だけが寄与していることが分かる. つまり, $\mathbf{U}_0$ は本質的に $m \times n$ の行列でよいわけである. このような行列を $\mathbf{U}$ とする. 合わせて, $\mathbf{\Sigma}_0$ の下側の $(m-n) \times n$ の零行列を除いた $n \times n$ の行列を $\mathbf{\Sigma}$ とする. 行列 $\mathbf{U}$ と $\mathbf{\Sigma}$ を利用するならば, m が n よりかなり大きいと, それなりに記憶領域の節約ができる. そのうえ, 実用上, $\mathbf{U}_0$ 及び $\mathbf{\Sigma}_0$ より, $\mathbf{U}$ 及び $\mathbf{\Sigma}$ の方が便利である. $m \times n$ の場合の以上の事柄が, $m < n$ の場合にも準じて言える.

C-SSL II では, 以上の事柄を考慮して, (16) なる特異値分解を行っている.

$$\mathbf{A} = \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T \quad (16)$$

ここで, $l = \min(m, n)$ として, $\mathbf{U}$ は $m \times l$ の行列, $\mathbf{\Sigma}$ は $\mathbf{\Sigma} = \text{diag}(\sigma_i)$, $\sigma_i \geq 0$ なる l 次の対角行列, $\mathbf{V}$ は $n \times l$ の行列である.

かつ,

$l = n$ ($m \geq n$) のとき,

$$\mathbf{U}^T \mathbf{U} = \mathbf{V}^T \mathbf{V} = \mathbf{V} \mathbf{V}^T = \mathbf{I}_n$$

$l = m$ ($n \geq m$) のとき,

$$\mathbf{U}^T \mathbf{U} = \mathbf{U} \mathbf{U}^T = \mathbf{V}^T \mathbf{V} = \mathbf{I}_m$$

である.

行列 $\mathbf{A}$ を特異値分解することにより計算される行列 $\mathbf{U}$, $\mathbf{V}$ 及び $\mathbf{\Sigma}$ は, 次のような意味と用途を持つ.

- 特異値の性質

特異値 σ_i , $i = 1, 2, \dots, l$ は, 行列 $\mathbf{A}^T \mathbf{A}$, 及び $\mathbf{A} \mathbf{A}^T$ の固有値の, 大きい方から数えて l 番目までのものの正平方根である. また, $\mathbf{V}$ の第 i 列は, 固有値 σ_i^2 に対する $\mathbf{A}^T \mathbf{A}$ の固有ベクトルであり, $\mathbf{U}$ の第 i 列は, 固有値 σ_i^2 に対する $\mathbf{A} \mathbf{A}^T$ の固有ベクトルである. これらを確認するためには, $\mathbf{A}^T = \mathbf{V} \mathbf{\Sigma} \mathbf{U}^T$ を, (16) の左及び右から乗じ, $\mathbf{U}^T \mathbf{U} = \mathbf{V}^T \mathbf{V} = \mathbf{I}_l$ を利用して,

$$\begin{aligned} \mathbf{A}^T \mathbf{A} \mathbf{V} &= \mathbf{V} \mathbf{\Sigma}^2 \\ \mathbf{A} \mathbf{A}^T \mathbf{U} &= \mathbf{U} \mathbf{\Sigma}^2 \end{aligned}$$

となることを見ればよい.

- $\mathbf{A}$ の条件数

すべての i について, $\sigma_i > 0, i=1, 2, \dots, l$ であるとき, $\mathbf{A}$ の条件数は(17)で与えられる.

$$\text{cond}(\mathbf{A}) = \sigma_1 / \sigma_l \quad (17)$$

- $\mathbf{A}$ の階数

$\sigma_r > 0, \sigma_{r+1} = \dots = \sigma_l = 0$, であるとき, $\mathbf{A}$ の階数は(18)で与えられる.

$$\text{rank}(\mathbf{A}) = r \quad (18)$$

- 同次方程式 $\mathbf{Ax} = \mathbf{0}, \mathbf{A}^T \mathbf{y} = \mathbf{0}$ の基本解

同次方程式 $\mathbf{Ax} = \mathbf{0}$ 及び $\mathbf{A}^T \mathbf{y} = \mathbf{0}$ の $\mathbf{0}$ でない一次独立な解の組は, $\sigma_i = 0$ なる特異値に対応する $\mathbf{V}$ の列の組及び $\mathbf{U}$ の列の組によって, それぞれ与えられる. このことは, $\mathbf{AV} = \mathbf{U}\Sigma$ 及び $\mathbf{A}^T \mathbf{U} = \mathbf{V}\Sigma$ から容易にわかる.

- $\mathbf{Ax} = \mathbf{b}$ の最小二乗最小ノルム解

上記の解 $\mathbf{x}$ は, $\mathbf{A}$ の特異値分解を用いて

$$\mathbf{x} = \mathbf{V}\Sigma^+ \mathbf{U}^T \mathbf{b}$$

と表される. ただし, 対角行列 Σ^+ は

$$\Sigma^+ = \text{diag}(\sigma_1^+, \sigma_2^+, \dots, \sigma_l^+)$$

$$\sigma_i^+ = \begin{cases} 1/\sigma_i, & \sigma_i > 0 \\ 0, & \sigma_i = 0 \end{cases}$$

で定義される.

- 一般逆行列

$\mathbf{A}$ の一般逆行列 $\mathbf{A}^+$ は, 特異値分解により(19)で与えられる.

$$\mathbf{A}^+ = \mathbf{V}\Sigma^+ \mathbf{U}^T \quad (19)$$

a) 使用上の注意

連立 1 次方程式と係数行列の階数

$m \times n$ 行列を係数に持つ連立 1 次方程式 ($\mathbf{Ax} = \mathbf{b}$) を解く場合, 係数行列 $\mathbf{A}$ の行数と列数の大小関係や階数に制約されないで, 最小二乗最小ノルム解は求められる. いわば最小二乗最小ノルム解はいかなる方程式にも適用可能な解である. しかし, この解は求める過程の計算量が多いという欠点を持つので, 連立 1 次方程式の係数行列が縦長行列でかつ最大階数 (full rank, $\text{rank}(\mathbf{A}) = n$) であることが明らかである場合は, 最小二乗解のルーチンを使用した方が計算量は少なく合理的である. ちなみに, このような場合, 最小二乗解が理論上最小二乗最小ノルム解である.

最小二乗最小ノルム解と一般逆行列

$m \times n$ 行列 $\mathbf{A}$ ($m \geq n$ 又は $m < n, \text{rank}(\mathbf{A}) \neq 0$) の連立 1 次方程式 $\mathbf{Ax} = \mathbf{b}$ は, 通常解が一意に定まらないが, 最小二乗最小ノルム解は常に存在し一意に定まる. この解は係数行列 $\mathbf{A}$ の一般逆行列 $\mathbf{A}^+$ を求めるなら, $\mathbf{x} = \mathbf{A}^+ \mathbf{b}$ で計算できる. しかし, この方法では, 計算量が多く得策でない. この場合, 最小二乗最小ノルム解を求める専用ルーチンを用いるなら処理が速い. 特に, このルーチンには, 以降に説明するとおり, 同一の係数行列を持つ方程式をできるだけ少ない計算量で解いたり, 単一の方程式を効率よく解くことを利用者が指示できる引数 `isw` を用意しており, 幅広い用途がある.

同一の係数行列を持つ方程式について

連立 1 次方程式の最小二乗解や最小二乗最小ノルム解は, 通常次の手順により求められる.

- 係数行列の分解

最小二乗解のときは, 三角行列化を, 最小二乗最小ノルム解のときは, 特異値分解を行う.

- 求解

最小二乗解のときは, 後退代入を行い, 最小二乗最小ノルム解のときは, 行列やベクトルの乗算を行う.

ところが, 同一の係数行列を持つ複数組の方程式の最小二乗解若しくは最小二乗最小ノルム解を求める場合,

$$\mathbf{Ax}_1 = \mathbf{b}_1$$

$$\mathbf{Ax}_2 = \mathbf{b}_2$$

$$\vdots$$

$$\mathbf{Ax}_m = \mathbf{b}_m$$

各方程式で上記手順を繰り返し計算することは得策でない。この場合、最初の方程式を解くときにだけ、 $\mathbf{A}$ を分解し、それ以降は求解だけの手順を繰り返し適用すれば、計算量が節約できる。

C-SSL II には、 $\mathbf{A}$ の分解から行うか、分解を省略するかどうかを制御する引数 `isw` が用意されている。

特異値の求め方

特異値は、(20)のように特異値分解すれば得られる。

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \quad (20)$$

しかし、(20)に現れるとおり、数学的には、特異値からなる行列 $\mathbf{\Sigma}$ の他に行列 $\mathbf{U}$ 及び $\mathbf{V}$ が存在する。ところで、特異値分解のための計算量は多いので、利用者にとって行列 $\mathbf{U}$ 及び $\mathbf{V}$ が不要なときは計算されないことが望ましい。そこで、C-SSL II には、行列 $\mathbf{U}$ 又は $\mathbf{V}$ も求めるかどうかを制御する引数 `isw` が用意されている。また C-SSL II で扱う行列は、縦長行列、正方行列あるいは、横長行列のどれであってもよく、制約がない。

1.8 固有値固有ベクトル

1.8.1 概要

固有値問題は、問題の種類 ($\mathbf{Ax}=\lambda\mathbf{x}$, $\mathbf{Ax}=\lambda\mathbf{Bx}$)、行列の構造(密, バンド, スパース, 3 項), 行列の型(実, 複素)及び行列の形式(対称, 非対称)により、表 1. 10のように体系化される。したがって、行列 $\mathbf{A}$ の種類に応じ、それぞれの該当箇所を参照していただきたい。

表 1. 10 固有値問題の体系一覧

行列の構造	問題の種類	行列の型・形式	標準ルーチン名	説明箇所
密行列	$\mathbf{Ax}=\lambda\mathbf{x}$	実行列	c_deig1	1.8.2
		複素実行列	c_dceig2	1.8.3
		実対称行列	c_dseig1 c_dvseg2 c_dvsevp	1.8.4
		エルミート行列	c_dheig2 c_dvhevp	1.8.5
	$\mathbf{Ax}=\lambda\mathbf{Bx}$	実対称行列	c_dvgsg2	1.8.9
バンド行列	$\mathbf{Ax}=\lambda\mathbf{x}$	実対称バンド行列	c_dbseg c_dbsegj	1.8.6
	$\mathbf{Ax}=\lambda\mathbf{Bx}$	実対称バンド行列	c_dgbseg	1.8.10
スパース行列	$\mathbf{Ax}=\lambda\mathbf{x}$	実対称行列	c_dvland	1.8.7
3 項行列	$\mathbf{Ax}=\lambda\mathbf{x}$	実行列	c_dvtdev	1.8.8
		実対称行列	c_dteig1 c_dteig2	

1.8.2 実行列の固有値固有ベクトル

実行列の固有値固有ベクトルを求めるとき、C-SSL II の各関数をどのように用いるか標準的な例をあげて簡単に説明する。C-SSL II には、実行列の固有値固有ベクトルを 1 度に求めることができる標準ルーチンと、細分化された各機能を果たすコンポーネントルーチンが用意されている。

ここでは具体的に、利用者の問題を

- すべての固有値を求めるとき
- すべての固有値固有ベクトル(あるいはすべての固有値と一部の固有ベクトル)を求めるとき

に分け、コンポーネントルーチンと標準ルーチンの使い方を、それぞれの手順に従って、個々に説明する。

実行列の固有値固有ベクトルを求めるとき、コンポーネントルーチンを順次呼び出すか、あるいは標準ルーチンを使うかは、利用者に任されている。通常は、使いやすさの面で後者を利用することを勧めたい。ただし、指定された一部の固有値に対応する固有ベクトルを求めることは、前者によってだけ可能である。

すべての固有値を求めるとき

次の step 1~3 により、実行列 **A** のすべての固有値を求めることができる。

```
...
ierr = c_dblnc((double *)a, k, n, dv, &icon);          /* step 1 */
ierr = c_dhes1((double *)a, k, n, pv, &icon);          /* step 2 */
ierr = c_dhsqr((double *)a, k, n, er, ei, &m, &icon);   /* step 3 */
if (icon >= 20000) {
    /* output a message, maybe terminate processing */
}
...
```

1. **A** の平衡化を行う。平衡化が不要なら省略する。
2. ハウスホルダー法により **A** をヘッセンベルグ行列に変換する。
3. 2 段 QR 法によりヘッセンベルグ行列の固有値、すなわち実行列 **A** の固有値を求める。

すべての固有値固有ベクトル(あるいはすべての固有値と一部の固有ベクトル)を求めるとき

行列 **A** のすべての固有値固有ベクトルを求めるとき、次のように標準ルーチンを使うことができる。

```
...
ierr = c_deig1((double *)a, k, n, mode, er, ei, (double *)ev, vw, &icon);
if (icon >= 20000) {
    /* output a message, maybe terminate processing */
}
...
```

ただし、固有ベクトルは逆反復法を用いずに、固有値が求まる迄の変換行列を順次掛け合わせることで、すべての固有ベクトルすべてを 1 度に求めている。したがって、固有値が一つでも求まらない場合は、固有値ベクトルの計算は不可能となるが、一方固有値が近接根や重根となる場合においては、逆反復法に比べ、より正しく固有ベクトルを求めることができる。

関連するコンポーネントルーチンに `c_dhvec` と `c_dhbk1` がある。`c_dhvec` は固有値に対応する固有ベクトルを逆反復法により求める。引数 `ind` はどの固有値に対応する固有ベクトルを求めるかの指標ベクトルであり、これによって利用者は、部分固有ベクトルを求めるための制御を行うことができる。`c_dhbk1` は求めた固有ベクトルの逆変換を行う。あわせて正規化も行う。

行列の平衡化について

固有値固有ベクトルの誤差を減少させるためには、実行列 **A** のノルムを小さくすればよい。標準ルーチン `c_deig1` の引数 `mode`、及びコンポーネントルーチン `c_dblnc` はこのためのもので、対角行列による相似変換により、**A** のたがいにに対応する行と列の絶対和を等しく(平衡化)しようとするものである。(したがって、実対称行列やエルミート行列は、この意味で既に平衡化されている)

A に大きい要素や小さい要素が混在しているとき有効なことがあるので、その場合はできるだけ平衡化を行うことを勧めたい。平衡化の、パス全体の処理時間に占める割合は、特別な場合(A の次数が小さい)を除いて、多くても 10%程度である。

1.8.3 複素行列の固有値固有ベクトル

複素行列の固有値固有ベクトルを求めるとき、C-SSL II の各関数をどのように用いるか標準的な例をあげて簡単に説明する。C-SSL II には、複素行列に固有値固有ベクトルを一度に求めることができる標準ルーチンと、細分化された各機能を果すコンポーネントルーチンが用意されている。

ここでは具体的に、利用者の問題を

- すべての固有値を求めるとき
- すべての固有値固有ベクトルを求める(あるいは、すべての固有値と一部の固有ベクトルを求める)とき

に分け、コンポーネントルーチンと標準ルーチンの使い方を、それぞれの手順に従って個々に説明する。

すべての固有値固有ベクトルを求めるとき、標準ルーチンだけを使用するか、又は各コンポーネントルーチンを組み合わせて使用するかは利用者に任されている。ただし、使い易さの点で、通常は前者の使用を勧める。

すべての固有値を求めるとき

次の step 1~3 により、複素行列 A のすべての固有値を求めることができる。

```
...
ierr = c_dcbllnc((dcomplex *)za, k, n, dv, &icon);          /* step 1 */
ierr = c_dches2((dcomplex *)za, k, n, pv, &icon);           /* step 2 */
ierr = c_dchsqr((dcomplex *)za, k, n, ze, &m, &icon);        /* step 3 */
if (icon >= 20000) {
    /* output a message, maybe terminate processing */
}
...
```

1. A の平衡化を行う。平衡化が不要なら省略する。
2. 安定化基本相似変換により A を複素ヘッセンベルグ行列に変換する..
3. 複素 QR 法により複素ヘッセンベルグ行列の固有値、すなわち A の固有値を求める。

すべての固有値固有ベクトルを求めるとき(あるいは、すべての固有値と一部の固有ベクトルを求めるとき)行列 A のすべての固有値固有ベクトルを求めるとき、次のように標準ルーチンを使うことができる。

```
...
ierr = c_dceig2((dcomplex *)za, k, n, mode, ze, (dcomplex *)zev, vw, ivw, &icon);
if (icon >= 20000) {
    /* output a message, maybe terminate processing */
}
...
```

ただし、固有ベクトルは逆反復法を用いずに、固有値が求まるまでの変換行列を掛け合わせることによって、固有ベクトルを一度に求めている。したがって、求まらない固有値がある場合は、もはや固有ベクトルの計算は不可能となる。なお、行列の平衡化を行うかどうかは、利用者による引数 `mode` の指示による。

関連するコンポーネントルーチンには `c_dchvec` と `c_dchbk2` がある。`c_dchvec` は固有値に対応する複素ヘッセンベルグ行列の固有ベクトルを逆反復法により求める。引数 `ind` はどの固有値に対応する固有ベクトルかを求めるかの指標ベクトルであり、これによって利用者は、一部の固有ベクトルを求めるための制御を行うことができる。`c_dchbk2` は求めた固有ベクトルの正規化を行う。正規化(ノルムを 1 とする)の際に、ユークリッドノルムを用いるか又はインフィニティノルムを用いるかの選択が行える。

1.8.4 実対称行列の固有値固有ベクトル

実対称行列の固有値固有ベクトルを求めるとき、C-SSL II の各関数をどのように用いるか標準的な例をあげて簡単に説明する。C-SSL II には、実対称行列の固有値固有ベクトルを一度に求めることができる標準ルーチンと、細分化された各機能を果すコンポーネントルーチンが用意されている。

ここでは具体的に、利用者の持つ問題を、

- すべての固有値を求めるとき
- 一部の固有値を求めるとき
- すべての固有値固有ベクトルを求めるとき
- 一部の固有値固有ベクトルを求めるとき

に分け、標準ルーチンとコンポーネントルーチンの使い方を説明する。

全固有値固有ベクトルあるいは部分固有値固有ベクトルを求める場合、コンポーネントルーチンを順次呼び出すことによって求めるか、それとも標準ルーチンによって求めるかは利用者に任されている。後者は前者を組み合わせたものであるため、使いやすさという理由から、通常は後者を利用することを勧めたい。

なお、C-SSL II で扱う実対称行列は、対称行列用圧縮格納法で与えることが必要である。（“1.3 データの格納方法”参照）

すべての固有値を求めるとき

次の step 1～2 により実対称行列 **A** のすべての固有値を求めることができる。

```
...
ierr = c_dtrid1(a, n, d, sd, &icon);          /* step 1 */
ierr = c_dtrql(d, sd, n, e, &m, &icon);       /* step 2 */
if (icon >= 20000) {
    /* output a message, maybe terminate processing */
}
...
```

1. ハウスホルダー法により **A** を 3 重対角行列に変換する。**A** が既に 3 重対角行列である場合は、この処理は不要である。
2. QL 法により 3 重対角行列の固有値すなわち **A** の固有値を求める。

一部の固有値を求めるとき

次の step 1～2 により実対称行列 **A** の一部の固有値を求めることができる。

```
...
ierr = c_dtrid1(a, n, d, sd, &icon);          /* step 1 */
ierr = c_dbsct1(d, sd, n, m, epst, e, vw, &icon); /* step 2 */
if (icon >= 20000) {
    /* output a message, maybe terminate processing */
}
...
```

1. ハウスホルダー法により **A** を 3 重対角行列に変換する。**A** が既に 3 重対角行列である場合は、この処理は不要である。
2. バイセクション法により 3 重対角行列の固有値を求める。利用者は引数 *m* を用いて固有値の大きい方からいくつ求めるか、また小さい方からいくつ求めるかを指定する。ただし、**A** の次数 *n* に対して $n/4$ 以上の固有値を求めたいときは、`c_dtrql` で対処した方が一般的には処理速度は速い。

すべての固有値固有ベクトルを求めるとき

実対称行列 **A** のすべての固有値固有ベクトルを求めるとき、次のように標準ルーチンを使うことができる。

...

```

ierr = c_dseig1(a, n, e, (double *)ev, k, m, vw, &icon);
if (icon >= 20000) {
    /* output a message, maybe terminate processing */
}
...

```

実対称行列 **A** のすべての固有値固有ベクトルを求めるには、まず行列 **A** を 3 重対角行列に変換し、QL 法と、QL 法による変換行列を順次掛けあわせることにより、3 重対角行列のすべての固有値固有ベクトルを求める。なお、固有ベクトルはユークリッドノルムが 1 となるように正規化されている。最後に 3 重対角行列の固有ベクトルを実対象行列 **A** の固有ベクトルへ逆変換する。

一部の固有値固有ベクトルを求めるとき

実対称行列 **A** の一部の固有値固有ベクトルを求めるとき、次のように標準ルーチンを使うことができる。

```

...
ierr = c_dvseg2(a, n, m, epst, e, (double *)ev, k, vw, ivw, &icon);
if (icon >= 20000) {
    /* output a message, maybe terminate processing */
}
...

```

実対称行列 **A** の一部の固有値固有ベクトルを求めるには、まず行列 **A** を 3 重対角行列に変換し、バイセクション法と逆反復法により 3 重対角行列の一部の固有値固有ベクトルを求める。逆反復法により求めた固有ベクトルは、固有値が接近している場合、必ずしも互いに直交するとは限らない。したがって、接近した固有値に対する固有ベクトルをすでに求めた分に対して直交するように修正することをあわせて行っている。また、求められた固有ベクトルは、ユークリッドノルムが 1 となるように正規化されている。最後に 3 重対角行列の固有ベクトルを実対象行列 **A** の固有ベクトルへ逆変換する。

QL 法について

QL 法は、実行例の固有値を求めるために採用している QR 法と基本的には同じである。ただ、QR 法では固有値を行列の右下すみから求めるのに対して、QL 法では、左上すみから求める。これらのことは行列のデータの並び具合に関連している。すなわち、前者は行列の各要素の絶対値が左上から右下に進むに従って小さくなっている場合に最適であり、後者は逆に左上から右下に進むに従って大きくなっている場合に最適である。c_dtrid1 によって出力される 3 重対角行列は、一般には右下方の要素の絶対値が左上方に比べて大きくなりやすい。QL 法を採用しているのはこのためである。

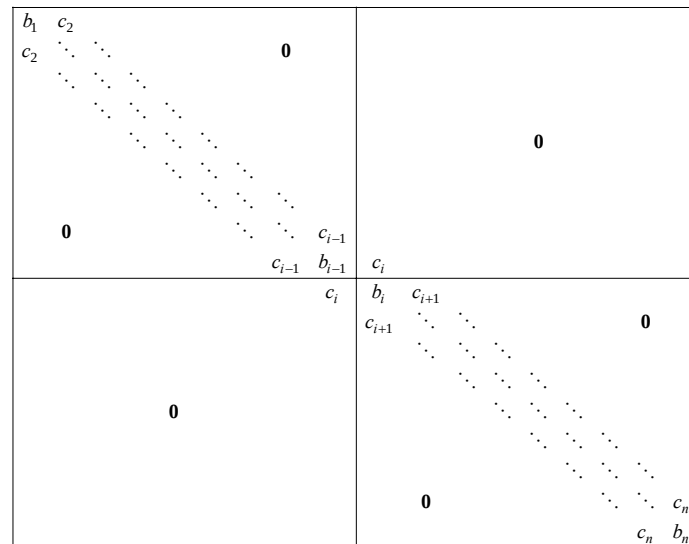
QL 法には、更に明示型 (explicit) QL 法と黙示型 (implicit) QL 法とがある。両者の違いは、固有値を求める収束速度を速くするための手段：原点移動を陽に行うか、暗に行うかの違いである。前者は各要素の絶対値が左上から右下へ進むにつれて大きくなっている場合はよいが、その逆の場合には小さな固有値の精度はかなり悪くなる。これをある程度カバーするものとして、黙示型 QL 法を用いている。

行列の直和分解について

固有値固有ベクトルを求めるとき、できるだけ行列を小行列に分解(直和分解)して、各小行列ごとに固有値固有ベクトルを求めた方が精度や処理速度の点で望ましい。C-SSL II では、3 重対角行列を(21)の条件を調べることで小行列に分解し、固有値固有ベクトルを求めている。

$$|c_i| \leq \mu (|b_{i-1}| + |b_i|) \quad , \quad i = 2, 3, \dots, n \quad (21)$$

ここで μ は丸め誤差の単位であり、 c_i, b_i は図 1.18 に示すとおりである。



注) c_i は (21)により零と見なした要素.

図 1.18.3 重対角行列が二つの小行列に直和分解された例

1.8.5 エルミート行列の固有値固有ベクトル

エルミート行列の固有値固有ベクトルを求めるとき, C-SSL II の各関数をどのように用いるか標準的な例をあげて簡単に説明する.

ここでは具体的に, エルミート行列についての利用者の問題を目的別に,

- すべての固有値を求めるとき
- 一部の固有値を求めるとき
- すべて(あるは一部)の固有値固有ベクトルを求めるとき

のように分ける, 以下ではそれぞれの目的別に分類し, 標準ルーチンとコンポーネントルーチンの使い方を説明する.

固有値固有ベクトルのすべて又は, 一部を求める場合において, 標準ルーチンだけを使用するか, あるいは各コンポーネントルーチンを使用するかは利用者に任されている. ただし, 使いやすさという点で, 通常は前者を使うことを勧める.

なお, C-SSL II で扱うエルミート行列は, エルミート行列用圧縮格納法で与えることが必要である. (“1.3 データの格納方法”参照)

すべての固有値を求めるとき

次の step 1~2 により, エルミート行列 A のすべての固有値を求めることができる.

```

...
ierr = c_dtridh((double *)a, k, n, d, sd, v, &icon);          /* step 1 */
ierr = c_dtrql(d, sd, n, e, m, &icon);                        /* step 2 */
if (icon >= 20000) {
    /* output a message, maybe terminate processing */
}
...

```

1. ハウスホルダー法によりエルミート行列 A を実対称 3 重対角行列に変換する.
2. QL 法により実対称 3 重対角行列の全固有値を求める.

一部の固有値を求めるとき

次の step 1~2 により, A の固有値のうち大きい方, 又は小さい方から m 個求めることができる.

```
...
ierr = c_dtridh((double *)a, k, n, d, sd, v, &icon);          /* step 1 */
ierr = c_dbsct1(d, sd, n, m, epst, e, vw, &icon);             /* step 2 */
if (icon >= 20000) {
    /* output a message, maybe terminate processing */
}
...
```

1. ハウスホルダー法によりエルミート行列 A を実対称 3 重対角行列へ変換する.
2. バイセクション法により実対称 3 重対角行列の固有値を大きい方(又は小さい方)から m 個求める. ただし, A の次数 n に対して $n/4$ 以上の固有値を求めるときは, `c_dtridh` とすべての固有値を求める関数 `c_dtrql` を使う方が一般に処理は速い.

すべて(あるいは一部)の固有値固有ベクトルを求めるとき

エルミート行列 A の固有値固有ベクトルを求めるとき, 次のように標準ルーチンを使うことができる.

```
...
ierr = c_dheig2((double *)a, k, n, m, e, (double *)evr, (double *)evi, vw, &icon);
if (icon >= 20000) {
    /* output a message, maybe terminate processing */
}
...
```

エルミート行列 A の固有値固有ベクトルを求めるには, まず行列 A を実対称 3 重対角行列に変換し, バイセクション法により実対称 3 重対角行列の固有値固有ベクトルを求める. このとき `c_dheig2` の第 4 引数 m は, 固有値を大きいほうから m 個求めることを指定している. m に n を指定すると, すべての固有値固有ベクトルを求めることができる.

1.8.6 実対称バンド行列の固有値固有ベクトル

実対称バンド行列の固有値固有ベクトルを求めるための関数として, `c_dbseg`, `c_dbsej`, `c_dbtrid`, `c_dbsvec` などが用意されている.

これらは一般に, 行列の次数 n が 100 以上の大型行列で, かつ行列のバンド幅 h との間で $h/n < 1/6$ である場合に適している. またジェニングス法による `c_dbsej` は, $n/10$ 個以内の固有値を求める場合に有効である. 実対称バンド行列の場合, 全固有値固有ベクトルを要求するケースは一般に少なく, したがって, 標準ルーチンもその趣旨に沿って用意されている. `c_dbseg` と `c_dbsej` は標準ルーチンであり, `c_dbtrid` と `c_dbsvec` は `c_dbseg` を構成するコンポーネントルーチンである. 今, これらの各関数を利用者の問題に応じて, どのように使うことができるか例をあげて説明する. なお, C-SSL II で扱う実対称バンド行列は, 対称バンド行列用圧縮格納法で与える必要がある (“1.3 データの格納方法”参照).

一部の固有値を求めるとき

次数 n , バンド幅 h の実対称バンド行列 A の固有値を, 大きい方(又は小さい方)から m 個求めるとき, 次のように標準ルーチンを使うことができる.

```
...
ierr = c_dbseg(a, n, nh, m, 0, epst, e, (double *)ev, k, vw, &icon);
if (icon >= 20000) {
    /* output a message, maybe terminate processing */
}
...
```

第 5 引数が 0 であるのは固有ベクトルが不要であることを意味する.

全固有値を求めるとき

一部の固有値を求めるときの `c_dbseg` の例において, 第 4 引数を n とすることにより全固有値を求めることができるが, 通常は次のコンポーネントルーチンによるパスを利用することを勧める.

```

...
ierr = c_dbtrid(a, n, nh, d, sd, &icon);      /* step 1 */
ierr = c_dtrql(d, sd, n, e, &m, &icon);      /* step 2 */
if (icon >= 20000) {
    /* output a message, maybe terminate processing */
}
...

```

1. 次数 n , バンド幅 h の実対称バンド行列 $\mathbf{A}$ をルティスハウザー・シュワルツ法により, 実対称 3 重対角行列 $\mathbf{T}$ へ変換する.
2. $\mathbf{T}$ の全固有値を QL 法により求める.

一部の固有値固有ベクトルを求めるとき
標準ルーチンとして次の二つがある.

c_dbseg

```

...
ierr = c_dbseg(a, n, nh, m, nv, epst, e, (double *)ev, k, vw, &icon);
if (icon >= 20000) {
    /* output a message, maybe terminate processing */
}
...

```

c_dbseg は次数 n , バンド幅 h の実対称バンド行列 $\mathbf{A}$ の固有値を大きい方(又は小さい方)から m 個, 固有ベクトルを n_v 個求める. ルティスハウザー・シュワルツ法, バイセクション法, 逆反復法を順次用いて, まず固有値を求め, それから固有ベクトルを求める.

c_dbsegj

```

...
ierr = c_dbsegj(a, n, nh, m, epst, lm, e, (double *)ev, k, &it, vw, &icon);
if (icon >= 20000) {
    /* output a message, maybe terminate processing */
}
...

```

c_dbsegj は与えられた m 個の初期固有ベクトルをもとに, 同時反復によるジェニングス法を用いて, 上記同行列 $\mathbf{A}$ の固有値固有ベクトルを同時に求める. 固有ベクトルの初期値及び反復回数の上限を ev と lm に与える点で若干の考慮が必要である. 固有値は絶対値の大きい順(又は小さい順)から求める. 手法の性格上, 行列の次数 n にくらべて極めて少ない($n/10$ 以下)固有値固有ベクトルを求める場合に, その利用が勧められる.

全固有値固有ベクトルを求めるとき

一部の固有値固有ベクトルを求めるときの c_dbseg の例で, 第 4 及び第 5 引数を n とすることにより全固有値固有ベクトルを求めることができる.

1.8.7 対称スパース行列の固有値固有ベクトル

大規模な実対称スパース行列の最大・最小固有値のいくつかと対応する固有ベクトルを求めるために関数 c_dvland を提供している. このルーチンでは Lanczos 法を使用している.

1.8.8 実 3 項行列の固有値固有ベクトル

実 3 重対角行列の指定された固有値固有ベクトルを求めるために, 関数 c_dvtddev を提供している. 固有値の計算には Sturm カウントに基づいたアルゴリズムを用いており, 固有ベクトルは逆反復法で計算する.

また, 実対称 3 重対角行列の固有値固有ベクトルを求めるためには, 関数 c_dteig2 を使うこともできる.

1.8.9 実対称行列の一般固有値固有ベクトル

$\mathbf{Ax}=\lambda\mathbf{Bx}$ ($\mathbf{A}$: 対称行列, $\mathbf{B}$: 正値対称行列)型の固有値固有ベクトルを求めるとき, C-SSL II の各関数をどのように用いるか標準的な例をあげて簡単に説明する.

実対称行列の一般固有値固有ベクトルを求める手順は,

1. 一般固有値問題 ($\mathbf{Ax}=\lambda\mathbf{Bx}$) を実対称行列の標準固有値問題 ($\mathbf{Sy}=\lambda\mathbf{y}$) に変換する.
2. 実対称行列 $\mathbf{S}$ を実対称 3 重対角行列 $\mathbf{T}$ に変換する ($\mathbf{Sy}=\lambda\mathbf{y} \rightarrow \mathbf{Ty}'=\lambda\mathbf{y}'$ への変換).
3. 実対称 3 重対角行列 $\mathbf{T}$ の固有値 λ を求める.
4. 実対称 3 重対角行列 $\mathbf{T}$ の固有ベクトル $\mathbf{y}'$ を求める.
5. 実対称 3 重対角行列 $\mathbf{T}$ の固有ベクトル $\mathbf{y}'$ を実対称行列 $\mathbf{S}$ の固有ベクトル $\mathbf{y}$ に逆変換する.
6. 実対称行列 $\mathbf{S}$ の固有ベクトル $\mathbf{y}$ を一般固有値問題の固有ベクトル $\mathbf{x}$ に逆変換する.

のように 6 段階に分けられる. C-SSL II では, これらの手順に対応してそれぞれのコンポーネントルーチンと, 全手順を一度に行うことができる標準ルーチンが用意されている.

ここでは具体的に, 実対称行列の一般固有値固有ベクトルについての利用者の問題を目的別に,

- すべての固有値を求めるとき
- 一部の固有値を求めるとき
- すべての固有値固有ベクトルを求めるとき
- 一部の固有値固有ベクトルを求めるとき

のように分け, それぞれについて, 標準ルーチンとコンポーネントルーチンの使い方を説明する.

固有値固有ベクトルのすべて又は, 一部を求めるとき, コンポーネントルーチンを順次呼び出すか, あるいは標準ルーチンを使用するかは利用者に任されている. ただ, 使いやすさという点で, 通常は後者を使うことを勧める.

なお, C-SSL II で扱う実対称行列は, 対称行列用圧縮格納法で与えることが必要である. (“1.3 データの格納方法”参照)

すべての固有値を求めるとき

次の step 1~3 により, すべての固有値を求めることができる.

```

...
ierr = c_dgschl(a, b, n, epsz, &icon);          /* step 1*/
if (icon >= 20000) {
    /* output a message, maybe terminate processing */
}
ierr = c_dtrid1(a, n, d, sd, &icon);            /* step 2*/
ierr = c_dtrql(d, sd, n, e, m, &icon);          /* step 3*/
...

```

1. 実対称行列の一般固有値問題 ($\mathbf{Ax}=\lambda\mathbf{Bx}$) を標準固有値問題 ($\mathbf{Sy}=\lambda\mathbf{y}$) に変換する.
2. ハウスホルダー法により実対称行列 $\mathbf{S}$ を実対称 3 重対角行列に変換する.
3. QL 法により実対称 3 重対角行列のすべての固有値を求める.

一部の固有値を求めるとき

次の step 1~3 により, 固有値を大きい方(又は小さい方)から m 個求めることができる.

```

...
ierr= c_dgschl(a, b, n, epsz, &icon);           /* step 1*/
if (icon >= 20000) {
    /* output a message, maybe terminate processing */
}

```



```

ierr = c_dtrid1(a, n, d, sd, &icon);          /* step 2*/
ierr = c_dbsct1(d, sd, n, m, epst, e, vw, &icon); /* step 3*/
...

```

1. 実対称行列の一般固有値問題 ($\mathbf{Ax} = \lambda \mathbf{Bx}$) を標準固有値問題 ($\mathbf{Sy} = \lambda \mathbf{y}$) に変換する.
2. ハウスホルダー法により実対称行列 $\mathbf{S}$ を実対称 3 重対角行列に変換する.
3. バイセクション法により実対称 3 重対角行列の固有値を大きい方(又は小さい方)から m 個求める.

ここで、 $\mathbf{A}$ の次数 n に対して $n/4$ 以上の固有値を求めたいときは、すべての固有値を求める関数 `c_dtrql` を使う方が一般に処理は速い.

すべての固有値固有ベクトルを求めるとき

実対称行列 $\mathbf{A}$ のすべての一般固有値固有ベクトルを求めるとき、次のように標準ルーチンを使うことができる.

```

...
ierr = c_dvgseg2(a, b, n, n, epsz, epst, e, (double *)ev, k, vw, &icon);
if (icon >= 20000) {
    /* output a message, maybe terminate processing */
}
...

```

`c_dvgseg2` の第 4 引数を n としているのは、固有値を大きい方からすべて求めることを指定した場合である.

一部の固有値固有ベクトルを求めるとき

実対称行列 $\mathbf{A}$ の一部の一般固有値固有ベクトルを求めるとき、次のように標準ルーチンを使うことができる.

```

...
ierr = c_dvgseg2(a, b, n, m, epsz, epst, e, (double *)ev, k, vw, &icon);
if (icon >= 20000) {
    /* output a message, maybe terminate processing */
}
...

```

`c_dvgseg2` の第 4 引数を m としているのは、固有値を大きい方から m 個求めることを指定した場合である.

1.8.10 実対称バンド行列の一般固有値固有ベクトル

C-SSL II では $\mathbf{Ax} = \lambda \mathbf{Bx}$ ($\mathbf{A}$: 対称バンド行列, $\mathbf{B}$: 正値対称バンド行列) 型の固有値固有ベクトルを求めるための標準ルーチンとして、`c_dgbseg` を用意している. これは一般には大型行列で、かつ行列の次数 n 、バンド幅 h との間に、 $h/n < 1/6$ である場合に適している. `c_dgbseg` はジェニングス法を用いており、 $n/10$ 個以内の固有値固有ベクトルを求める場合時間的に有効である. 指定された m 個の固有値と固有ベクトルを同時に求めるので、正常終了しないときは、固有値と固有ベクトルが一つも求まっていないので注意すること.

今、利用者の問題に応じて、関数をどのように使うことができるか例をあげて説明する. なお、C-SSL II で扱う実対称バンド行列は、対称バンド行列用圧縮格納法で与える必要がある. (“1.3 データの格納方法”参照)

一部の固有値固有ベクトルを求めるとき

```

...
ierr = c_dgbseg(a, b, n, nh, m, epsz, epst, lm, e, (double *)ev, k, &it, vw, &icon);
if (icon >= 20000) {
    /* output a message, maybe terminate processing */
}
...

```

同時反復法によるジェニングス法を用いて、固有値固有ベクトルを求める. 引数 m の与え方により、固有値の絶対値の大きい方又は小さい方から m 個の固有値固有ベクトルが求められる.

1.9 非線型計算

1.9.1 概要

この節では以下の問題を取り上げる.

方程式の根：代数方程式, 超越方程式, 連立非線型方程式の根を求める.

1.9.2 代数方程式

代数方程式の関数を表 1. 11に示す.

表 1. 11 代数方程式の関数

目的	関数名	手法	備考
実係数 2 次方程式	c_drqdr	根の公式	
複素係数 2 次方程式	c_dcqdr	根の公式	
実係数低次代数方程式	c_dlowp	代数的手法と反復的手法を併用	次数は 5 次以下
実係数高次代数方程式	c_drjetr	ジェンキンス・トラウプの方法	
複素係数高次代数方程式	c_dcjart	ヤラット法	

5 次以下の実係数代数方程式を解くときは c_dlowp を使えばよいが, 特に 2 次方程式だけを解く場合は c_drqdr を使うこと.

代数方程式を(22)で表す.

$$a_0x^n + a_1x^{n-1} + \cdots + a_n = 0, \quad |a_0| \neq 0 \quad (22)$$

ここで $a_i (i = 0, 1 \dots n)$ は実数又は複素数である.

a_i が実数の場合を実係数代数方程式と言い, a_i が複素数の場合を複素係数代数方程式と言う. 後者の場合は, (22)において特に x の代りに z を用いることにする. 代数方程式を解く関数は断らないかぎりすべての根を求めることを目的としている.

以下, 手法とそれにかかわる事柄について述べる.

代数方程式を解く手法には, 代数的手法と反復的手法とがある. 代数的手法とは, 4 次以下の方程式に対して代数学で言うところの“根の公式”を用いて解く方法であり, 一方反復的手法とは, 任意の次数の方程式に対して根の大ざっぱな近似値を設定しそれを反復的に改良してゆき最後に良い近似値を得る, という手法である. 反復的手法は多くの場合, 根を一つずつ順番に求めていく方法を取り, ある根が求まったあとその根を除いた, より低次の代数方程式を作り出してまた別の根を求めるという方法である.

代数的手法と反復的手法とを優劣の点で比較するのは難しいように思われる. というのは各々独自の難点を持っているからである. それを以下に挙げる.

代数的手法の難点

(22) の係数 a_i に, 大きさにおいて激しいへだたりがあるものが混在している場合は計算過程でオーバーフローあるいはアンダフローを起こす可能性がある.

反復的手法の難点

初期の近似値を設定することが難しい。それを誤ると反復しても近似値が収束しない。逆に近似値が収束しなかったときは初期近似の設定に誤りがあると判断できる。また、ある根は求まっても別な根は求まらないということもある。さらに、反復的に求まってくる近似値が収束したか否かの判定、いわゆる収束判定という手間の問題がある。

C-SSL II においては、特に代数的手法の難点をさけることに重点をおき、2 次方程式の場合を除いて、もっぱら反復的手法に頼っている。

ここで収束判定法について、C-SSL II の関数がとっている方法を述べておく。代数方程式

$$f(x) \equiv \sum_{k=0}^n a_k x^{n-k} = 0$$

を反復的に求めるときは $f(x)$ の計算値が計算誤差の範囲内の大きさになってしまうと、もはやそれ以上 $f(x)$ を小さくしても意味がない。ところで $f(x)$ を計算するときの計算誤差の上限を $\varepsilon(x)$ とすれば、誤差理論より

$$|\varepsilon(x)| = \mu \sum_{k=0}^n |a_k x^{n-k}|$$

が成り立つ。ここでは μ は丸め誤差の単位である。したがって

$$|f(x)| \leq |\varepsilon(x)|$$

を満たす x の領域において、 x が真の根であるかどうかを確認する方法はない。ゆえに、

$$\left| \sum_{k=0}^n a_k x^{n-k} \right| \leq \mu \sum_{k=0}^n |a_k x^{n-k}|$$

を満たす x が求まったとき収束したと判断し、それを一つの根として採用する。

最後に根の精度について注意を述べる。代数的手法であれ反復的手法であれ、一定の桁数で計算する限り、ある根は精度良く求まり、ある根はそれに反して精度良く求まらないことが起こる。一般に、重根あるいは近接根は、そうでない根の精度ほどにはよく求まらない。利用者はこのことに注意され、もし求まった根のなかに近接しているものがあれば、そうでない根ほどには精度は良くないと考えてもらいたい。

1.9.3 超越方程式

超越方程式を(23)で表す。

$$f(x) = 0 \quad (23)$$

$f(x)$ が実関数の場合を実超越方程式と言い、複素関数の場合を複素超越方程式と言う。後者の場合は(23)において特に x の代りに z を用いることにする。超越方程式を解く関数は、独立変数の指定された範囲内の、あるいは指定された点の近傍の、ただ一つの根を求めることを目的としている。

超越方程式の関数を表 1.12 に示す。

表 1.12 超越方程式の関数

目的	関数名	手法	備考
実超越方程式	c_dtsd1	2 分法, 線型補間法, 逆 2 次補間法を併用	微係数不要区間指定
	c_dtsdm	マラー法	微係数不要初期値指定
複素超越方程式	c_dctsdm	マラー法	微係数不要初期値指定

超越方程式を解く手法はもっぱら反復的手法である。反復的手法における収束の速さはまず第 1 に指定された範囲がいかに狭いか、あるいは指定された点がいかに根に近いかに依存している。収束判定法については関数ごとに異なるので個々の説明のところで述べてある。

1.9.4 連立非線型方程式

方程式を(24)で表す.

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \quad (24)$$

ここで, $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_n(\mathbf{x}))^T$, $\mathbf{x} = (x_1, x_2, \dots, x_n)^T$ 及び $\mathbf{0}$ は n 次元ベクトルである.

連立非線型方程式の解法は, 反復的手法である. すなわち, 利用者が与える初期ベクトル $\mathbf{x}_0$ から出発して, 逐次改良していくことにより, (24)の解ベクトルを, 必要な精度まで求めようとするものである.

連立非線型方程式の関数を表 1.13に示す.

表 1.13 連立非線型方程式の関数

目的	関数名	手法	備考
連立非線型方程式	c_dnlbr	ブレント法	微係数不要

反復的手法の中で最もよく知られているのはニュートン法であり, それは (25) で表せる.

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{J}_i^{-1} \mathbf{f}(\mathbf{x}_i), \quad i = 0, 1, \dots \quad (25)$$

ここで $\mathbf{J}_i$ は, $\mathbf{x} = \mathbf{x}_i$ における $\mathbf{f}(\mathbf{x})$ のヤコビアン行列であり, (26) を意味する.

$$\mathbf{J}_i = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \dots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}_{\mathbf{x}=\mathbf{x}_i} \quad (26)$$

このニュートン法は, 理論的には良い方法である. すなわち, 収束の速さは 2 次であり, 計算式が単純であるという利点がある. しかし, 複雑な(又は大規模な)方程式になると, 計算上のいくつかの問題点が出てくる. 主な原因は次の 3 つである.

- (26)の係数 $\partial f_i / \partial x_j$ の計算式を作るのが困難である(式が複雑で事実上偏微分できないことがある).
- (26)の全要素の計算量が多すぎる.
- 反復 1 回ごとに, $\mathbf{J}_i$ を係数行列とする連立 1 次方程式を解くので時間がかかる.

これらの問題点を解消し, しかも収束の速さを 2 次に保つことができれば, 利用者の使いやすさ及び計算時間の短縮が期待できる. 上記の問題点を解消する具体的な方法としては以下の方法が考えられる.

最初の問題に対しては, $\partial f_i / \partial x_j$ を差分で近似する. すなわち, 適当な $h (> 0)$ を選ぶことにより,

$$\frac{\partial f_i}{\partial x_j} \approx \frac{f_i(x_1, \dots, x_j + h, \dots, x_n) - f_i(x_1, \dots, x_n)}{h}$$

とする.

残りの 2 つの問題に対しては, ヤコビアン行列を直接求めないで, 計算量が少なく済むような, 擬似的(全要素を求めないという意味で)なヤコビアン行列を使って連立 1 次方程式を解く.

C-SSL II ではこれらを実現するような手法を採用している.

次に, 連立非線型方程式の関数を使用する上での注意を述べる.

利用者は, 方程式を定義する関数系を計算するルーチンを用意する必要がある. このルーチンは, C-SSL II ルーチンを効率良く利用し, 指定した精度の解を得るためにも, 次の点に十分注意して用意しなければならない.

- 関数の計算は、桁落ちを避けるように工夫すること。これは、この関数値が、関数内で微係数を評価するために使われることから特に重要である。
- 変数ベクトル $\mathbf{x}$ や関数ベクトル $\mathbf{f}(\mathbf{x})$ などの各要素の大きさをそろえることが望ましい。なぜならば、計算の過程で、絶対値が極めて大きな要素のために、他の要素が無視されてしまうようなことが起きるからである(特に本関数では、収束判定において、最大値要素の変化を見て収束したか否かの判断をしているため)。

また、解ベクトルの精度は、利用者の与える収束判定値に依存する。一般に、収束判定値を小さくすれば、解ベクトルの精度の向上が期待できる。しかしながら、丸め誤差の影響のため精度の限界があるので、むやみに収束判定値を小さく与えても、それだけの精度が得られないことがあるので注意を要する。

次に、初期ベクトル $\mathbf{x}_0$ の選び方であるが、これは方程式の物理的情報によってなされるべきであり、利用者に任された問題である。もし、そのような物理的情報がない場合は、初期ベクトルをいろいろ変えて、何回か試行するのもよい。

1.10 極値問題

1.10.1 概要

この節では、以下の問題を取り上げる。

- 1 変数関数の極小化
- 制約なし多変数関数の極小化
- 制約なし関数二乗和の極小化(非線型最小二乗解)
- 線形計画問題
- 非線形計画問題(制約付き多変数関数の極小化)

1.10.2 1 変数関数の極小化

この問題では、1 変数の実関数 $f(x)$ が与えられたとき、区間 $[a, b]$ で $f(x)$ の極小点 x^* とその関数値 $f(x^*)$ を求める。

a) ルーチンの種類

利用者が関数 $f(x)$ 以外に、導関数 $g(x)$ を解析的に定義できるか否かの条件に応じて、表 1. 14に示す関数が用意されている。

表 1. 14 1 変数関数の極小化の関数

解析的定義	関数名	備考
$f(x)$	c_dlminf	2 次補間法
$f(x), g(x)$	c_dlmng	3 次補間法

b) 区間 $[a, b]$ の与え方

C-SSL II では、区間 $[a, b]$ で $f(x)$ が単峰であることを仮定し、 $f(x)$ の唯一の極小点を許容誤差の範囲内で求める。区間 $[a, b]$ に多くの極小点を持つ場合には、いずれの極小点に収束するかは保証されない。したがって、求める極小点 x^* を含む区間の端点 a 及び b の値は、可能なかぎり x^* の近くにとることが望ましい。

1.10.3 制約なし多変数関数の極小化

この問題では、 n 変数の実関数 $f(\mathbf{x})$ と初期ベクトル $\mathbf{x}_0$ が与えられたとき、 $f(\mathbf{x})$ の極小値をとるベクトル(極小点) $\mathbf{x}^*$ と、その関数値 $f(\mathbf{x}^*)$ を求める。ここで $\mathbf{x}$ は、 $\mathbf{x} = (x_1, x_2, \dots, x_n)^T$ なるベクトルである。

関数の極小化とは、一般に任意の初期ベクトル $\mathbf{x}_0$ から出発し、 $f(\mathbf{x}_{k+1}) < f(\mathbf{x}_k)$, $k = 0, 1, \dots$ なる関係を満たすように反復し、極小点 $\mathbf{x}^*$ を求めることである。ここで反復ベクトルは、関数 $f(\mathbf{x})$ が $\mathbf{x}_k$ において局所的に減少する方向に基づき修正されるが、そのため通常、 $f(\mathbf{x})$ の値だけでなく、傾斜ベクトル $\mathbf{g}$ とヘシアン行列 $\mathbf{B}$ とが利用される。

$$\mathbf{g} = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right)^T$$

$$\mathbf{B} = (b_{ij}), \quad b_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$$

a) ニュートン法を基本とした公式

関数 $f(\mathbf{x})$ が 2 次関数で、凹関数である場合、ニュートン法による反復公式により、理論的には高々 n 回の反復で最小点 $\mathbf{x}^*$ を求めることができる。

ところで、一般の関数の場合は、極小点 $\mathbf{x}^*$ の近傍では近似的に 2 次関数となる。すなわち、

$$f(\mathbf{x}) \approx f(\mathbf{x}^*) + \frac{1}{2}(\mathbf{x} - \mathbf{x}^*)^T \mathbf{B}(\mathbf{x} - \mathbf{x}^*) \quad (27)$$

と表せる。したがってヘシアン行列 $\mathbf{B}$ が正定値行列であるなら、2 次関数の場合に適用されるニュートン法を基本とした反復公式は、一般の関数に対しても(27)に基づき良い反復公式となることが期待できる。そこで極小点 $\mathbf{x}^*$ の近傍の任意な点 $\mathbf{x}_k$ における傾斜ベクトルを $\mathbf{g}_k$ とすると、(27)よりニュートン法の基本反復公式を得る。

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{B}^{-1} \mathbf{g}_k \quad (28)$$

C-SSL II では、(28)に基づく、次の二つの反復公式を導入している。

改訂準ニュートン法

反復公式：

$$\mathbf{B}_k \mathbf{p}_k = -\mathbf{g}_k$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$$

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \mathbf{E}_k$$

ここで、 $\mathbf{B}_k$ はヘシアン行列の近似行列であり、反復の過程で階数 2 の行列 $\mathbf{E}_k$ により改良される。 $\mathbf{p}_k$ は、関数が局所的に減少する方向を定める探索ベクトルである。 α_k は、 $f(\mathbf{x}_{k+1})$ が局所的最小となるように定められる(直線探索)定数である。

準ニュートン法

反復公式：

$$\begin{aligned}\mathbf{p}_k &= -\mathbf{H}_k \mathbf{g}_k \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + \alpha_k \mathbf{p}_k \\ \mathbf{H}_{k+1} &= \mathbf{H}_k + \mathbf{F}_k\end{aligned}$$

ここで、 $\mathbf{H}_k$ はヘシアン行列の逆行列 $\mathbf{B}^{-1}$ の近似行列であり、反復の過程で階数 2 の行列 $\mathbf{F}_k$ により改良される。 $\mathbf{p}_k$ は、関数が局所的に減少する方向を定める探索ベクトルである。 α_k は、 $f(\mathbf{x}_{k+1})$ が局所的最小となるように定められる(直線探索)定数である。

b) ルーチンの種類

利用者が関数 $f(\mathbf{x})$ 以外に傾斜ベクトル $\mathbf{g}$ を解析的に定義できるか否かの条件に応じて、表 1.15 に示す関数が用意されている。

表 1.15 制約なし多変数関数の極小化の関数

解析的定義	関数名	備考
$f(\mathbf{x})$	c_dminf1	改訂準ニュートン法
$f(\mathbf{x}), \mathbf{g}(\mathbf{x})$	c_dming1	準ニュートン法

c) 使用上の注意

初期ベクトル $\mathbf{x}_0$ の与え方

初期ベクトル $\mathbf{x}_0$ は、可能なかぎり期待する極小点 $\mathbf{x}^*$ の近傍に取ることが望ましい。関数 $f(\mathbf{x})$ がいくつかの極小点を持つ場合、初期ベクトルが適切でないと、期待する極小点 $\mathbf{x}^*$ に収束しない可能性がある。通常、 $\mathbf{x}_0$ は関数 $f(\mathbf{x})$ の持つ物理的情報によって設定されるべきであろう。

関数計算プログラム

利用者は、関数 $f(\mathbf{x})$ 、傾斜ベクトル $\mathbf{g}$ の値を計算するルーチンを用意する場合、効率の良いコーディングをすることが望ましい。C-SSL II ルーチンによる各関数の評価回数は、手法、初期ベクトルなどに依存するが、極値問題において関数計算の占める割合は大きく、全体の処理速度はコーディング方法に影響される。

さらに、関数 $f(\mathbf{x})$ だけを与える関数の場合は、通常傾斜ベクトル $\mathbf{g}$ を差分により近似するため、桁落ちなどを避ける工夫も必要である。また、関数 $f(\mathbf{x})$ の定義において、可能なかぎり変数ベクトル $\mathbf{x}$ の大きさが同程度となるように平衡化することが望ましい。

収束判定値と極小値 $f(\mathbf{x}^*)$ の精度

極小化のアルゴリズムでは、極小点 $\mathbf{x}^*$ において関数 $f(\mathbf{x})$ の傾斜ベクトル $\mathbf{g}(\mathbf{x}^*)$ は、 $\mathbf{g}(\mathbf{x}^*) = \mathbf{0}$ となることを前提としている。すなわち反復公式は、関数 $f(\mathbf{x})$ を極小点 $\mathbf{x}^*$ の近傍では、2 次関数

$$f(\mathbf{x}^* + \delta \mathbf{x}) \approx f(\mathbf{x}^*) + \frac{1}{2} \delta \mathbf{x}^T \mathbf{B} \delta \mathbf{x} \quad (29)$$

として近似している。

(29) は $f(\mathbf{x})$ が適度に平衡化されているとき、 $\mathbf{x}$ に対して ε 程度の摂動を与えた場合に、関数 $f(\mathbf{x})$ は ε^2 程度の変化率を示すことを意味している。

ところで、C-SSL II での収束判定条件は、反復ベクトルを $\mathbf{x}_k$ と表すと、

$$\|\mathbf{x}_{k+1} - \mathbf{x}_k\|_\infty \leq \max(1.0, \|\mathbf{x}_k\|_\infty) \cdot \varepsilon \quad (30)$$

を満たした場合に $\mathbf{x}_{k+1}$ を極小点 $\mathbf{x}^*$ とみなす。ここで ε は収束判定値である。

したがって、極小値 $f(\mathbf{x}^*)$ を丸め誤差程度まで正しく求めるためには、収束判定値 ε を $\varepsilon = \mu^{1/2}$ (μ は丸め誤差の単位)程度に与えるのが適当である。

1.10.4 制約なし関数二乗和の極小化(非線型最小二乗解)

この問題では, n 変数の m 個の実関数 $f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_m(\mathbf{x})$ と初期ベクトル $\mathbf{x}_0$ が与えられたとき,

$$F(\mathbf{x}) = \sum_{i=1}^m \{f_i(\mathbf{x})\}^2$$

が極小値をとるベクトル (極小点) $\mathbf{x}^*$ と, その関数値 $F(\mathbf{x}^*)$ を求める. ここで $\mathbf{x}$ は, $\mathbf{x} = (x_1, x_2, \dots, x_n)^T$ なるベクトルである. ただし, $m \geq n$ である.

すべての関数 $f_i(\mathbf{x})$ が $\mathbf{x}$ の 1 次式の場合は, 線型最小二乗解の問題であり, そのための解法は“1.7.7 最小二乗解”で解説されている (例えば c_dla1). 一方, $f_i(\mathbf{x})$ のいずれかが $\mathbf{x}$ の非線型関数である場合に, この節で述べる関数を利用する.

さて, $\mathbf{x}^*$ の近似ベクトル $\mathbf{x}_k$ を $\Delta\mathbf{x}$ だけ変化させると, $F(\mathbf{x}_k + \Delta\mathbf{x})$ は (31) により近似される.

$$\begin{aligned} F(\mathbf{x}_k + \Delta\mathbf{x}_k) &= \mathbf{f}^T(\mathbf{x}_k + \Delta\mathbf{x}_k) \mathbf{f}(\mathbf{x}_k + \Delta\mathbf{x}_k) \\ &\approx \mathbf{f}^T(\mathbf{x}_k) \mathbf{f}(\mathbf{x}_k) + 2\mathbf{f}^T(\mathbf{x}_k) \mathbf{J}_k \Delta\mathbf{x}_k + \Delta\mathbf{x}_k^T \mathbf{J}_k^T \mathbf{J}_k \Delta\mathbf{x}_k \end{aligned} \quad (31)$$

ただし $|F(\mathbf{x}_k)|$ は十分小であると仮定する. ここで, $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_m(\mathbf{x}))^T$ であり, $\mathbf{J}_k$ は $\mathbf{x}_k$ における $\mathbf{f}(\mathbf{x})$ のヤコビアン行列である.

この $F(\mathbf{x}_k + \Delta\mathbf{x}_k)$ を最小化する $\Delta\mathbf{x}_k$ は (31) の右辺を微分して得られる連立 1 次方程式 (32) の解として与えられる.

$$\mathbf{J}_k^T \mathbf{J}_k \Delta\mathbf{x}_k = -\mathbf{J}_k^T \mathbf{f}(\mathbf{x}_k) \quad (32)$$

(32) 式は正規方程式と呼ばれ, この $\Delta\mathbf{x}_k$ に基づく反復法が Newton-Gauss 法である. この方法は, $\Delta\mathbf{x}_k$ の方向としては $F(\mathbf{x})$ の降下方向ではあるが, $\Delta\mathbf{x}_k$ 自体は発散する可能性がある.

一方, $F(\mathbf{x})$ の $\mathbf{x}_k$ における傾斜ベクトル $\nabla F(\mathbf{x}_k)$ は

$$\nabla F(\mathbf{x}_k) = 2\mathbf{J}_k^T \mathbf{f}(\mathbf{x}_k)$$

で与えられる. $-\nabla F(\mathbf{x}_k)$ は $F(\mathbf{x})$ の $\mathbf{x}_k$ における最急降下方向であり,

$$\Delta\mathbf{x}_k = -\nabla F(\mathbf{x}_k) \quad (33)$$

とするのが, 最急降下法である. (33) の $\Delta\mathbf{x}_k$ は, $F(\mathbf{x})$ の減少を最も確実に保証するが, 反復を繰り返すとジグザグ運動を始めるという欠点がある.

Levenberg-Marquardt 法を基本とした公式

Newton-Gauss 法と最急降下法の折衷案として, Levenberg, Marquardt 及び Morrison は, 次の方程式より $\Delta\mathbf{x}_k$ を決定することを提案した.

$$\{\mathbf{J}_k^T \mathbf{J}_k + v_k^2 \mathbf{I}\} \Delta\mathbf{x}_k = -\mathbf{J}_k^T \mathbf{f}(\mathbf{x}_k) \quad (34)$$

ここで v_k は正数 (Marquardt 数と呼ぶ) である.

(34) により決定される $\Delta\mathbf{x}_k$ は, 明らかに v_k の値に依存する. すなわち, $\Delta\mathbf{x}_k$ の方向は, $v_k \rightarrow 0$ とすれば, Newton-Gauss 法方向となり, $v_k \rightarrow \infty$ とすれば最急降下法方向となる.

C-SSL II では, (34) に基づく反復公式を採用している. ただし, (34) を直接解くのではなく, 数値的な安定性を保つために, (34) と同値な系

$$\begin{bmatrix} \mathbf{J}_k \\ \vdots \\ v_k \mathbf{I} \end{bmatrix} \Delta\mathbf{x}_k = -\begin{bmatrix} \mathbf{f}(\mathbf{x}_k) \\ \vdots \\ \mathbf{0} \end{bmatrix} \quad (35)$$

に対する最小二乗法 (ハウスホルダー法) により求めている. また, v_k の値は反復の状況を観察しながら, 適応的に決定する方法を採用している.

a) ルーチンの種類

利用者が関数 $f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_m(\mathbf{x})$ 以外にヤコビアン行列 $\mathbf{J}$ を解析的に定義できるか否かの条件に応じて, 表 1.16 に示す関数が用意されている.

表 1.16 制約なし関数二乗和の極小化の関数

解析的定義	関数名	備考
$f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_m(\mathbf{x})$	c_dnof1	改訂マルカート法
$f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_m(\mathbf{x}), \mathbf{J}$	c_dnolg1	改訂マルカート法

b) 使用上の注意

初期ベクトル $\mathbf{x}_0$ の与え方

初期ベクトル $\mathbf{x}_0$ は, 可能なかぎり期待する極小点 $\mathbf{x}^*$ の近傍に取ることが望ましい. 関数 $F(\mathbf{x})$ がいくつかの極小点を持つ場合, 初期ベクトルが適切でないと, 期待する極小点 $\mathbf{x}^*$ に収束しない可能性がある. 通常, $\mathbf{x}_0$ は本関数を適用しようとする問題の性質をよく考察した上で設定されるべきであろう.

関数計算プログラム

利用者は, 関数 $\{f_i(\mathbf{x})\}$, 又はヤコビアン行列 $\mathbf{J}$ の値を計算する副プログラムを用意する場合, 効率の良いコーディングをすることが望ましい. C-SSL II ルーチンによる各関数の評価回数は, 手法, 初期ベクトルなどに依存するが, 極値問題において関数計算の占める割合は大きく, 全体の処理速度はコーディング方法に影響される. さらに, 関数 $\{f_i(\mathbf{x})\}$ だけを与えるルーチンの場合, 通常ヤコビアン行列 $\mathbf{J}$ を差分により近似するため, 桁落ちなどを避ける工夫も必要である. また, 関数 $f_i(\mathbf{x})$ の定義において, 可能なかぎり変数ベクトル $\mathbf{x}$ の大きさが同程度となるように平衡化することが望ましい. また, $f_i(\mathbf{x})$ 自体の値もできるだけ同程度になるように平衡化することが望ましい.

収束判定値と極小値 $F(\mathbf{x}^*)$ の精度

極小化のアルゴリズムでは, 極小点 $\mathbf{x}^*$ において,

$$\nabla F(\mathbf{x}^*) = 2\mathbf{J}^T \mathbf{f}(\mathbf{x}^*) = 0$$

となることを前提としている. すなわち反復公式は, 関数 $F(\mathbf{x})$ を極小点 $\mathbf{x}^*$ の近傍では, 2 次関数

$$F(\mathbf{x}^* + \delta \mathbf{x}) \approx F(\mathbf{x}^*) + \delta \mathbf{x}^T \mathbf{J}^T \mathbf{J} \delta \mathbf{x} \quad (36)$$

として近似している.

(36)は, $F(\mathbf{x})$ が適度に平衡化されているとき, $\mathbf{x}$ に対して ε 程度の摂動を与えた場合に, 関数 $F(\mathbf{x})$ は ε^2 程度の変化率を示すことを意味している.

ところで, C-SSL II での収束判定条件は, 反復ベクトルを $\mathbf{x}_k$ と表すと,

$$\begin{aligned} F(\mathbf{x}_{k+1}) &< F(\mathbf{x}_k) \\ \|\mathbf{x}_{k+1} - \mathbf{x}_k\|_2 &\leq \max(1.0, \|\mathbf{x}_k\|_2) \cdot \varepsilon \end{aligned}$$

を満たした場合に $\mathbf{x}_{k+1}$ を極小点 $\mathbf{x}^*$ とみなす. したがって, 極小値 $F(\mathbf{x}^*)$ を丸め誤差程度まで正しく求めるためには, 収束判定値 ε を $\varepsilon \approx \mu^{1/2}$ (μ は丸め誤差の単位) 程度に与えるのが適当である. C-SSL II での標準収束判定値は, $2 \cdot \sqrt{u}$ を採用している.

1.10.5 線形計画問題

いくつかの変数に関する一次不等式と一次等式の組合せによって表される制約条件のもとで, 与えられた一次関数を最小(又は最大)にする変数の値及びその関数の最小値(又は最大値)を求める問題を, 線形計画問題という.

次の形の線形計画問題を標準形という.

“条件 :

$$\mathbf{Ax} = \mathbf{d} \quad (37)$$

$$\mathbf{x} \geq \mathbf{0} \quad (38)$$

のもとで、線形の目的関数

$$z = \mathbf{c}^T \mathbf{x} + c_0$$

を最小にせよ”

ここで、 $\mathbf{A}$ は $m \times n$ なる係数行列で $\text{rank } \mathbf{A} = m \leq n$ とする。また、 $\mathbf{x} = (x_1, x_2, \dots, x_n)^T$ は変数ベクトル、 $\mathbf{d} = (d_1, d_2, \dots, d_m)^T$ は定数ベクトル、 $\mathbf{c} = (c_1, c_2, \dots, c_n)^T$ は係数ベクトル、 c_0 は定数項である。

いま、 $\mathbf{A}$ の第 j 列 (x_j の係数ベクトル) を $\mathbf{a}_j$ で表すことにする。 $\mathbf{A}$ の m 個の列 $\mathbf{a}_{k_1}, \mathbf{a}_{k_2}, \dots, \mathbf{a}_{k_m}$ が 1 次独立であるとき対応する変数の組 $(x_{k_1}, x_{k_2}, \dots, x_{k_m})$ を基底といい、 x_{k_i} を (i 番目の) 基底変数という。非基底変数 (基底変数でない変数) の値をすべて 0 とおいて得られる (37) の解を基底解といい、その中で、(38) も満たすものを可能基底解という。(37), (38) を満たす解が存在するならば、可能基底解が存在し、目的関数の値を最小にする最適解が存在するならば、可能基底解の中に最適解が存在することが示されている。(線形計画法の基本定理)。一つの可能基底解から出発して、基底変数の一つずつ交換することにより、 z がより小さくなるような可能基底解を順次求めていって、最適解を求める方法をシンプレックス法という。

シンプレックス法の反復計算において、交換すべき基底変数の決定に必要な係数や定数項だけを、基底行列 $\mathbf{B} = [\mathbf{a}_{k_1}, \mathbf{a}_{k_2}, \dots, \mathbf{a}_{k_m}]$ の逆行列と、はじめの係数 $\mathbf{A}$, $\mathbf{c}$ 及び定数項 $\mathbf{d}$ から計算する方法を改訂シンプレックス法という。C-SSL II では、この改訂シンプレックス法を採用した関数 `c_dlpss1` を用意している。

制約条件の中に不等式条件が含まれているときは、変数を付加して等式条件に直す。例えば、

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n \leq d_1$$

は、

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n + x_{n+1} = d_1, \quad x_{n+1} \geq 0$$

とする。また、

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n \geq d_2$$

は、

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n - x_{n+2} = d_2, \quad x_{n+2} \geq 0$$

とする。 x_{n+1} や x_{n+2} のように不等式を等式に直すために付加する非負変数をスラック変数という。最大化問題は、目的関数に -1 をかけることにより最小化問題に変えられる。

可能基底解が得られていないときは、解法を 2 段階に分けて、第 1 段階で可能基底解を求め、可能基底解が得られたら、第 2 段階で最適解を求める。このような方法を 2 段階法という。第 1 段階では、次の問題の最適解を求める。

“条件 :

$$\mathbf{Ax} + \mathbf{A}^{(a)} \mathbf{x}^{(a)} = \mathbf{d},$$

$$\mathbf{x} \geq \mathbf{0}, \mathbf{x}^{(a)} \geq \mathbf{0}$$

のもとで、

$$z_1 = \sum_{i=1}^m x_i^{(a)}$$

を最小にせよ”

ここで、 $\mathbf{x}^{(a)}$ は $\mathbf{x}^{(a)} = (x_1^{(a)}, x_2^{(a)}, \dots, x_m^{(a)})^T$ なる m 次元ベクトル、 $\mathbf{A}^{(a)}$ は $\mathbf{A}^{(a)} = (a_{ii}^{(a)})$ なる m 次の対角行列で、 $d_i \geq 0$ のとき $a_{ii}^{(a)} = 1$, $d_i < 0$ のとき $a_{ii}^{(a)} = -1$ である。 $x_i^{(a)}$ を人為変数という。

この最適解が得られたとき、 $z_1 > 0$ の場合は、(37), (38) を満たす $\mathbf{x}$ は存在しない。

一方、 $z_1 = 0$ すなわち $\mathbf{x}^{(a)} = \mathbf{0}$ であれば、与えられた問題の可能基底解が得られているので、第 2 段階に進む。ただし、 $\text{rank } \mathbf{A} < m$ であって、(37) を満たす $\mathbf{x}$ が存在するときは、 $r = \text{rank } \mathbf{A}$ とおくと、 $(m-r)$ 個の条件式はむだである (r 個の条件式が成り立てば、他は必ず成り立つ)。第 1 段階の最適解は、むだな条件式を除いたあとの可能基底解になっている。この場合基底変数の中に $(m-r)$ 個、人為変数がのこっているが、それらに対応する条件式 (i 番目の基底変数は i 番目の条件式に対応している) がむだな条件式である。

1.10.6 非線形計画問題(制約付き多変数関数の極小化)

この問題では, n 変数の実関数 $f(\mathbf{x})$ と初期ベクトル $\mathbf{x}_0$ が与えられたとき, 制約条件

$$c_i(\mathbf{x}) = 0, i = 1, 2, \dots, m_1 \quad (39)$$

$$c_i(\mathbf{x}) \geq 0, i = m_1 + 1, \dots, m_1 + m_2 \quad (40)$$

のもとで, $f(\mathbf{x})$ の極小値を与える $\mathbf{x}^*$ とその関数値 $f(\mathbf{x}^*)$ を求める. ここでは $\mathbf{x}$ は, $\mathbf{x} = (x_1, x_2, \dots, x_n)^T$ なるベクトル, m_1 及び m_2 はそれぞれ等式制約, 不等式制約の個数である.

この制約付き極小化の解法は, 基本的には制約なし極小化の解法に対して, (39), (40)の条件に基づく制約を付加したものである. すなわち, 極小点の近似点 $\mathbf{x}_k$ における $f(\mathbf{x})$ の 2 次近似

$$f(\mathbf{x}) \approx f(\mathbf{x}_k) + \mathbf{y}^T \mathbf{g}_k + \frac{1}{2} \mathbf{y}^T \mathbf{B} \mathbf{y}$$

ここで $\mathbf{y} = \mathbf{x} - \mathbf{x}_k$, $\mathbf{B}$ はヘシアン行列

に基づく極小化において, (39), (40) の 1 次近似よりなる制約

$$c_i(\mathbf{x}_k) + \mathbf{y}^T \nabla c_i(\mathbf{x}_k) = 0, \quad i = 1, 2, \dots, m_1$$

$$c_i(\mathbf{x}_k) + \mathbf{y}^T \nabla c_i(\mathbf{x}_k) \geq 0,$$

$$i = m_1 + 1, \dots, m_1 + m_2$$

∇c_i は c_i の勾配ベクトル.

を満たす解を求めるわけである. これは, $\mathbf{y}$ に関する 2 次計画問題である.

C-SSL II では, 反復の過程で逐次 2 次計画問題を解く方法を採用した c_dnlpg1 を用意している.

1.11 補間・近似

1.11.1 概要

この節では, 以下の問題を取り上げる.

補間

離散点 $x_1, x_2, \dots, x_n$ ($x_1 < x_2 < \dots < x_n$) に対して関数値 $y_i = f(x_i)$, $i = 1, \dots, n$ が与えられたとき(場合によっては $y'_i = f'(x_i)$ も与えられたとき), それらの点を通るような, $f(x)$ の近似式(以後, 補間式と言う)を求める. あるいはその補間式を使って, 与えられた離散点以外の点 $x = v$ における $f(x)$ の近似値(以後, 補間値という)を求める.

最小二乗近似

離散点 $x_1, x_2, \dots, x_n$ ($x_1 < x_2 < \dots < x_n$) に対して観測値, y_i , $i = 1, \dots, n$ が与えられたとき,

$$\sum_{i=1}^n w(x_i) \{y_i - \bar{y}_m(x_i)\}^2, \quad w(x_i) \geq 0$$

を最小にする近似式 $\bar{y}_m(x)$ を求める. ここで $w(x)$ は重み関数であり, $\bar{y}_m(x)$ は m 次の多項式である. この問題は, y_i が実験データであつてかつ y_i の観測誤差の大きさが個々にまちまちである場合に適用される.

平滑化

離散点 $x_1, x_2, \dots, x_n$ ($x_1 < x_2 < \dots < x_n$) に対して観測値 y_i , $i = 1, \dots, n$ が与えられたとき $\{y_i\}$ に含まれる観測誤差を消して, 真の関数をより正しく表すような新しい点列 $\{\bar{y}_i\}$ を求める. 以後, この操作を平滑化と言い, $\bar{y}_i$ (あ

るいは $\{\bar{y}_i\}$) を y_i (あるいは $\{y_i\}$) に対する平滑値, $|y_i - \bar{y}_i|$ を y_i に対する平滑化の程度, 平滑化に用いる近似式を平滑化式と言う.

級数

任意の有限区間で滑らかな関数 $f(x)$ の関数値計算が複雑で手間がかかったり, $f(x)$ の微分積分が解析的にできない場合, $f(x)$ を一旦チェビシェフ級数に展開すればよい. チェビシェフ級数展開の特徴は, 次のとおりである.

- 級数の収束が速い.
- 項別の微分積分が容易である.
- 高速フーリエ変換の手法を用いるので能率がよく, 数値的に安定である.

そこで, 要求精度に応じて項数 n と, n 個のチェビシェフ展開係数を求める. さらに, 得られた級数の項別微分, 積分によって $f(x)$ の導関数, 不定積分を級数の形で求める. また, これらの級数を求和することにより, 関数値, 微係数, 定積分を求める.

関数 $f(x)$ が任意の周期を持つ滑らかな周期関数である場合は, 三角級数に展開できる. ここでは, 偶関数, 奇関数を要求精度に応じて, それぞれ cosine 級数, sine 級数に展開する.

1.11.2 スプライン (Spline) 関数について

補間及び平滑化の分野をはじめ, 数値微積分においても, “スプライン関数”と呼ばれる関数が有力な道具として使われる. それで以下に, スプライン関数の定義及びその表現などについて述べる.

a) Spline 関数の定義

ここで述べる Spline 関数とは次のような関数である. すなわち,

$$a = x_0 < x_1 < \dots < x_n = b$$

を実軸上の区間 $[a, b]$ の分割とし, $D \equiv d/dx$ として,

$$\begin{aligned} & \text{a. 各部分区間 } (x_i, x_{i+1}) \text{ で } D^k S(x) = 0 \\ & \text{b. } S(x) \in C^{k-2}[a, b] \end{aligned} \quad (41)$$

を満たすような関数 $S(x)$ を $k-1$ 次の Spline 関数という. また, 点列 $\{x_i\}$ を Spline 関数の節点という.

(41) から分かるように, $S(x)$ は各部分区間 (x_i, x_{i+1}) において別々に定義される高々 $k-1$ 次の多項式であって, しかも全区間 $[a, b]$ においては $k-2$ 次までの導関数が連続であるような関数である.

b) Spline 関数の表現 – その 1

任意の $S(x)$ は次の式で表現される. すなわち, $a_j, j = 0, 1, \dots, k-1$ 及び $b_i, i = 1, 2, \dots, n-1$ を任意の定数として,

$$\begin{cases} S(x) = p(x) + \sum_{i=1}^{n-1} b_i (x - x_i)_+^{k-1} \\ \text{ここで,} \\ p(x) = \sum_{j=0}^{k-1} a_j (x - x_0)^j \end{cases} \quad (42)$$

である. ただし記号 $(x - x_i)_+^{k-1}$ は (43) を意味する.

$$(x - x_i)_+^{k-1} = \begin{cases} (x - x_i)^{k-1}, & x \geq x_i \\ 0, & x < x_i \end{cases} \quad (43)$$

式 (42) が (41) を満たすことを示そう.

式 (42) において, x を x_0 から右へ動かしてみる.

$x_0 \leq x < x_1$ では, $S(x) = p(x)$ であるから, $S(x)$ は $k-1$ 次の多項式である.

$x_1 \leq x < x_2$ では, $S(x) = p(x) + b_1(x - x_1)^{k-1}$ であるから $S(x)$ はやはり $k-1$ 次の多項式である.

一般に, $x_i \leq x < x_{i+1}$ では,

$$S(x) = p(x) + \sum_{r=1}^i b_r (x - x_r)^{k-1}$$

であるから $S(x)$ は各区間で別々に定義される $k-1$ 次の多項式であることは容易に分かる.

式(42)から,

$$\begin{aligned} \frac{d^l}{dx^l} S(x) &= S^{(l)}(x) \\ &= \sum_{j=1}^{k-1} j(j-1)\cdots(j-l+1) a_j (x - x_0)^{j-l} + \sum_{i=1}^{n-1} (k-1)(k-2)\cdots(k-l) b_i (x - x_i)_+^{k-1-l} \end{aligned}$$

である. したがって, $S(x)$ の x_i における左側 l 階微係数, 右側 l 階微係数は各々,

$$\begin{aligned} \lim_{\varepsilon \rightarrow 0} S^{(l)}(x_i - \varepsilon) &= \sum_{j=1}^{k-1} j(j-1)\cdots(j-l+1) a_j (x_i - x_0)^{j-l} + \sum_{r=1}^{i-1} (k-1)(k-2)\cdots(k-l) b_r (x_i - x_r)^{k-1-l} \\ \lim_{\varepsilon \rightarrow 0} S^{(l)}(x_i + \varepsilon) &= \sum_{j=1}^{k-1} j(j-1)\cdots(j-l+1) a_j (x_i - x_0)^{j-l} + \sum_{r=1}^{i-1} (k-1)(k-2)\cdots(k-l) b_r (x_i - x_r)^{k-1-l} \\ &\quad + \lim_{\varepsilon \rightarrow 0} (k-1)(k-2)\cdots(k-l) b_i (x_i + \varepsilon - x_i)^{k-1-l} \end{aligned}$$

となる. このことから,

$$\lim_{\varepsilon \rightarrow 0} S^{(l)}(x_i + \varepsilon) - \lim_{\varepsilon \rightarrow 0} S^{(l)}(x_i - \varepsilon) = \lim_{\varepsilon \rightarrow 0} (k-1)(k-2)\cdots(k-l) b_i \varepsilon^{k-1-l} \quad (44)$$

である.

$l=0, 1, \dots, k-2$ に対しては, (44)の右辺は 0 であるから,

$$\lim_{\varepsilon \rightarrow 0} S^{(l)}(x_i + \varepsilon) = \lim_{\varepsilon \rightarrow 0} S^{(l)}(x_i - \varepsilon)$$

であり, $S^{(l)}(x)$ は $x = x_i$ で連続である.

一方, $l = k-1$ に対しては, (44)の右辺は $(k-1)(k-2)\cdots(k-l) b_i$ であり, 一般に $b_i \neq 0$ であるから,

$$\lim_{\varepsilon \rightarrow 0} S^{(k-1)}(x_i + \varepsilon) \neq \lim_{\varepsilon \rightarrow 0} S^{(k-1)}(x_i - \varepsilon)$$

となり $S(x)$ の $k-1$ 次導関数は $x = x_i$ で不連続となる. もしすべての $b_i, i=1, 2, \dots, n-1$ が 0 のときは, $S(x)$ は全区間 $[a, b]$ で $k-1$ 次導関数までが連続になり, この場合 (42) から明らかなように全区間 $[a, b]$ で $S(x) = p(x)$ である. これは $[a, b]$ で定義される $k-1$ 次多項式の $x = x_0$ を中心とするべき級数展開にほかならない. このことから, $[a, b]$ で定義される任意の多項式は, Spline 関数の特殊形であることが分かる.

式 (42) は Spline 関数の打ち切りべき関数 (truncated power function) による表現と呼ぶ. この表現は Spline 関数の性質を直観的に表す式ではあるが, $(x - x_i)^{k-1}$ などは絶対値が大きくなりがちであるから, この表現は一般に数値的に不安定である.

c) Spline 関数の表現 – その 2 (B-spline の導入)

(42)に対して, 以下に定義される B-spline による表現には数値的な不都合が少ない. 今, 点列 $\{t_r\}$ を,

$$t_{-k+1} \leq t_{-k+2} \leq \cdots \leq t_{-1} \leq t_0 = x_0 < t_1 = x_1 < \cdots < t_n = x_n \leq t_{n+1} \leq t_{n+2} \leq \cdots \leq t_{n+k-1} \quad (45)$$

のようにとる(図 1.19 参照).

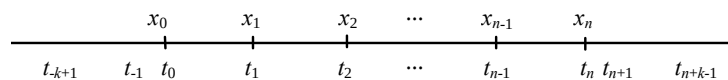


図 1.19 点列 $\{t_r\}$ の採り方

また, $g_k(t; x)$ を x を引数とする t の関数として,

$$g_k(t; x) = (t - x)_+^{k-1} = \begin{cases} (t - x)^{k-1}, & t \geq x \\ 0, & t < x \end{cases} \quad (46)$$

とする(図 1.20 参照).

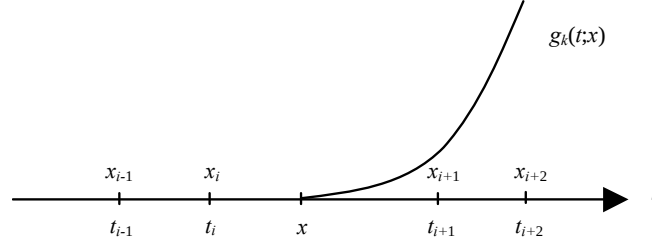


図 1.20 $g_k(t; x)$

このとき (46) の $t = t_j, t_{j+1}, \dots, t_{j+k}$ における k 階差分商の定数倍,

$$N_{j,k}(x) = (t_{j+k} - t_j) g_k[t_j, t_{j+1}, \dots, t_{j+k}; x] \quad (47)$$

を, $k-1$ 次の正規化された B-spline(又は単に B-spline)と呼ぶ.

ここで B-spline $N_{j,k}(x)$ の性質について述べる. 今, $t_j, t_{j+1}, \dots, t_{j+k}$ を固定して x を動かして見る. $x \leq t_j$ のとき $N_{j,k}(x)$ は t についての $k-1$ 次多項式の k 階差分商の定数倍であるから 0 である. また, $t_{j+k} \leq x$ のとき, $N_{j,k}(x)$ は恒等的に 0 であるような関数の k 階差分商の定数倍であるから 0 である.

$t_j < x < t_{j+k}$ のときは $N_{j,k}(x)$ は 0 ではない. これをまとめると,

$$N_{j,k}(x) \begin{cases} \neq 0, & t_j < x < t_{j+k} \\ = 0, & x \leq t_j \text{ 又は } t_{j+k} \leq x \end{cases} \quad (48)$$

である(実際には $t_j < x < t_{j+k}$ において $0 < N_{j,k}(x) \leq 1$ である). 次に x を固定して j を変えてみる. 今, $t_i = x_i < x < x_{i+1} = t_{i+1}$ とする. このとき上と同じような考えで,

$$N_{j,k}(x) \begin{cases} \neq 0, & i - k + 1 \leq j \leq i \\ = 0, & j \leq i - k \text{ 又は } i + 1 \leq j \end{cases} \quad (49)$$

がいえる.

以上の (48), (49) を B-spline の局所性と呼ぶことにする.

ところで, この B-spline $N_{j,k}(x)$ は (47) より,

$$N_{j,k}(x) = (t_{j+k} - t_j) \sum_{r=j}^{j+k} \frac{(t_r - x)_+^{k-1}}{(t_r - t_j) \cdots (t_r - t_{r-1})(t_r - t_{r+1}) \cdots (t_r - t_{j+k})} \quad (50)$$

となる. したがって, $N_{j,k}(x)$ は各区間 (x_i, x_{i+1}) で別々に定義される $k-1$ 次の多項式であり, $k-2$ 次導関数までが連続である. $N_{j,k}(x)$ のこの性質に基づいて, (41) を満たす任意の Spline 関数 $S(x)$ は, $c_j, j = -k+1, -k+2, \dots, n-1$ を定数として,

$$S(x) = \sum_{j=-k+1}^{n-1} c_j N_{j,k}(x) \quad (51)$$

と表現できることが証明される.

差分商(divided difference)の定義

関数 $f(t)$ の $t = t_j, t_{j+1}, \dots, t_{j+k}$ ($t_j \leq t_{j+1} \leq \dots \leq t_{j+k}$) における k 階差分商 $f[t_j, t_{j+1}, \dots, t_{j+k}]$ とは次のように再帰的に定義される量である.

$$f[t_j, t_{j+1}, \dots, t_{j+k}] = \begin{cases} \frac{f[t_{j+1}, t_{j+2}, \dots, t_{j+k}] - f[t_j, t_{j+1}, \dots, t_{j+k-1}]}{t_{j+k} - t_j} & : t_{j+k} \neq t_j \\ \frac{1}{k!} \frac{d^k f(t_j)}{dt^k} & : t_{j+k} = t_j \end{cases}$$

ただし, $f[t_j] = f(t_j)$

また, $t_r \neq t_s : r \neq s$ のとき, 以下の2式が成り立つ.

$$f[t_j, t_{j+1}, \dots, t_{j+k}] = \sum_{r=j}^{j+k} \frac{f(t_r)}{(t_r - t_j) \dots (t_r - t_{r-1}) (t_r - t_{r+1}) \dots (t_r - t_{j+k})}$$

$$f[t_j, t_{j+1}, \dots, \overbrace{t_i, t_i, \dots, t_i}^{r+1}, t_{i+1}, \dots, t_{j+k}] = \frac{1}{r!} \frac{\partial^r}{\partial t_i^r} f[t_j, t_{j+1}, \dots, t_{i-1}, t_i, t_{i+1}, \dots, t_{j+k}]$$

d) Spline 関数の計算法

$k-1$ 次の Spline 関数

$$S(x) = \sum_{j=-k+1}^{n-1} c_j N_{j,k}(x) \quad (52)$$

が与えられたとき $x \in [x_i, x_{i+1})$ における関数値, 微係数, あるいは積分

$$\int_{x_0}^x S(y) dy$$

などの計算のしかたについて述べる.

関数値の計算

$x \in [x_i, x_{i+1})$ における $S(x)$ の値は, $N_{j,k}(x)$ を計算することにより計算される. ところが, $N_{j,k}(x)$ の局所性 (49) により非零のものだけを計算すればよい.

$N_{j,k}(x)$ の計算は, 次の漸化式に基づく.

$$N_{r,s}(x) = \frac{x - t_r}{t_{r+s-1} - t_r} N_{r,s-1}(x) + \frac{t_{r+s} - x}{t_{r+s} - t_{r+1}} N_{r+1,s-1}(x) \quad (53)$$

ここで,

$$\begin{aligned} N_{r,1}(x) &= (t_{r+1} - t_r) g_1[t_r, t_{r+1}; x] \\ &= (t_{r+1} - t_r) \frac{g_1(t_{r+1}; x) - g_1(t_r; x)}{t_{r+1} - t_r} \\ &= (t_{r+1} - x)_+^0 - (t_r - x)_+^0 \\ &= \begin{cases} 1, & r = i \\ 0, & r \neq i \end{cases} \end{aligned} \quad (54)$$

そこで (53), (54) を使って $s = 2, 3, \dots, k, r = i - s + 1, i - s + 2, \dots, i$ に対して適用すれば, 図 1. 21 に挙げたすべての $N_{r,s}(x)$ を計算することができ, 最右列のものが $S(x)$ の計算に使われる.

微係数, 積分の計算

$$\frac{d^l}{dx^l} S(x) = S^{(l)}(x) = \sum_{j=k+1}^{n-1} c_j N_{j,k}^{(l)}(x) \quad (55)$$

$$\frac{\partial^l}{\partial x^l} g_k(t; x) = \frac{(-1)^l (k-1)!}{(k-1-l)!} (t-x)_+^{k-1-l} \quad (56)$$
$$d_k(t; x) = (t - x)_+^{k-1-l} = \begin{cases} (t - x)^{k-1-l}, & t \geq x \\ 0 & , \quad t \leq x \end{cases}$$
$$D_{i,k}(x) = d_k[t_i, t_{i+1}, \dots, t_{i+k}; x] \quad (57)$$
$$\begin{aligned} D_{r+1}(x) &= \begin{cases} 1/(x_{i+1} - x_i), & r = i \\ 0, & r \neq i \end{cases} \\ D_{r,s}(x) &= \frac{D_{r+1,s-1}(x) - D_{r,s-1}(x)}{t_{r+s} - t_s}, \quad 2 \leq s \leq l+1 \\ D_{r,s}(x) &= \frac{(x - t_r)D_{r,s-1}(x) + (t_{r+s} - x)D_{r+1,s-1}(x)}{t_{r+s} - t_r}, \quad l+2 \leq s \leq k \end{aligned} \quad (58)$$
$$N_{j,k}^{(l)}(x) = (t_{j+k} - t_j) \frac{(-1)^l (k-1)!}{(k-1-l)!} D_{j,k}(x)$$
$$I = \int_{x_0}^x S(y) dy = \sum_{j=-k+1}^{n-1} c_j \int_{x_0}^x N_{j,k}(y) dy \quad (59)$$
$$\int g_k(t; x) dx = -\frac{1}{k}(t-x)_+^k$$
$$I_{i,k}(x) = e_k[t_i, t_{i+1}, \dots, t_{i+k}; x] \quad (60)$$

54

$$I_{r,1}(x) = \begin{cases} 0 & , \quad r \leq i-1 \\ (x_{i+1} - x)/(x_{i+1} - x_i), & r = i \\ 1 & , \quad r \geq i+1 \end{cases} \quad (61)$$

$$I_{r,s}(x) = \frac{(x - t_r)I_{r,s-1}(x) + (t_{r+s} - x)I_{r+1,s-1}(x)}{t_{r+s} - t_r}$$

この (61) を $s = 2, 3, \dots, k, r = i - s + 1, i - s + 2, \dots, i$ について適用すれば $I_{j,k}(x)$ は図 1.22 に挙げるように最右列として求まる。

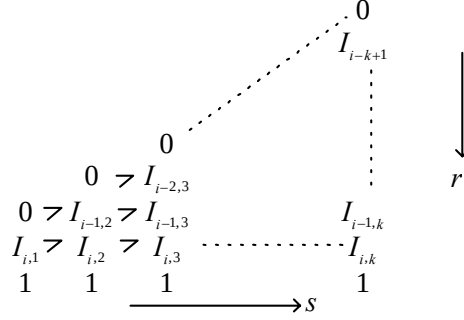


図 1.22 $I_{r,s}(x)$ の計算, ただし $x \in [x_b, x_{i+1}]$

$$\begin{aligned} \int_{x_0}^x N_{j,k}(y) dy &= -\frac{(t_{i+k} - t_j)}{k} [I_{j,k}(x) - I_{j,k}(x_0)] \\ &= \frac{(t_{j+k} - t_j)}{k} [I_{j,k}(x_0) - I_{j,k}(x)] \end{aligned}$$

よって (59) より,

$$\begin{aligned} I &= \int_{x_0}^x S(y) dy = \frac{1}{k} \sum_{j=-k+1}^{n-1} c_j (t_{j+k} - t_j) [I_{j,k}(x_0) - I_{j,k}(x)] \\ &= \frac{1}{k} \left\{ \sum_{j=-k+1}^0 c_j (t_{j+k} - t_j) I_{j,k}(x_0) - \sum_{j=i-k+1}^i c_j (t_{j+k} - t_j) I_{j,k}(x) + \sum_{j=1}^i c_j (t_{j+k} - t_j) \right\} \end{aligned} \quad (62)$$

として得られる。

以上, Spline 関数 $S(x)$ の関数値, 微係数及び積分の計算法を述べたが, 今まで, (52) の係数 c_j は既知と仮定してきた。 c_j の決定は $S(x)$ が補間式ならば補間条件から, あるいは, $S(x)$ が平滑化式ならば最小二乗近似的条件から決められる。例えば, (52) では, $n + k - 1$ 個の係数 c_j ($-k + 1 \leq j \leq n - 1$) を含むので (52) が補間式として用いられるなら, $n + k - 1$ 個の補間条件を与えることにより決定される。仮に図 1.19において, $x = x_0, x_1, \dots, x_n$ という $n + 1$ 個の点で関数値が与えられているとしても, 補間条件より $n + k - 1$ 個の c_j を決めようとする場合, あと $(n + k - 1) - (n + 1) = k - 2$ 個の点で関数が与えられるか, それとも $k - 2$ 個の別な条件(例えば $S(x)$ の微係数に対する条件)が付加されなくてはならない。これについてはあとの“1.11.3 補間”で述べる。

C-SSL II では, (52) で表現される Spline 関数を, 補間, 数値微分, 数値積分, 及び最小二乗近似による平滑化に応用する。

2 変数 Spline 関数の定義, 表現及び計算法

2 変数 Spline 関数は, 先に述べた 1 変数の場合の拡張として定義される。

まず, xy 平面上の閉領域 $R = \{(x, y) | a \leq x \leq b, c \leq y \leq d\}$ を考え, この上に分割 (63) に従って, 格子点 (x_b, y_j) $0 \leq i \leq m, 0 \leq j \leq n$ を作る。

$$\begin{aligned} a &= x_0 < x_1 < \dots < x_m = b \\ c &= y_0 < y_1 < \dots < y_n = d \end{aligned} \quad (63)$$

このとき, $D_x = \partial/\partial x, D_y = \partial/\partial y$ として

$$\text{各部分開領域 } R_{i,j} = \{(x,y) | x_i < x < x_{i+1}, y_j < y < y_{j+1}\} \text{ 上で } D_x^k S(x,y) = D_y^k S(x,y) = 0 \quad (64)$$

$$S(x,y) \in C^{k-2,k-2}[R] \quad (65)$$

を満たすような関数 $S(x,y)$ を双 $k-1$ 次の 2 変数 Spline 関数という。(64)は, $S(x,y)$ が各 $R_{i,j}$ 上で x, y についての多項式であり, x についても y についても高々 $k-1$ 次であることを表し, 更に (65)は, $S(x,y)$ が R 全体では, $\lambda = 0, 1, \dots, k-2, \mu = 0, 1, \dots, k-2$ とするとき,

$$\frac{\partial^{\lambda+\mu}}{\partial x^\lambda \partial y^\mu} S(x,y)$$

が存在してしかも連続であることを表す.

今, 点列 $\{s_i\}, \{t_j\}$ を次のようにとる.

$$s_{-k+1} \leq s_{-k+2} \leq \dots \leq s_{-1} \leq s_0 = x_0 < s_1 = x_1 < \dots < s_m = x_m \leq s_{m+1} \leq \dots \leq s_{m+k-1} \\ t_{-k+1} \leq t_{-k+2} \leq \dots \leq t_{-1} \leq t_0 = y_0 < t_1 = y_1 < \dots < t_n = y_n \leq t_{n+1} \leq \dots \leq t_{n+k-1}$$

これに基づいて, x 方向, y 方向の $k-1$ 次の B-spline を, 1 変数の場合と同様に

$$N_{\alpha,k}(x) = (s_{\alpha+k} - s_\alpha) g_k[s_\alpha, s_{\alpha+1}, \dots, s_{\alpha+k}; x] \\ N_{\beta,k}(y) = (t_{\beta+k} - t_\beta) g_k[t_\beta, t_{\beta+1}, \dots, t_{\beta+k}; y]$$

と定義する. このとき上で定義した双 $k-1$ 次の 2 変数 Spline 関数は, $c_{\alpha,\beta}$ を任意定数として

$$S(x,y) = \sum_{\beta=-k+1}^{n-1} \sum_{\alpha=-k+1}^{m-1} c_{\alpha,\beta} N_{\alpha,k}(x) N_{\beta,k}(y) \quad (66)$$

という形で書き表すことができる.

$S(x,y)$ の関数値, 偏微分あるいは不定積分の計算は, (66) の表現を使えば, 1 変数の場合の計算を単に応用するだけでできる.

$$S^{(\lambda,\mu)}(x,y) = \frac{\partial^{\lambda+\mu}}{\partial x^\lambda \partial y^\mu} S(x,y) \\ = \sum_{\beta=-k+1}^{n-1} \sum_{\alpha=-k+1}^{m-1} c_{\alpha,\beta} N_{\alpha,k}^{(\lambda)}(x) N_{\beta,k}^{(\mu)}(y) \quad (67)$$

と書けるから, 関数値, 偏微分の計算は $N_{\alpha,k}^{(\lambda)}(x)$, $N_{\beta,k}^{(\mu)}(y)$ を別々に計算することによりなされ, これは, 先に述べた 1 変数の場合の手法を適用するだけでよい.

次に, $S(x,y)$ を y については μ 階偏微分し, x については積分した量

$$S^{(-1,\mu)}(x,y) = \int_{x_0}^x \frac{\partial^\mu S(x,y)}{\partial y^\mu} dx \quad (68)$$

を考える. ここで, 偏微分と積分の順序を逆にしても値は同じであることに注意する.

(66) を使って (68) の右辺を書き直せば,

$$\sum_{\alpha=-k+1}^{m-1} \left\{ \sum_{\beta=-k+1}^{n-1} c_{\alpha,\beta} N_{\beta,k}^{(\mu)}(y) \right\} \cdot \int_{x_0}^x N_{\alpha,k}(x) dx = \sum_{\alpha=-k+1}^{m-1} c_\alpha \int_{x_0}^x N_{\alpha,k}(x) dx \quad (69)$$

$$\text{ただし, } c_\alpha = \sum_{\beta=-k+1}^{n-1} c_{\alpha,\beta} N_{\beta,k}^{(\mu)}(y)$$

となるが, これは, 先に挙げた (60) に類似する. したがって (69) を計算するには, まず c_α を計算し, 続いて, 1 変数の積分の手法を使って (69) を計算すればよい.

$S^{(-1,\mu)}(x,y)$ のほか,

$$S^{(\lambda, -1)}(x, y) = \int_{y_0}^y \frac{\partial^\lambda S(x, y)}{\partial x^\lambda} dy$$

$$S^{(-1, -1)}(x, y) = \int_{y_0}^y dy \int_{x_0}^x S(x, y) dx$$

の計算も、1 変数の場合の微分、積分の手法を、 x, y 別々に適用することにより処理できる。

1.11.3 補間

補間を行う場合の一般の手順は、与えられた標本点、 (x_i, y_i) を通るある近似関数—多項式、区分的多項式、その他—を作り、それを計算することから成る。

近似関数が多項式の場合、それはラグランジュ補間多項式、あるいはエルミート補間多項式(関数値のほかに微係数の情報も使う)と呼ばれている。C-SSL II のエイトケン・ラグランジュ補間及びエイトケン・エルミート補間は、これに類する手法であるが、特徴として反復的に補間多項式の次数を上げていくことにより、最も適当と思われる補間値を見出そうとするものである。

一方、近似関数が区分的多項式の場合は、単なる多項式を用いたのでは補間がうまくいかないときに用いることができる。C-SSL II はこれに属するものとして、準エルミート補間、Spline 関数による補間を用意している。

Spline 関数による補間式は、“1.11.2 スプライン (Spline) 関数について”で述べた性質に、補間の条件—与えられた標本点を通ること—を加えた spline 関数のことであり、実はそれは、何らかの付加的条件を加えることによりいろいろな補間式を作り出すことができる。C-SSL II では四つのタイプの spline 補間式を考え各々関数を用意している。

Spline 関数の表現形式は、“1.11.2 スプライン (Spline) 関数について”で述べた B-spline の安定性を重要視し、もっぱらそれによる表現を用いている。

B-spline による補間

B-spline を用いる補間の関数は目的により、次の二つに分けられる。

1. 補間値(あるいは微分値, 積分値)を求める関数
2. 補間式を求める関数

1. はすべて、2. で求めた補間式を利用するので、利用者は手順として、2. を先に呼び出し、続いて 1. を呼び出すことになる。

B-spline を用いる補間式として、C-SSL II ではいくつか用意する。すなわち、離散点を $x_i, i = 1, 2, \dots, n$ として、 $m (=2l-1, l \geq 2)$ 次の B-spline 補間式 $S(x)$ を求める場合、 $S(x)$ の境界条件の有無、あるいは境界条件の内容によって、次に示す四つの“タイプ”に分類し、各々関数を用意する。

- タイプ I $S^{(j)}(x_1), S^{(j)}(x_n), j = 1, 2, \dots, l-1$ を(利用者の)指定した値にとる。
- タイプ II $S^{(j)}(x_1), S^{(j)}(x_n), j = l, l+1, \dots, 2l-2$ を指定した値にとる。
- タイプ III 境界条件なし。
- タイプ IV $S^{(j)}(x_1) = S^{(j)}(x_n), j = 0, 1, \dots, 2l-2$ を満たす。このタイプは周期関数の補間に適している。

これら四つのタイプの使分けは、利用者の持っている元の関数についての情報量による。通常は、タイプ III の関数を利用すればよい。

2 次元補間の場合は、(66) で表現される 2 変数 Spline 関数 $S(x, y)$ を補間式として用いる。このとき、 x 方向、 y 方向に対して独立にタイプを考えることもできるが、C-SSL II では両方向に対してタイプ I 又はタイプ III を使った補間式を用意する。

B-spline 補間式により補間値を求める際、その補間式の次数 m の選び方が問題となる。次数は 3 又は 5 程度がよい。倍精度では、元の関数がおとなしく、関数値が高精度で与えられているならばもう少し次数を高くしてもよい。しかし 15 を超えるとよくないといわれている。

表 1.17 に補間の関数を示す。

表 1.17 補間の関数

目的	関数名	手法	備考
補間値	c_daklag	エイトケン・ラグランジュ補間	微係数不要
	c_dakher	エイトケン・エルミート補間	微係数要
	c_dbif1	B-spline 補間式 (I) の利用	タイプ I
	c_dbif2	B-spline 補間式 (II) の利用	タイプ II
	c_dbif3	B-spline 補間式 (III) の利用	タイプ III
	c_dbif4	B-spline 補間式 (IV) の利用	タイプ IV
	c_dbifd1	B-spline 2 次元補間式 (I-I) の利用	タイプ I-I
	c_dbifd3	B-spline 2 次元補間式 (III-III) の利用	タイプ III-III
	c_dakmid	2 次元準エルミート補間	
補間式	c_dakmin	準エルミート補間式	
	c_dbic1	B-spline 補間式 (I)	タイプ I
	c_dbic2	B-spline 補間式 (II)	タイプ II
	c_dbic3	B-spline 補間式 (III)	タイプ III
	c_dbic4	B-spline 補間式 (IV)	タイプ IV
	c_dbicd1	B-spline 2 次元補間式 (I-I)	タイプ I-I
	c_dbicd3	B-spline 2 次元補間式 (III-III)	タイプ III-III

準エルミート補間

これは、spline 補間と同じように、区分的な多項式による補間である。ただ、準エルミート補間式は、高階導関数の連続性を spline 補間式ほどには強く要求しない点が異なっている。

準エルミート補間式の唯一の特徴は、離散点がどのような分布で与えられても離散点の間に不自然な屈曲点が見れないということであり、このため「曲線又は曲面のあてはめ」に適している。曲線(又は曲面)のあてはめでは、どんな離散点に対しても、あたかも熟練した製図工が手で描くような曲線を作り出すことに重点が置かれる。準エルミート補間式は、まさにこの目的のための補間式である。

しかし、高精度な補間値、又は微分値、積分値を得るためには、むしろ、B-spline による補間を用いた方がよい。

1.11.4 近似

最小二乗近似多項式の関数を表 1.18 に示す。B-spline による最小二乗近似は、“1.11.5 平滑化”で扱われている。

表 1.18 近似の関数

目的	関数名	手法	備考
最小二乗近似多項式	c_dlesq1	離散点に関して直交する多項式を導入する方法	近似式の次数は関数内で決定する

1.11.5 平滑化

平滑化に関して用意されている関数を表 1.19 に示す。

表 1.19 平滑化の関数

目的	関数名	手法	備考
平滑値	c_dsmle1	局部的最小二乗近似多項式の適用	等間隔離散点
	c_dsmle2	局部的最小二乗近似多項式の適用	不等間隔離散点
	c_dbsf1	B-spline 平滑化式の利用	不等間隔離散点
	c_dbsfd1	B-spline 2 次元平滑化式の利用	不等間隔格子点
平滑化式	c_dbsc1	B-spline 平滑化式(固定節点)	不等間隔離散点
	c_dbsc2	B-spline 平滑化式(節点追加方式)	
	c_dbscd2	B-spline 2 次元平滑化式(節点追加方式)	不等間隔格子点

c_dsmle1, c_dsmle2 は平滑化を求めるのに、観測値の全体にわたるただ一つの最小二乗近似多項式をあてはめて行う代わり、局所的に、それぞれ異なる最小二乗近似多項式をあてはめることにより行うものである。

しかし、一般目的のためには、B-spline による関数を利用したほうがよい。B-spline による平滑化は、1 次元の場合は、(51)、2 次元の場合は (66) で表現される Spline 関数を平滑化式として用いる。その際、係数 c_j 若しくは $c_{\alpha,\beta}$ は最小二乗法により決定される。平滑値は、求まった平滑化式を評価することにより得られ、そのための関数も用意されている。

ところで、B-spline 平滑化式を求める関数には、節点列の決め方の違いにより二つのタイプがある。一つは、節点列を利用者が与えるタイプ(固定節点)であり、他のひとつは、節点列を関数が適応的に決定するタイプ(節点追加方式)である。前者の関数は、節点列の与え方についての経験を要する。したがって、通常は、後者の関数を使えばよい。

1.11.6 級数

関数のチェビシェフ級数展開、級数の求和、導関数、不定積分のために、表 1.20 に示す関数が用意されている。

表 1.20 チェビシェフ級数の関数

目的	関数名	手法	備考
級数展開	c_dfcheb	高速 cosine 変換	項数は(2 のべき) + 1
級数の求和	c_decheb	後退漸化式	
級数の導関数	c_dgcheb	チェビシェフ多項式の微分公式	
級数の不定積分	c_dicheb	チェビシェフ多項式の積分公式	

一方、周期関数に対しては、cosine 級数展開、sine 級数展開と、それぞれの級数の求和を求めるために、表 1.21 に示す関数が用意されている。

表 1.21 cosine, sine 級数の関数

目的	関数名	手法	備考
cosine 級数展開	c_dfcosf	高速 cosine 変換	偶関数
cosine 級数の求和	c_decosp	後退漸化式	偶関数
sine 級数展開	c_dfsinf	高速 sine 変換	奇関数
sine 級数の求和	c_desinp	後退漸化式	奇関数

1.12 変換

1.12.1 概要

この節では、離散型フーリエ変換及びラプラス変換を取り上げる。

離散型フーリエ変換には、変換の対象とするデータの性質に応じて関数が用意されている。データの性質とは、次の内容を指す。

- 実データか複素データか。
- 実データでも偶関数か奇関数か。

1.12.2 離散型実フーリエ変換

データが実データである場合に、(70)なる変換又は、(71)なる逆変換に相当した変換を行う関数が用意されている。

$$\left. \begin{aligned} a_k &= \frac{2}{n} \sum_{j=0}^{n-1} x_j \cos \frac{2\pi kj}{n}, \quad k = 0, 1, \dots, \frac{n}{2} \\ b_k &= \frac{2}{n} \sum_{j=0}^{n-1} x_j \sin \frac{2\pi kj}{n}, \quad k = 1, 2, \dots, \frac{n}{2} - 1 \end{aligned} \right\} \quad (70)$$

$$x_j = \frac{1}{2} a_0 + \sum_{k=1}^{n/2-1} \left(a_k \cos \frac{2\pi kj}{n} + b_k \sin \frac{2\pi kj}{n} \right) + \frac{1}{2} a_{n/2} \cos \pi j, \quad j = 0, 1, \dots, n-1 \quad (71)$$

a_k, b_k を離散型フーリエ係数という。

周期 2π の実数値関数 $x(t)$ のフーリエ係数を定義する積分

$$\left. \begin{aligned} \frac{1}{\pi} \int_0^{2\pi} x(t) \cos kt \, dt \\ \frac{1}{\pi} \int_0^{2\pi} x(t) \sin kt \, dt \end{aligned} \right\} \quad (72)$$

に対し、閉区間 $[0, 2\pi]$ を n 等分し $x(t)$ を

$$x_j = x\left(\frac{2\pi}{n} j\right), \quad j = 0, 1, \dots, n-1,$$

と標本化し台形公式を適用したものが変換 (70) である。特に $x(t)$ が $n/2 - 1$ 次の三角多項式ならば (70) は、積分 (72) の正確な数値積分公式となる。すなわち、離散型フーリエ係数と、本来のフーリエ係数とは一致する。更に $x(t)$ が偶関数あるいは奇関数ならば、それらの性質を利用した離散型 cosine 変換、sine 変換が用意されている。

1.12.3 離散型 cosine 変換

偶関数 $x(t)$ に対して 2 種類の変換を行う関数が用意されている。一方は、閉区間 $[0, \pi]$ の端点を標本点に含む方法であり、他方は含まない。

離散型 cosine 変換(台形公式)

偶関数 $x(t)$ を閉区間 $[0, \pi]$ 上で,

$$x_j = x\left(\frac{\pi}{n}j\right), \quad j=0, 1, \dots, n$$

と標本化し, (73) なる変換又は, (74) なる逆変換に相当した変換を行う.

$$a_k = \frac{2}{n} \sum_{j=0}^n x_j \cos \frac{\pi}{n} kj, \quad k=0, 1, \dots, n \quad (73)$$

$$x_j = \sum_{k=0}^{n-1} a_k \cos \frac{\pi}{n} kj, \quad j=0, 1, \dots, n \quad (74)$$

ここで Σ'' は, 初項と末項を $1/2$ 倍して和をとることを意味する.

偶関数 $x(t)$ のフーリエ係数を定義する積分

$$\frac{2}{\pi} \int_0^{\pi} x(t) \cos kt \, dt \quad (75)$$

に対し, 閉区間 $[0, \pi]$ を n 等分して $x(t)$ を $x_j = x\left(\frac{\pi}{n}j\right)$, $j=0, 1, \dots, n$ と標本化し, 台形公式を適用したものが変換 (73) である.

離散型 **cosine** 変換(中点公式)

偶関数 $x(t)$ を開区間 $(0, \pi)$ 上で

$$x_{j+1/2} = x\left(\frac{\pi}{n}\left(j + \frac{1}{2}\right)\right), \quad j=0, 1, \dots, n-1$$

と標本化し, (76) なる変換又は, (77) なる逆変換に相当した変換を行う.

$$a_k = \frac{2}{n} \sum_{j=0}^{n-1} x_{j+\frac{1}{2}} \cos \frac{\pi}{n} k\left(j + \frac{1}{2}\right), \quad k=0, 1, \dots, n-1 \quad (76)$$

$$x_{j+\frac{1}{2}} = \sum_{k=0}^{n-1} a_k \cos \frac{\pi}{n} k\left(j + \frac{1}{2}\right), \quad j=0, 1, \dots, n-1 \quad (77)$$

ここで Σ' は, 初項だけを $1/2$ 倍して和をとることを意味する.

積分 (75) に n 項の中点公式を適用したものが, 変換 (76) である.

1.12.4 離散型 sine 変換

関数 $x(t)$ が奇関数の場合, 2 種類の変換を行う関数が用意されている. 離散型 cosine 変換の場合と同様に, 一方は台形公式に基づく変換であり, 他方は中点公式による方法である.

離散型 **sine** 変換(台形公式)

奇関数 $x(t)$ を閉区間 $[0, \pi]$ で

$$x_j = x\left(\frac{\pi}{n}j\right), \quad j=1, \dots, n-1$$

標本化し (78) なる変換又は, (79) なる逆変換に相当した変換を行う.

$$b_k = \frac{2}{n} \sum_{j=1}^{n-1} x_j \sin \frac{\pi}{n} kj, \quad k=1, 2, \dots, n-1 \quad (78)$$

$$x_j = \sum_{k=1}^{n-1} b_k \sin \frac{\pi}{n} kj, \quad j=1, 2, \dots, n-1 \quad (79)$$

奇関数 $x(t)$ のフーリエ係数を定義する積分

$$\frac{2}{\pi} \int_0^{\pi} x(t) \sin kt \, dt \quad (80)$$

に対し、閉区間 $[0, \pi]$ を n 等分して $x(t)$ を $x_j = x\left(\frac{\pi}{n}j\right)$, $j=1, 2, \dots, n-1$ と標本化し、台形公式を適用したものが変換 (78) である。

離散型 **sine** 変換(中点公式)

奇関数 $x(t)$ を开区間 $(0, \pi)$ 上で

$$x_{j+1/2} = x\left(\frac{\pi}{n}\left(j + \frac{1}{2}\right)\right), \quad j=0, 1, \dots, n-1$$

と標本化し、(81) なる変換又は、(82) なる逆変換に相当した変換を行う。

$$b_k = \frac{2}{n} \sum_{j=0}^{n-1} x_{j+1/2} \sin \frac{\pi}{n} k \left(j + \frac{1}{2}\right), \quad k=1, 2, \dots, n \quad (81)$$

$$x_{j+1/2} = \sum_{k=1}^{n-1} b_k \sin \frac{\pi}{n} k \left(j + \frac{1}{2}\right) + \frac{1}{2} b_n \sin \pi \left(j + \frac{1}{2}\right), \quad j=0, 1, \dots, n-1 \quad (82)$$

積分 (80) に n 項の中点公式を適用したものが、変換 (81) である。

1.12.5 離散型複素フーリエ変換

データが複素データある場合には、(83) なる変換又は(84) なる逆変換に相当した変換を行う関数が用意されている。

$$\alpha_k = \frac{1}{n} \sum_{j=0}^{n-1} x_j \exp\left(-2\pi i \frac{jk}{n}\right), \quad k=0, 1, \dots, n-1 \quad (83)$$

$$x_j = \sum_{k=0}^{n-1} \alpha_k \exp\left(2\pi i \frac{jk}{n}\right), \quad j=0, 1, \dots, n-1 \quad (84)$$

周期 2π の複素数値関数 $x(t)$ のフーリエ係数を定義する積分

$$\frac{1}{2\pi} \int_0^{2\pi} x(t) \exp(-ikt) \, dt \quad (85)$$

に対し、閉区間 $[0, 2\pi]$ を n 等分し $x(t)$ を

$$x_j = x\left(\frac{2\pi}{n}j\right), \quad j=0, 1, \dots, n,$$

と標本化し、 $x_0 = x_n$ を考慮して台形公式を適用したものが、変換 (83) である。

これらの離散型フーリエ変換は、高速変換手法(FFT) により行うので処理速度が速い。

C-SSL II では次のようなデータ数の変換を行うことができる。

- データ数が 2 の巾である (2 基底).
- データ数が 2, 3, 5 の巾の積として表せる (2, 3 及び 5 の混合基底).
- データ数が $\{2, 3, 4, 5, 7, 8, 9, 16\}$ の中で互いに疎な因子の積で表せる (混合基底).

さらに、ルーチンによっては多重変換、多次元変換も可能である。

表 1. 22 に、データの性質に応じた関数名を示す。

表 1.22 離散型フーリエ変換の関数名

変換の種類		基底	関数名	備考
cosine 変換	台形公式	2	c_dvcos1	
	中点公式		c_dfcosm	
	—	任意	c_dvmcst	1 次元, 多重
sine 変換	台形公式	2	c_dvsin1	
	中点公式		c_dfsinm	
	—	任意	c_dvmsnt	1 次元, 多重
実変換		2, 3 又は 5	c_dvmrft	多重, 多次元
		2, 3 又は 5	c_dvsrft	1 次元, 多重
		2	c_dvrft1	
			c_dvrft2	メモリ節約型
		混合基底	c_dvrpf3	3 次元
		混合基底	c_dvmrf2	1 次元, 多重, 多次元
複素変換		任意	c_dvmcft	多重, 多次元
		2	c_dvcft1	
		2	c_dvcft2	メモリ節約型
		2	c_dvcft3	一定間隔を持つデータ
		混合基底	c_dvcpf1	1 次元
		混合基底	c_dvcpf3	3 次元
		混合基底	c_dvcfm1	1 次元
		混合基底	c_dvmcf2	1 次元, 多重, 多次元

a) 使用上の注意

標本数(項数)

変換の対象となるデータの標本数 n は, 関数 $x(t)$ の性質(データの性質)に応じて, その定義される意味が異なるので注意すること. すなわち, n は,

- cosine, sine 変換の場合, 半周期分 ($[0, \pi]$ 又は $(0, \pi)$) の標本に相当する.
- 実変換, 複素変換の場合, 1 周期分 ($[0, 2\pi]$) の標本に相当する.

実変換と cosine, sine 変換との使い分け

関数 $x(t)$ が一般の実数関数の場合は, 実変換の関数が利用できるが, $x(t)$ に対してあらかじめ偶関数か奇関数かの判別がつく場合には, cosine, sine 変換の関数を利用することが望ましい(処理速度は, 実変換による場合の約半分である). なお, $x(t)$ が実数値関数でも, $x(t) + x(-t)$ 及び $x(t) - x(-t)$ は, それぞれ偶関数, 奇関数となるので, cosine 変換, sine 変換に分解して計算することができる.

実変換と複素変換によるフーリエ係数の関連

実変換 (cosine, sine 変換を含む) によるフーリエ係数 $\{a_k\}$, $\{b_k\}$ と複素変換によるフーリエ係数, $\{\alpha_k\}$ との間には, 次の関係式が成り立つ.

$$\begin{aligned}
 a_0 &= 2\alpha_0, \quad a_{n/2} = 2\alpha_{n/2}, \\
 a_k &= (\alpha_k + \alpha_{n-k}), \quad k=1, 2, \dots, n/2-1 \\
 b_k &= i(\alpha_k - \alpha_{n-k}), \quad k=1, 2, \dots, n/2-1
 \end{aligned} \tag{86}$$

ここで n は, 1 周期 $[0, 2\pi]$ の等分数を表す.

この関係に基づき、利用者は必要に応じて、実変換と複素変換の関数を併用することができる、ただし、正規化及びデータの並びについては注意すること。

三角関数表について

cosine, sine 変換の場合、処理効率を高めるために、変換で必要な三角関数表を関数内部で作成している。関数表は引数 tab に出力され、繰り返し変換を行う場合の効率を高めるためにも、有効に利用できる。各々の変換に対して、台形公式と中点公式に基づく二つの関数が用意されているが、三角関数表の大きさは、前者の方が小さいので若干能率が良い。

正規化について

変換を行うすべての関数は、変換結果に対する正規化は行っていない。利用者は、必要に応じて任意に正規化することができる。

1.12.6 ラプラス変換

関数 $f(t)$ のラプラス変換と逆変換は次式で定義される。

$$F(s) = \int_0^{\infty} f(t) e^{-st} dt \quad (87)$$

$$f(t) = \frac{1}{2\pi i} \int_{\gamma-i\infty}^{\gamma+i\infty} F(s) e^{st} ds \quad (88)$$

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

$f(t)$ をラプラス変換における原関数, $F(s)$ を像関数という。像関数 $F(s)$ に次の条件を仮定する。

$$\left. \begin{array}{l} \text{(i) } \operatorname{Re}(s) > \gamma_0 \text{ で } F(s) \text{ は正則} \\ \text{(ii) } \operatorname{Re}(s) > \gamma_0 \text{ で } \lim_{|s| \rightarrow \infty} F(s) = 0 \\ \text{(iii) } \operatorname{Re}(s) > \gamma_0 \text{ で } F^*(s) = F(s^*) \end{array} \right\} \quad (89)$$

$\operatorname{Re}(s)$ は $\operatorname{Re}(s)$ の実数部, $F^*(s)$ は $F(s)$ の共役複素数を表す。条件 (i) は常に成り立ち、条件 (ii) は $f(t)$ が超関数の場合を除き満足される。条件 (iii) は $f(t)$ が実関数ということである。用意されている関数は式 (88) の逆変換を数値的に行うものである。以下、その手法の概要を示す。

a) 変換の計算式

まず簡単のため、 $\gamma_0 \leq 0$ の場合を考える。すなわち、 $F(s)$ は $\operatorname{Re}(s) > 0$ で正則でありかつ (88) の積分は、 $\gamma > 0$ なる任意の実数 γ に対して存在するものとする。

$$e^s = \lim_{\sigma_0 \rightarrow \infty} e^{\sigma_0} / [2 \cosh(\sigma_0 - s)]$$

の関係に着目して (88) の e^{st} を適当な σ_0 を選ぶことにより、

$$E_{ec}(st, \sigma_0) \equiv e^{\sigma_0} / [2 \cosh(\sigma_0 - st)]$$

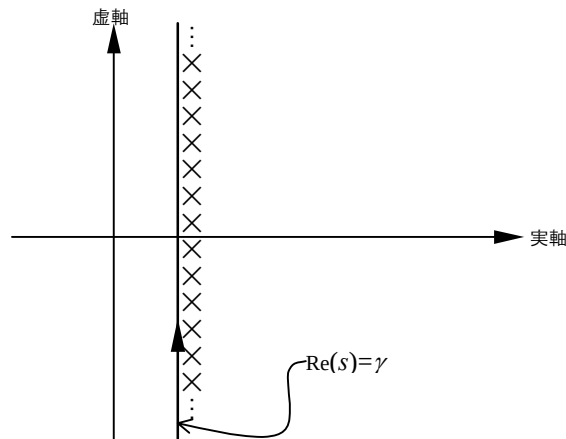
で近似する。関数 $E_{ec}(st, \sigma_0)$ の特徴は、 $\operatorname{Re}(s) = \sigma_0/t$ の直線上に無数の極を持つことで、極の配置を図 1.23 に × 印で示す。また、このことを式で陽に書くと

$$E_{ec}(st, \sigma_0) = \frac{e^{\sigma_0}}{2t} \sum_{n=-\infty}^{\infty} \frac{(-1)^n i}{s - [\sigma_0 + i(n-0.5)\pi]/t}$$

となる。このとき、原関数 $f(t)$ の σ_0 近似 $f(t, \sigma_0)$ は

$$f(t, \sigma_0) \equiv \frac{1}{2\pi i} \int_{r-i\infty}^{r+i\infty} F(s) E_{ec}(st, \sigma_0) ds \quad (90)$$

となる。ただし γ は $\gamma_0 < \gamma < \sigma_0/t$ を満たすものとする。右辺の積分は $E_{ec}(st, \sigma_0)$ の極のまわりの積分に変換できることが証明できる。

図 1.23 $E_{ec}(st, \sigma_0)$ の極

$F(s)$ は $\text{Re}(s) > 0$ で正則であるからコーシーの積分公式より

$$\begin{aligned} f(t, \sigma_0) &= \frac{e^{\sigma_0}}{2t} \sum_{n=-\infty}^n (-1)^{n+1} i F\left(\frac{\sigma_0 + i(n-0.5)\pi}{t}\right) \\ &= \frac{e^{\sigma_0}}{t} \sum_{n=1}^{\infty} (-1)^n \text{Im} \left[F\left(\frac{\sigma_0 + i(n-0.5)\pi}{t}\right) \right] \end{aligned} \quad (91)$$

となる. $\text{Im}[\cdot]$ は $[\cdot]$ の虚数部を表す. ただし (90) の条件 $\gamma_0 < \gamma < \sigma_0/t$ において, もし $\gamma_0 > 0$ であると, ある $t(0 < t < \infty)$ でこの条件が満足できなくなる. したがって (91) を $0 < t < \infty$ に対して適用するときは $\gamma_0 \leq 0$ の条件が必要である.

$$f(t, \sigma_0) = f(t) - e^{-2\sigma_0} \cdot f(3t) + e^{-4\sigma_0} \cdot f(5t) - \dots \quad (92)$$

となる. したがって, $\sigma_0 \gg 1$ のとき $f(t, \sigma_0)$ は $f(t)$ の良い近似であることを示し, また, (92) は近似誤差の評価式として使える.

数値計算を行う場合, 原理的には (91) を適当な項数 N で打ち切ればよいが, 単純な打ち切りでは実用的でない. したがって, 個々の場合に応じて実用的な計算式を工夫する必要がある. 用意されている関数では, かなり一般性のある Euler 変換を利用している.

$$F_n \equiv (-1)^n \text{Im} \left[F\left(\frac{\sigma_0 + i(n-0.5)\pi}{t}\right) \right] \quad (93)$$

と定義すると, Euler 変換は次の条件が成り立つ場合に有効である(文献 [13]).

$$\left. \begin{aligned} \text{(i)} \quad & \text{適当な整数 } k \geq 1 \text{ が存在し, } n \geq k \text{ のとき } F_n \text{ の符号が交代する} \\ \text{(ii)} \quad & n \geq k \text{ のとき } 1/2 \leq F_{n+1} / F_n < 1 \end{aligned} \right\} \quad (94)$$

F_n がこの条件を満足しているとき, (91) の級数を次のように書き換える.

$$\sum_{n=1}^{\infty} F_n = \sum_{n=1}^{k-1} F_n + \sum_{q=0}^p \frac{1}{2^{q+1}} D^q F_k + R_{p+1}(k) \quad (95)$$

ここで $R_p(k)$ は

$$R_p(k) \approx 2^{-p} (D^p F_k + D^p F_{k+1} + D^p F_{k+2} + \dots)$$

で与えられる. $D^p F_k$ は次式で定義される第 p 階差である.

$$D^0 F_k = F_k, \quad D^{p+1} F_k = D^p F_k + D^p F_{k+1} \quad (96)$$

関数では

$$f_N(t, \sigma_0) = \frac{e^{\sigma_0}}{t} \sum_{n=1}^N F_n = \frac{e^{\sigma_0}}{t} \left\{ \sum_{n=1}^{k-1} F_n + \sum_{q=0}^p \frac{D^q F_k}{2^{q+1}} \right\} \quad (97)$$

を用いている。ただし $N = k + p$,

$$\left. \begin{aligned} \sum_{q=0}^p \frac{D^q F_k}{2^{q+1}} &= \frac{1}{2^{p+1}} \sum_{r=0}^p A_{p,r} F_{k+r} \\ A_{p,p} &= 1, A_{p,r-1} = A_{p,r} + \binom{p+1}{r} \end{aligned} \right\} \quad (98)$$

σ_0, k, p の決め方は各関数の箇所ですべて。

$f_N(t, \sigma_0)$ の打ち切り誤差については、次のことが証明されている。 $\phi(n) \equiv F_n$ とすると、もし、 $\phi(x)$ の p 階微分係数 $\phi^{(p)}(x)$ が x の正の値に対して定符号であり、かつ x の増加と共に単調に減少するならば、(例えば $F(s)$ が有理関数の場合)、

$$\begin{aligned} |f(t, \sigma_0) - f_N(t, \sigma_0)| &= \frac{e^{\sigma_0}}{t} |R_{p+1}(k)| \\ &\leq |f_{N+1}(t, \sigma_0) - f_N(t, \sigma_0)| = \frac{e^{\sigma_0}}{t} \left| \frac{1}{2^{p+1}} D^{p+1} F_k \right| \end{aligned} \quad (99)$$

が成り立つ。ここで $f_{N+1}(t, \sigma_0)$ は、(97) において k を 1 つ増やしたものである。上式において $D^{p+1} F_k$ を計算するには、 $f_N(t, \sigma_0)$ の計算に使われる列 $\{F_n; n = k, k+1, \dots, k+p\}$ に加えて F_{k+p+1} を必要とし、関数計算を 1 回余分に行うことになる。これを避けるため、関数では、 $f_N(t, \sigma_0)$ の打ち切り誤差を

$$|f_N(t, \sigma_0) - f_{N-1}(t, \sigma_0)| = \frac{e^{\sigma_0}}{t} \left| \frac{1}{2^{p+1}} D^{p+1} F_{k-1} \right|$$

で代用している。また、関数では、打ち切り誤差を、相対誤差の形、

$$\left| \frac{f_N(t, \sigma_0) - f_{N-1}(t, \sigma_0)}{f_N(t, \sigma_0)} \right| = \left| \frac{\frac{1}{2^{p+1}} D^{p+1} F_{k-1}}{\sum_{n=1}^{k-1} F_n + \frac{1}{2^{p+1}} \sum_{r=0}^p A_{p,r} F_{k+r}} \right|$$

で出力する。

$D^{p+1} F_{k-1}$ は $F_{k-1}, F_k, \dots, F_{k+p}$ の 1 次結合となるが、その係数は 2 項展開係数に等しく、また、 $A_{p,r}$ は (98) より分かるように 2 項展開係数の累和として計算できる。したがって、これらの係数は、いずれもパスカルの三角形から容易に求めることができる。例として、 $p=4$ の場合を図 1.24 に示す。

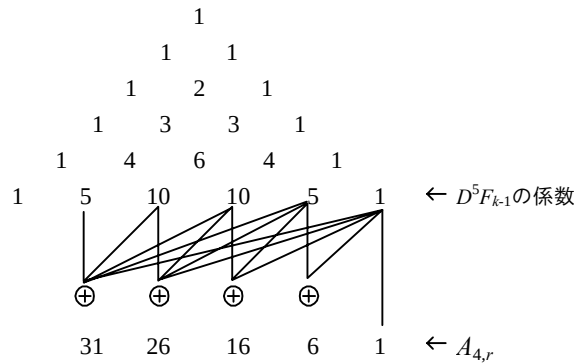


図 1.24 $p=4$ の場合のパスカルの三角形

次に $\gamma_0 > 0$ の場合を考える。このとき $F(s)$ は $\text{Re}(s) > 0$ で正則とはならないので $F(s)$ に対して上記の手法を直接適用することはできない。しかし (88) の積分は、

$$\begin{aligned}
f(t) &= \frac{1}{2\pi i} \int_{r-i\infty}^{r+i\infty} F(s+\gamma_0) e^{(s+\gamma_0)t} ds \\
&= \frac{e^{\gamma_0 t}}{2\pi i} \int_{r-i\infty}^{r+i\infty} G(s) e^{st} ds \\
&= e^{\gamma_0 t} g(t)
\end{aligned} \tag{100}$$

ここで $r>0$, $G(s)=F(s+r_0)$

$$g(t) = \frac{1}{2\pi i} \int_{r-i\infty}^{r+i\infty} G(s) e^{st} ds$$

と書けることに注目する. $G(s)$ は $\text{Re}(s) > 0$ で正則となるので, $g(t)$ を上記の手法で計算することにすれば $f(t)$ は $g(t)$ に $e^{\gamma_0 t}$ を乗ずることにより求めることができる.

b) 有理関数の変換

$F(s)$ が有理関数の場合, 実係数多項式 $Q(s), P(s)$ を用いて

$$F(s) = Q(s) / P(s) \tag{101}$$

と表せる. 収束座標 γ_0 が $\gamma_0 \leq 0$ であるか $\gamma_0 > 0$ であるかの判定は, $P(s)$ が Hurwitz 多項式(すべての零点が複素平面の左半面, すなわち $\text{Re}(s) < 0$ の領域に存在する多項式)かどうかを判定すればよい. 関数で使用した判定法を以下に示す(文献 [53]).

n 次の実係数多項式 $P(s)$ を

$$\begin{aligned}
P(s) &= a_1 s^n + a_2 s^{n-1} + \dots + a_n s + a_{n+1} \\
&= m(s) + n(s) \\
\text{ただし } m(s) &= a_1 s^n + a_3 s^{n-2} + \dots, \quad a_1 \neq 0 \\
n(s) &= a_2 s^{n-1} + a_4 s^{n-3} + \dots
\end{aligned}$$

と分解する. $m(s)$ と $n(s)$ の比を

$$W(s) \equiv m(s)/n(s)$$

と定義し, $W(s)$ を連分数展開する.

$$W(s) = h_1 s + \frac{1}{\left| \frac{1}{h_2 s} + \frac{1}{\left| \frac{1}{h_3 s} + \frac{1}{\left| \frac{1}{h_4 s} + \dots \right.} \right.} \right.}$$

$h_1, h_2, \dots$ がすべて正であれば $P(s)$ は Hurwitz 多項式である.

$F(s)$ が $\text{Re}(s) > 0$ に特異点を持つ場合, 上記の判定法を繰り返し使うことにより $G(s)=F(s+\alpha)$ が $\text{Re}(s) > 0$ で正則となるように $\alpha (> 0)$ を増加させる. この $G(s)$ に対して $g_N(t, \sigma_0)$ を求め, それに $e^{\alpha t}$ を乗ずれば $f_N(t, \sigma_0)$ が得られる.

$F(s)$ が無理関数や超関数の場合, $\text{Re}(s) > 0$ で $F(s)$ が正則かどうかを判定する有効な方法はない. したがって $F(s)$ が一般関数の場合は, 収束座標 γ_0 の決定は利用者に任せられる.

c) 関数の使い分け

ラプラス逆変換のための関数を表 1.23 に示す. c_dlaps1 は $\gamma_0 \leq 0$ であることが既知な場合に, c_dlaps2 は $\gamma_0 > 0$ であることが既知かそれとも γ_0 については不明な場合に, それぞれ有理関数の変換を行うものである. 一方, c_dhrwiz は $\gamma_0 \leq 0$ であるか否かの判定, すなわち, (101) の $P(s)$ が Hurwitz 多項式であるか否かの判定を行い, かつ $\gamma_0 > 0$ であったときは γ_0 の近似値を計算する.

$\gamma_0 > 0$ であることは, 原関数 $f(t)$ が $t \rightarrow \infty$ のとき指数関数的に増大することを意味する. そのような振舞を調べる目的でも c_dhrwiz を使用することができる.

以上の使い分けを流れ図に示すと図 1.25 のようになる.

表 1.23 ラプラス変換の関数

関数の型	関数名	備考
有理関数	c_dlaps1	複素右半平面で正則な有理関数
	c_dlaps2	一般の有理関数
	c_dhrwiz	Hurwitz 多項式の判定
一般関数	c_dlaps3	収束座標 γ_0 の入力が必要.

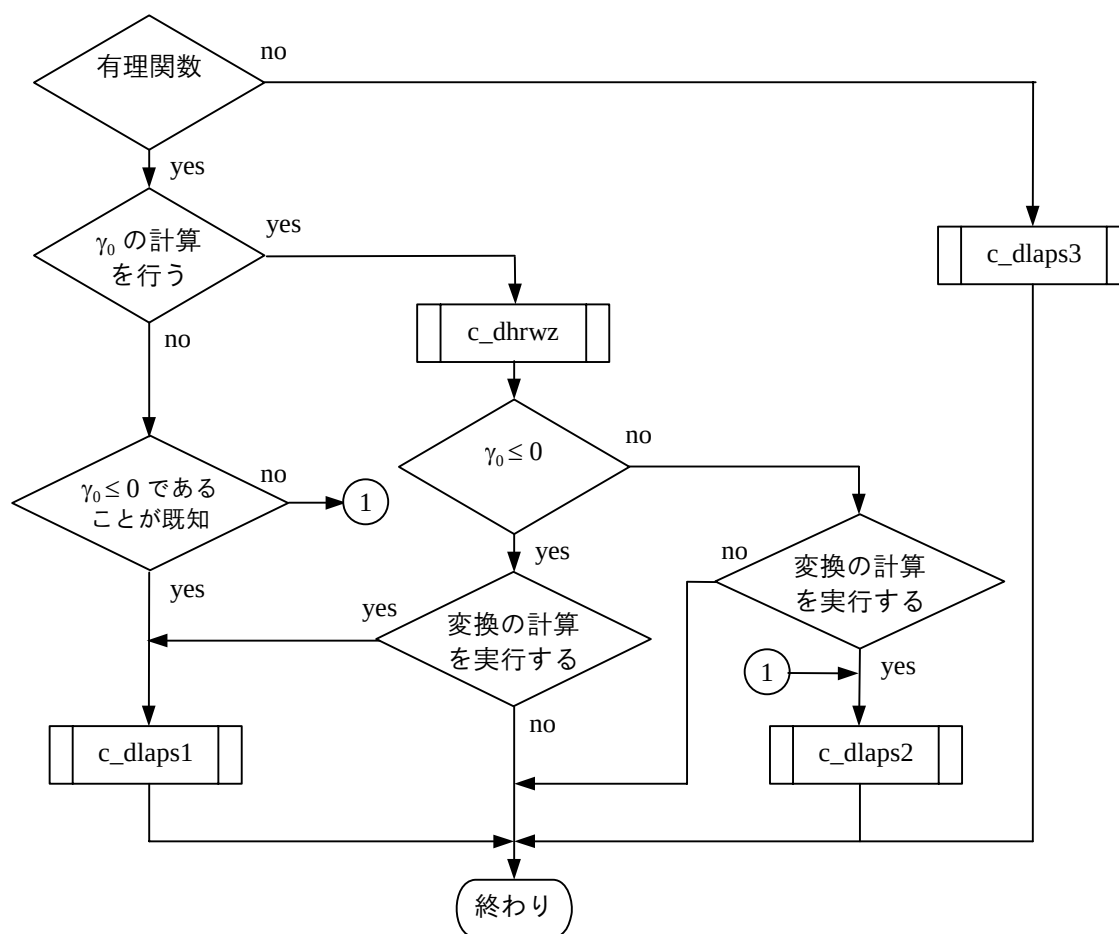


図 1.25 ラプラス変換ルーチンの使い分け

1.13 数値微積分

1.13.1 概要

この節では以下の問題を取り上げる.

数値微分:

離散点 $x_1, x_2, \dots, x_n$ ($x_1 < x_2 < \dots < x_n$) に対して関数値 $y_i = f(x_i)$, $i = 1, \dots, n$ が与えられたとき, 区間 $[x_1, x_n]$ 内の $x = v$ における $f(x)$ の微分値 $f^{(l)}(v)$, $l \geq 1$ を近似する. さらに 2 次元の微分も扱う.

また, 関数 $f(x)$ が与えられたとき, その導関数 $f^{(l)}(x) = d^l f(x)/dx^l$, $l \geq 1$ をチェビシェフ級数で展開する.

数値積分:

離散点 $x_1, x_2, \dots, x_n$ ($x_1 < x_2 < \dots < x_n$) に対して関数値 $y_i = f(x_i)$, $i = 1, \dots, n$ が与えられたとき, 区間 $[x_1, x_n]$ に渡る $f(x)$ の積分値を求める. あるいは関数 $f(x)$ が与えられたとき, 積分値

$$S = \int_a^b f(x) dx$$

を所要の精度まで求める. さらに多重積分も扱う.

1.13.2 数値微分

数値微分では, 問題を次の二つに分けてある.

a) 離散点入力

微分値を求めるには, 標本点 (x_i, y_i) , $i = 1, 2, \dots, n$, に対して適当な補間式をあてはめ, それを微分することにより行う. その際, 補間式の種類はたくさん考えられるが, C-SSL II としては, もっぱら Spline 関数を利用した補間式によっている. しかも, spline 補間式の表現は数値的安定性を重視して, B-spline を使った表現を採用している. spline 関数及び B-spline については“1.11 補間・近似”を参照されたい.

b) 関数入力

関数 $f(x)$ と定義域 $[a, b]$ が与えられたとき, まず, $f(x)$ を所要の精度でチェビシェフ級数展開する. すなわち,

$$f(x) \approx \sum_{k=0}^{n-1} c_k T_k \left(\frac{2x - (b+a)}{b-a} \right)$$

と近似する. 次に, これを項別微分し,

$$f^{(l)}(x) \approx \sum_{k=0}^{n-l-1} c_k^l T_k \left(\frac{2x - (b+a)}{b-a} \right)$$

として導関数をチェビシェフ級数で展開する.

微分値は, 関数が定義される区間の任意の点 $x = v$ において $f^{(l)}(v)$ を評価すること, すなわちチェビシェフ級数を求和することにより求める.

表 1.24 に数値微分の関数を示す.

1.13.3 数値積分

数値積分では問題を次の二つに分けてある。

離散点入力

離散点 $x_1, x_2, \dots, x_n$ ($x_1 < x_2 < \dots < x_n$) に対して関数値 $y_i = f(x_i)$, $i = 1, \dots, n$ が与えられたとき, 積分値

$$S = \int_{x_1}^{x_n} f(x) dx$$

を与えられた関数値 y_i だけを用いて近似する。この場合は, 得られた近似値がどれだけの誤差をもつかを計算できない。また離散点が等間隔か否かにより, ルーチンを使い分けなければならない。

表 1.24 数値微分の関数

目的	関数名	手法	備考
微分値	c_dbif1	B-spline 補間式 (I) の利用	離散点入力
	c_dbif2	B-spline 補間式 (II) の利用	
	c_dbif3	B-spline 補間式 (III) の利用	
	c_dbif4	B-spline 補間式 (IV) の利用	
	c_dbsf1	B-spline 平滑化式の利用	
	c_dbifd1	B-spline 2 次元補間式 (I-I) の利用	離散点入力 2 次元
	c_dbifd3	B-spline 2 次元補間式 (III-III) の利用	
	c_dbsfd1	B-spline 2 次元平滑化式の利用	
導関数と 微分値	c_dfcheb	高速 cosine 変換	関数入力 チェビシェフ 級数展開
	c_dgcheb	後退漸化式	チェビシェフ 級数の導関数
	c_decheb	後退漸化式	チェビシェフ 級数の求和

関数入力

関数 $f(x)$ $[a, b]$ と積分区間 $[a, b]$ が与えられたとき, 積分値

$$S = \int_a^b f(x) dx \quad (102)$$

を所要の精度まで求める。用意されたルーチンは, 積分区間と $f(x)$ の形あるいは性質とにより使い分けるようになっている。

また, (102) のほかに,

$$\int_0^\infty f(x) dx$$

$$\int_{-\infty}^\infty f(x) dx$$

$$\int_a^b dx \int_c^d f(x, y) dy$$

の型の積分も考える。

表 1.25 に数値積分に関して用意されている関数を示す。

表 1. 25 数値積分の関数

目的	関数名	手法	備考
1次元有限区間 (等間隔)	c_dsimp1	シンプソン則	離散点入力
1次元有限区間 (不等間隔)	c_dtrap	台形則	
	c_dbif1	B-spline 補間式 (I) の利用	
	c_dbif2	B-spline 補間式 (II) の利用	
	c_dbif3	B-spline 補間式 (III) の利用	
	c_dbif4	B-spline 補間式 (IV) の利用	
	c_dbsf1	B-spline 平滑化式の利用	
2次元有限区間	c_dbifd1	B-spline 2次元補間式 (I-I) の利用	離散点入力 2次元
	c_dbifd3	B-spline 2次元補間式 (III-III) の利用	
	c_dbsfd1	B-spline 2次元平滑化式の利用	
1次元有限区間	c_daqn9	適応型ニュートン・コーツ9点則	関数入力
	c_daqc8	クレンショー・カーチス型積分法	
	c_daqe	二重指数関数型積分公式	
1次元半無限区間	c_daqeh	二重指数関数型積分公式	
1次元全無限区間	c_daqei	二重指数関数型積分公式	
多次元有限領域	c_daqmc8	クレンショー・カーチス型積分法	多変数関数 入力
多次元有限領域	c_daqme	二重指数関数型積分公式	

a) 数値積分の一般的規約と注意

C-SSL II では、数値積分に関するルーチンを区分するのに、主に次の点に注意した。

- 積分変数の次元数…1次元か2次元か。
- 積分区間…有限区間か全無限区間かそれとも半無限区間か。

各ルーチンはこの区分に従って名前が決められている。例えば「1次元有限区間積分」、「1次元全無限区間積分」という具合である。この区分のほかに、細かい条件、あるいは手法により特徴づけられる場合は () の中に示した。例えば「1次元有限区間積分(不等間隔離散点入力, 台形則)」、「1次元有限区間積分(関数入力, 適応型シンプソン則)」という具合である。

数値積分の手法については離散点入力の場合と関数入力の場合とで決定的な違いがある。離散点入力の場合は、与えられた関数値 $y_i = f(x_i)$, $i = 1, \dots, n$ だけを使って積分値を求めるという制約があるので精度の良い近似値を計算するのは難しい。一方、関数入力の場合は、一般に積分区間のいたる点(特異な場合を除く)における関数値を計算できるので、関数値を多く計算することによって、望む精度まで積分値を計算できるし、また誤差がいくら位かを、計算によって予測することができる。

b) 関数入力 1次元有限区間積分

積分 $\int_a^b f(x)dx$ のためのルーチンを使用する際の注意事項を述べる。

自動積分ルーチン

積分 $\int_a^b f(x)dx$ のためのルーチンとしては、表 1. 25に見られるように c_daqn9, c_daqc8 及び c_daqe の三つがある。

これらのルーチンはすべて自動積分ルーチンである。自動積分ルーチンとは、被積分関数 $f(x)$ 、積分区間 $[a, b]$ 、及び積分値に対する要求精度が与えられたとき、その要求精度を満たす積分値を求める積分ルーチンのことを意味し、その目的のために考案されたアルゴリズムを自動積分法という。

自動積分ルーチンでは一般に、最初、少ない分点(関数を計算する点)から積分値の計算を始めて、そのあと徐々に分点を増やしながら、要求精度を満たすまで積分値を改良していき、そして要求精度が満たされた時点で反復を止め、そのときの積分値を出力して計算を終える。

自動積分ルーチンは、興味深いルーチンが世界中で数多く作られ、これに伴って、各ルーチンの信頼性(計算された積分値が要求精度を満たしていること)と経済性(計算量が少ないこと)に関する比較テストも、多くの研究者によって繰り返されてきた。C-SSL II のルーチンは、これらの成果を十分に反映して作られたものである。

適応型手法

自動積分法の代表として、現在最も広く用いられている積分の計算法である。これは、特定の積分公式(例えばシンプソン則、ニュートン・コーツ9点則、あるいはガウス側など)を言うのではなく、一般に、被積分関数の変化の急なところでは分点を密にとり、反対に緩やかなところでは疎にとるようにして、なるべく関数の挙動に応じて分点の位置と個数を自動的に調整する機能を具えた方法である。c_daqn9 がこの方法を用いている。

ルーチンの使分け

ルーチンの使分けを述べる前準備として、実用面で現れる被積分関数の型を、ここで、表 1.26 のように分類しておく。さて、ルーチンの使分けで大切なことは、被積分関数がどのルーチンに適しているかを知ることである。そこで、各ルーチンが得意、不得意とする関数を、表 1.26 の分類に従って述べる。

表 1.26 関数の分類

型	意味	例
滑らか型	収束性のよいべき級数展開を持つ解析関数	$\int_0^1 e^{-x} dx$
ピーク型	急激な山や谷が積分区間の中に存在する関数	$\int_{-1}^1 \frac{dx}{(x^2 + 10^{-6})}$
振動型	波長の短い、激しい振動を持つ関数	$\int_0^1 \sin 100\pi x dx$
特異型	代数特異点 (x^α , $-1 < \alpha$), 対数特異点($\log x$)を持つ関数	$\int_0^1 dx / \sqrt{x}$ $\int_0^1 \log x dx$
不連続型	関数値や微係数の不連続点を持つ関数	$\int_0^\pi \cos x dx$

c_daqn9 ニュートン・コーツ9点則に基づく適応型手法である。現在のところ、適応型手法の中では信頼性、経済性のどちらの面でも最も優れたものといってよい。このルーチンは、関数の局所的挙動を巧みにとらえるため、比較的広範な関数に対して使用することができる。まず、積分区間内に代数特異点、対数特異点あるいは不連続点などの異常点を持つ関数に対して、さらにはピーク型の関数に対しても効果的に処理する。

c_daqc8 関数のチェビシェフ級数展開に基づく手法であるために、この級数展開の収束性の良い関数ほど効果的に処理される。例えば滑らかな関数、振動型の関数がそれである。特異点を持つ関数、ピーク型の関数に対しては弱い。

c_daqe 積分区間 $[a, b]$ を変数変換により区間 $(-\infty, \infty)$ 、へ拡張し、そのあと台形則を適用する手法である。その際、変換後の被積分関数が、 $x \rightarrow \pm\infty$ のとき二重指数関数的($a > 0$ として、 $\exp(-a \exp|x|)$)に減衰するような変換が選ばれる。このため、元の区間 $[a, b]$ の端点付近で変化の激しい関数でも効果的に処理される。特に、端点だけに代数特異点あるいは対数特異点を持つ関数に対しては、他のルーチンに比べ最も効率良く積分される。区間 $[a, b]$ の中に特異点を持つ関数に対しては弱い。

以上をまとめて、表 1. 27に、関数の型ごとに最優先的に使用すべきルーチンに○印、使用を避けるべきものに×印をつけ、使分けを明らかにした。空欄の部分は、その使用は最良ではないにしても実用に十分耐えられるか、使用してみる価値があることを意味する。また、滑らか型の関数に対しては、これらのどの関数を使用しても要求精度の積分値が得られるので、ことさら関数の選択に気をを使う必要はないが、関数の計算回数という経済性からいえば、c_daqc8 が最も良い。

表 1. 27ルーチンの使分け

ルーチン名	関数の型						
	滑らか型	ピーク型	振動型	特異点 (端点)	特異点 (端点以外)	不連続型	不明 (注)
c_daqn9		○			○	○	○
c_daqc8	○	×	○	×	×	×	
c_daqe				○	×	×	

注) 性質がよくわからない関数

c) 関数入力多次元積分

3次元以内の多次元積分として、ルーチン c_daqlmc8 及び c_daqme が用意されている。いずれも自動積分ルーチンである。以下、特徴を述べる。

- c_daqlmc8 クレンシヨール・カーチス型積分法を各次元に適用する。このため、滑らかな関数は勿論のこと、振動型の関数に対して強く、反対に、特異点を持つ関数、ピーク型の関数に対して弱い。
- c_daqme 二重指数関数型積分公式を各次元に適用する。本ルーチンは、1次元のルーチン c_daqe, c_daqeh 及び c_daqei で使ったすべての公式を備えているので、積分領域は、有限、半無限、全無限のいずれであってもよく、更に、これらの組合せでできる領域であってもよい。領域の境界で特異点を持つ関数でも効率良く積分する。領域の内部に特異点を持つ関数に対して弱い。

1.14 微分方程式

1.14.1 概要

この節では以下の問題を取り上げる。

常微分方程式(初期値問題)：

連立 1 階常微分方程式の初期値問題,

$$\begin{aligned}
 y_1' &= f_1(x, y_1, \dots, y_n), & y_{10} &= y_1(x_0) \\
 y_2' &= f_2(x, y_1, \dots, y_n), & y_{20} &= y_2(x_0) \\
 &\vdots & &\vdots \\
 y_n' &= f_n(x, y_1, \dots, y_n), & y_{n0} &= y_n(x_0)
 \end{aligned} \tag{103}$$

を解く。

高階常微分方程式の初期値問題も(103)の形に変形して解くことができる。すなわち、

$$y^{(k)} = f(x, y, y', y'', \dots, y^{(k-1)}),$$

$$y_{10} = y(x_0), y_{20} = y'(x_0), \dots, y_{k0} = y^{(k-1)}(x_0)$$

は,

$$y_1 = y(x), y_2 = y'(x), \dots, y_k = y^{(k-1)}(x)$$

とおくことにより次の (104) のような連立 1 階常微分方程式に置き換えることができる.

$$\begin{aligned} y_1' &= y_2, & y_{10} &= y_1(x_0) \\ y_2' &= y_3, & y_{20} &= y_2(x_0) \\ &\vdots & &\vdots \\ y_k' &= f(x, y_1, \dots, y_k), & y_{k0} &= y_k(x_0) \end{aligned} \quad (104)$$

1.14.2 常微分方程式

初期値問題 $y' = f(x, y)$, $y(x_0) = y_0$ を区間 $[x_0, x_e]$ の上で数值的に解くということは, 図 1. 26 に示すように, 離散点, $x_0 < x_1 < x_2 < \dots < x_e$ における近似解を順次求めていくことを意味する.

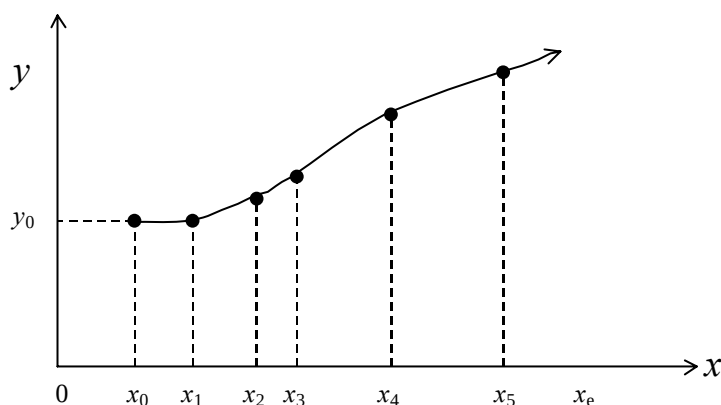


図 1. 26 $y' = f(x, y)$, $y(x_0) = y_0$ の近似解

a) 解の出力

図 1. 26 において, 解の出力点 $x_1, x_2, x_3, \dots$ は, 利用者が指定した点でなくてはならないこともあれば, 微分方程式関数がキザミ制御 (step size control) の結果として選んだ点であってよいこともある.

ところで, 利用者の立場から見ると, 微分方程式を数值的に解く目的としては, 以下のいくつかが挙げられる.

- x_e における解 $y(x_e)$ だけを求める.
- ルーチンがキザミ制御の結果として選んだ点における解を求める. この場合は, 解の挙動を知るのが目的であり, 解の出力点に関して何ら制限を与えず, ただ解の挙動を知るのに十分な分布を成す離散点上で解が求まればよい.
- 利用者の意図した点列 $\{\xi_i\}$ の上で解を求める. あるいは, 一定間隔にとられた点列上で解を求める.

C-SSL II の微分方程式ルーチンは, 解の出力方法(関数から利用者プログラムに戻るタイミング)として, 上記の目的に合わせて, 次の 2 つを設定している.

- 最終値出力- 解 $y(x_e)$ が得られたとき利用者プログラムに戻る. 上記 3) の目的のためには, x_e を順番に ξ_i , $i = 1, 2, \dots$, にとりながらルーチンを繰り返し呼び出す.
- ステップ出力- キザミ制御のもとで, 1 ステップ積分することに利用者プログラムに戻る. 利用者は, 繰り返し関数を呼び出すことにより, 上記 2) の目的を果すことができる.

C-SSL II の関数 `c_dodrk1`, `c_dodam` は、これら 2 つの出力方法を備えている。利用者は引数の指定により出力方法を選ぶことができる。

b) スティフな微分方程式

ここでは、多くの応用分野で現れるスティフな微分方程式について述べ、その定義を明らかにし、簡単な例を挙げる。

(103) をベクトルの形で表すと、

$$\mathbf{y}' = \mathbf{f}(x, \mathbf{y}), \mathbf{y}(x_0) = \mathbf{y}_0 \quad (105)$$

となるここで、

$$\begin{aligned} \mathbf{y} &= (y_1, y_2, \dots, y_n)^T, \\ \mathbf{f}(x, \mathbf{y}) &= (f_1(x, \mathbf{y}), f_2(x, \mathbf{y}), \dots, f_n(x, \mathbf{y}))^T, \\ f_i(x, \mathbf{y}) &= f_i(x, y_1, y_2, \dots, y_n) \end{aligned}$$

である。

手始めに、 $\mathbf{f}(x, \mathbf{y})$ が線形、すなわち、

$$\mathbf{f}(x, \mathbf{y}) = \mathbf{A}\mathbf{y} + \Phi(x) \quad (106)$$

である場合を考える。ここでは $\mathbf{A}$ は定数の係数行列、 $\Phi(x)$ は、ある関数ベクトルである。このとき (105) の解は、 $\mathbf{A}$ の固有値 λ_i と対応する固有ベクトル $\mathbf{u}_i$ を用いて、

$$\mathbf{y}(x) = \sum_{i=1}^n k_i e^{\lambda_i x} \mathbf{u}_i + \Psi(x) \quad (107)$$

k_i : 定数

と表せる。いま、(107) の λ_i , $\Psi(x)$ に関して、次のように仮定する。

- $\operatorname{Re}(\lambda_i) < 0$, for $i=1, 2, \dots, n$
- $\Psi(x)$ は、どの $e^{\lambda_i x}$ よりも滑らか(すなわち、収束性の良いべき級数展開を持つ)関数である。

このような条件下では、 $x \rightarrow \infty$ のとき、

$$\sum_{i=1}^n k_i e^{\lambda_i x} \mathbf{u}_i \rightarrow 0$$

となり、解 $\mathbf{y}(x)$ は、 $\Psi(x)$ に近づく。このように、 $\Psi(x)$ が支配的になったあとは、 $\Psi(x)$ に対する近似解を求めれば十分であり、したがって、キザミ幅も比較的大きく採れることができる。

しかし、オイラー法、ルンゲ・クッタ法などの公式では、キザミ幅をある値 (λ_i に依存する) よりも大きく採ると、あるステップで発生した誤差が、後続のステップで増幅されるという現象が起こる。したがって、これらの公式を適用する限り、キザミ幅の大きさは制約され、実際、 $\max(|\operatorname{Re}(\lambda_i)|)$ が大きければ大きいほど、それに逆比例してキザミ幅は一層小さく採らざるを得ない。

このように、解 $\mathbf{y}(x)$ が実質的に滑らかな関数 $\Psi(x)$ で近似できる状況下にあるにもかかわらず、小さなキザミ幅で積分せざるを得ないという困難さは、結局、解を近似するのに十分なキザミ幅と、誤差の増幅を防ぐのに必要なキザミ幅との間の不均衡に起因する。

(105) において $\Phi(x)=0$ 、すなわち $\Psi(x)=0$ の場合は、解 $\mathbf{y}(x)$ は減衰し、したがって、実質的に、小さい $|\operatorname{Re}(\lambda_i)|$ に対応する項 $k_i e^{\lambda_i x} \mathbf{u}_i$ で近似される。この場合でも、 $\max|\operatorname{Re}(\lambda_i)|$ が大きいと上記の困難が生ずる。

さて、スティフな微分方程式を次のように定義する。

定義 1: 線形微分方程式

$$\mathbf{y}' = \mathbf{A}\mathbf{y} + \Phi(x) \quad (108)$$

が、以下の(109), (110) 満たすとき, その方程式はスティフであると言う.

$$\operatorname{Re}(\lambda_i) < 0, i=1, 2, \dots, n \quad (109)$$

$$\frac{\max(|\operatorname{Re}(\lambda_i)|)}{\min(|\operatorname{Re}(\lambda_i)|)} \gg 1 \quad (110)$$

(111) の左辺は, スティフ率 (stiff ratio) と呼ばれ, この値が大きいか小さいかにより, (109) は強スティフ又は弱スティフであると言う. 実際の応用分野では, スティフ率が 10^6 にも及ぶ強スティフな方程式もしばしば現れる.

以下の (111) は, スティフな線形微分方程式の例であり, 解は (112) で表せる(図 1.27参照).

$$\mathbf{y}' = \begin{pmatrix} 998 & 1998 \\ -999 & -1999 \end{pmatrix} \mathbf{y}, \quad \mathbf{y}(0) = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad (111)$$

$$\mathbf{y} = e^{-x} \begin{pmatrix} 2 \\ -1 \end{pmatrix} + e^{-1000x} \begin{pmatrix} -1 \\ 1 \end{pmatrix} \quad (112)$$

明らかに $x \rightarrow \infty$ のとき $y_1 \rightarrow 2e^{-x}, y_2 \rightarrow -e^{-x}$ である.

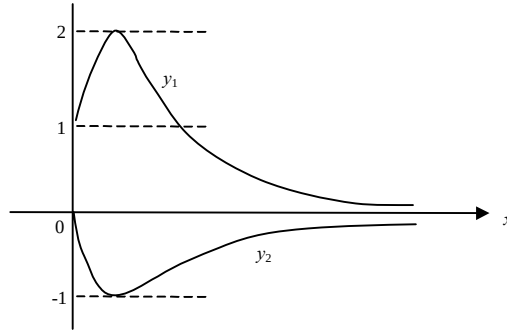


図 1.27 解(112) のグラフ

次に $\mathbf{f}(x, \mathbf{y})$ が非線形である場合を考える. このときは, ヤコビアン行列

$$\mathbf{J} = \frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial \mathbf{y}}$$

の固有値がスティフ性を決定するが, 固有値は x と共に変化する. そこで, $\mathbf{f}(x, \mathbf{y})$ が非線形のときは, 定義 1 を次のように拡張する.

定義 2: 非線形微分方程式

$\mathbf{y}' = \mathbf{f}(x, \mathbf{y})$ が, ある区間 I で, 以下の (113), (114) を満たすとき, その方程式は区間 I でスティフであると言う.

$$\operatorname{Re}(\lambda_i(x)) < 0, \quad i=1, 2, \dots, n \quad x \in I \quad (113)$$

$$\frac{\max(|\operatorname{Re}(\lambda_i)|)}{\min(|\operatorname{Re}(\lambda_i)|)} \gg 1, \quad x \in I \quad (114)$$

ここで $\lambda_i(x)$ は $\mathbf{J}$ の固有値である.

解こうとしている方程式がスティフであるか否かを見分ける方法については, 一般に次のことが言える.

- 方程式が (108) のように線形ならば, $\mathbf{A}$ の固有値を計算することにより直接, スティフ性を調べることができる.
- 一方, 方程式が非線形であるときは, スティフ性検出の能力を持つ関数 `c_dodam` を使えばよい. `c_dodam` は, 非スティフな方程式を解く手法のひとつであるアダムス法を適用するが, 解法の過程で, 方程式がアダムス法にとって苦手な問題であると判断したとき, その旨を引数 `icon` に出力する. そのときは方程式がスティフであると思ってよい.

C-SSL II では、スティフな方程式を解く関数として `c_dodge` を用意している。

c) 関数の使分け

微分方程式の関数を表 1.28 に示す。その使分けは以下のように行う。

- 方程式がスティフなら `c_dodge` を使うこと。
- 方程式が非スティフなら `c_dodrk1` 又は `c_dodam` のいずれかを使うこと。
`c_dodrk1` は特に、以下に示す条件の場合に効果的に利用できる。
 - 微係数 $f(x, y)$ を評価するための計算量が少ない。
 - 解に対する要求精度が高くない。
 - 独立変数 x の特定の点列の上で解の出力を望むとき、その点列の間隔が十分に広い。
 これらの条件のいずれでもないときは `c_dodam` を使うこと。
- 方程式がスティフであるか否か不明なときは、まず、`c_dodam` を使うこと。これにより、もしスティフ性が検出されたときは、`c_dodge` を使うようにすればよい。

表 1.28 常微分方程式の関数

目的	関数名	手法	備考
初期値問題	<code>c_dodrk1</code>	ルンゲ・クッタ ・ヴァーナー法	キザミ可変
	<code>c_dodam</code>	アダムス法	キザミ可変, 次数可変
	<code>c_dodge</code>	ギア法	キザミ可変, 次数可変 (スティフ方程式)

1.15 特殊関数

1.15.1 概要

C-SSL II の特殊関数とは、Fortran の基本関数に含まれていない関数のことであり、これらの関数値を求めるものである。このような特殊関数は、変数と関数が実数型の場合と複素数型の場合に大別され、多くの種類がある。

以下に、幾つかの特殊関数関数に共通な事項を述べる。

a) 精度について

関数で重要なことは、精度と速度のバランスであるが計算式の選定にあたっては、この点に留意し、理論精度(近似式の精度)を倍精度で 18 桁程度となるように選定した。しかし、計算された関数値の精度は、計算機の計算桁数に依存するので、理論精度がいつでも保証されるわけではない。

関数の精度の検定は、4 倍精度で計算する関数(倍精度関数よりも十分精度が良いと確認される関数)の結果を原器と見なして比較を行い、十分実用的であることを確認した。

b) 速度について

特殊関数は、どちらかという精度に重点を置き、そのうえで、速度を上げる配慮をした。実数型は、複素数型の一部であるが、速度を上げるために、別々に用意してある。また、同じ理由で、特殊関数の相互間に関係のある関数も別々に用意してある。また頻繁に使われる関数は、一般的な関数の他に、専用の関数を用意してある。

c) 呼び出し方法について

特殊関数の計算では、いろいろな状況が起きるので、その状況を知らせるために、引数 `icon` を設定するようにしてある。このため、C-SSL II の特殊関数ルーチンは計算結果を引数の 1 つに設定するように作成されている。

d) `icon` について

特殊関数は、その計算のために、Fortran 基本関数の指数関数、三角関数などを利用しているが、それらの中でオーバフロー、アンダフローなど、エラーが発生すると、トラブルの真の原因の発見が遅れる。そこで、特殊関数の側で、できるだけ早めに、トラブルを検出するように配慮し、検出されたトラブルは、引数 `icon` に出力するようにしてある。

1.15.2 楕円積分

楕円積分は、表 1.29 のような種類に分かれる。

完全楕円積分の計算法には、2 次の反復法があるが、この方法は変数の大きさによって、速度が変る欠点がある。そこで、本ルーチンは、変数の大きさに関係なく速度を一定に保つように、近似式を用いて計算する。

表 1.29 楕円積分の関数

項目	数学記号	関数名
完全第 1 種完全楕円積分	$K(x)$	<code>c_dceli1</code>
完全第 2 種完全楕円積分	$E(x)$	<code>c_dceli2</code>

1.15.3 指数積分

指数積分は、表 1.30 のようになっている。

表 1.30 指数積分の関数

項目	数学記号	関数名
指数積分	$E_i(-x), x > 0$	<code>c_dexp1</code>
	$\overline{E}_i(x), x > 0$	

指数積分は、計算の困難な関数で、計算式はいくつかの区間に分けられて、いろいろな近似式が使われている。

1.15.4 正弦・余弦積分

正弦・余弦積分は、表 1.31 のようになっている。

表 1.31 正弦・余弦積分の関数

項目	数学記号	関数名
正弦積分	$S(x)$	<code>c_dsini</code>
余弦積分	$C(x)$	<code>c_dcasi</code>

1.15.5 フレネル積分

フレネル積分は表 1.32 のようになっている。

表 1.32 フレネル積分の関数

項目	数学記号	関数名
正弦フレネル積分	$S(x)$	c_dsfri
余弦フレネル積分	$C(x)$	c_dcfri

1.15.6 ガンマ関数

ガンマ関数には、表 1.33 のものが用意されている。

表 1.33 ガンマ関数の関数

項目	数学記号	関数名
第 1 種不完全ガンマ関数	$\gamma(v, x)$	c_digam1
第 2 種不完全ガンマ関数	$\Gamma(v, x)$	c_digam2

(完全)ガンマ関数 $\Gamma(v)$ と第 1 種及び第 2 種不完全ガンマ関数の間には、

$$\Gamma(v) = \gamma(v, x) + \Gamma(v, x)$$

の関係がある。

なお、 $\Gamma(v)$ については C 言語の数学関数を利用のこと。

1.15.7 誤差関数

誤差関数には表 1.34 のものが用意されている。

表 1.34 誤差関数の関数

項目	数学記号	関数名
逆誤差関数	$\operatorname{erf}^{-1}(x)$	c_dierf
逆余誤差関数	$\operatorname{erfc}^{-1}(x)$	c_dierfc

逆誤差関数と逆余誤差関数の間には

$$\operatorname{erf}^{-1}(x) = \operatorname{erfc}^{-1}(1 - x)$$

の関係があり、それぞれ x の範囲に応じて適切な方で計算している。

なお、誤差関数 $\operatorname{erf}(x)$ については C 言語の数学関数を利用のこと。

1.15.8 ベッセル関数

実変数のベッセル関数は、表 1.35 のような種類に分けられ、非常に多くの種類がある。また利用頻度も高い。特に、0 次、1 次は利用頻度が高いので、専用関数を用意した。0 次、1 次を計算するときには、専用関数を使用する方が速度的に有利である。複素変数のベッセル関数は、表 1.36 のようになっている。

表 1.35 実変数のベッセル関数の関数

項目		数学記号	関数名
第一種	0 次ベッセル関数	$J_0(x)$	c_dbj0
	1 次ベッセル関数	$J_1(x)$	c_dbj1
	整数次ベッセル関数	$J_n(x)$	c_dbjn
	実数次ベッセル関数	$J_\nu(x) \ (\nu \geq 0.0)$	c_dbjr
	0 次変形ベッセル関数	$I_0(x)$	c_dbj0

項目		数学記号	関数名
	1 次変形ベッセル関数	$I_1(x)$	c_dbi1
	整数次変形ベッセル関数	$I_n(x)$	c_dbin
	実数次変形ベッセル関数	$I_\nu(x) \ (\nu \geq 0.0)$	c_dbir
第二種	0 次ベッセル関数	$Y_0(x)$	c_dby0
	1 次ベッセル関数	$Y_1(x)$	c_dby1
	整数次ベッセル関数	$Y_n(x)$	c_dbyn
	実数次ベッセル関数	$Y_\nu(x) \ (\nu \geq 0.0)$	c_dbyr
	0 次変形ベッセル関数	$K_0(x)$	c_dbk0
	1 次変形ベッセル関数	$K_1(x)$	c_dbk1
	整数次変形ベッセル関数	$K_n(x)$	c_dbkn
	実数次変形ベッセル関数	$K_\nu(x)$	c_dbkr

表 1.36 複素変数のベッセル関数の関数

項目		数学記号	関数名
第一種	整数次ベッセル関数	$J_n(z)$	c_dcbjn
	実数次ベッセル関数	$J_\nu(z) \ (\nu \geq 0.0)$	c_dcbjr
	整数次変形ベッセル関数	$I_n(z)$	c_cbin
第二種	整数次ベッセル関数	$Y_n(z)$	c_dcbyn
	整数次変形ベッセル関数	$K_n(z)$	c_dcbtn

1.15.9 正規分布関数

正規分布関数には, 表 1.37 のものが用意されている.

表 1.37 正規分布関数の関数

項目	数学記号	関数名
正規分布関数	$\phi(x)$	c_dndf
余正規分布関数	$\psi(x)$	c_dndfc
逆正規分布関数	$\phi^{-1}(x)$	c_dindf
逆余正規分布関数	$\psi^{-1}(x)$	c_dindfc

1.16 擬似乱数

1.16.1 概要

この節では種々の分布に従う擬似乱数の生成を扱う。

1.16.2 擬似乱数の生成

種々の確率分布に従う擬似乱数はすべて、一様分布から得られた擬似乱数に適当な変換を施すことにより得られる。すなわち、ある分布の擬似乱数を生成するとき、その分布の確率密度関数を $g(x)$ とするならば、擬似乱数 y はその分布の累積分布関数 $F(y) = \int_0^y g(x)dx$ の逆関数

$$y = F^{-1}(u)$$

で示される。ここに、 u は生成された一様乱数である。

ただし、離散型分布の擬似乱数生成は中間の計算で少し複雑になる。例えば `c_dranp2` は、まず、ポアソン分布の累積分布表を作成する。次に累積分布表を参照する索引表を作成し、それから累積分布表で与えられた分布に従う擬似乱数を生成するといった具合である。

C-SSL II で準備する擬似乱数生成関数の種類を表 1. 38 に示す。なお、これらの関数には初期値を与える引数が設けられており、これによってある初期値のもとでの擬似乱数列の生成を続けるかどうかのコントロールを行うことができる。(通常は、ただ一回だけの初期値設定を行ったかたちで使用される)。また、`c_rane2`, `c_ranp2`, `c_ranb2` は、他の C-SSL II ルーチンとは異なり、単精度の結果を返す。

表 1. 38 擬似乱数生成関数の種類

種類	精度	関数名
一様乱数 $[0, 1)$	倍精度	<code>c_dvrau4</code>
正規乱数	倍精度	<code>c_dvran3</code> <code>c_dvran4</code>
指数乱数	単精度	<code>c_rane2</code>
ポアソン乱数	単精度	<code>c_ranp2</code>
二項乱数	単精度	<code>c_ranb2</code>

第2章 関数の使用方法

c_daggm

行列の和 (実行列)

<pre>ierr = c_daggm(a, ka, b, kb, c, kc, m, n, &icon);</pre>

1. 機能

$m \times n$ の実行列 **A** と実行列 **B** の和 **C** を求める.

$$\mathbf{C} = \mathbf{A} + \mathbf{B}$$

ここで **C** は, $m \times n$ の実行列である. $m, n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_daggm((double*)a, ka, (double*)b, kb, (double*)c, kc, m, n, &icon);
```

引数の説明:

a	double	入力	行列 A .
	a[m][ka]		
ka	int	入力	配列 a の整合寸法 ($\geq n$).
b	double	入力	行列 B .
	b[m][kb]		
kb	int	入力	配列 b の整合寸法 ($\geq n$).
c	double	出力	行列 C (“使用上の注意”参照).
	c[m][kc]		
kc	int	入力	配列 c の整合寸法 ($\geq n$).
m	int	入力	行列 A, B, C の行数 m .
n	int	入力	行列 A, B, C の列数 n .
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
30000	次のいずれかであった. • $m < 1$ • $n < 1$ • $ka < n$ • $kb < n$ • $kc < n$	処理を打ち切る.

3. 使用上の注意

a) 領域の節約について

配列 a あるいは b 上の内容を保存する必要のない場合は, 次のように呼び出すことにより領域が節約できる.

配列 a の内容が不要の場合:

```
ierr = c_daggm((double *)a, ka, (double *)b, kb, (double *)a, ka, m, n, &icon);
```

配列 b の内容が不要の場合:

```
ierr = c_daggm((double *)a, ka, (double *)b, kb, (double *)b, kb, m, n, &icon);
```

この場合, 行列 C は配列 a あるいは, 配列 b 上に得られる.

4. 使用例

行列の和を計算し, 結果を確認する. 行列のサイズは 100×100.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 100

MAIN__()
{
    int ierr, icon;
    int n, m, ka, kb, kc, i, j;
    double eps, err;
    double a[NMAX][NMAX], b[NMAX][NMAX], c[NMAX][NMAX];

    /* initialize matrices*/
    m = NMAX;
    n = NMAX;
    ka = NMAX;
    kb = NMAX;
    kc = NMAX;
    for (i=0; i<n; i++)
        for (j=0; j<n; j++) {
            a[i][j] = n-i-j;
            b[i][j] = i+j;
        }
    /* add matrices */
    ierr = c_daggm((double*)a, ka, (double*)b, kb, (double*)c, kc, m, n, &icon);
    if (icon != 0) {
        printf("ERROR: c_daggm failed with icon = %d\n", icon);
        exit(1);
    }
    /* check matrix */
    eps = 1e-6;
    for (i=0; i<n; i++)
        for (j=0; j<n; j++) {
            err = fabs((c[i][j]-n)/n);
            if (err > eps) {
                printf("WARNING: result inaccurate\n");
                exit(1);
            }
        }
    printf("Result OK\n");
    return(0);
}
```

c_dakher

エイトケン・エルミート補間

<pre>ierr = c_dakher(x, y, dy, n, v, &m, &eps, &f, vw, &icon);</pre>

1. 機能

離散点 $x_1, x_2, \dots, x_n$ ($x_1 < x_2 < \dots < x_n$) に対して、関数値 $y_i = f(x_i)$, 1階微係数 $y'_i = f'(x_i)$, $i = 1, \dots, n$ が与えられたとき、 $x = v$ における補間値をエイトケン・エルミート補間法により求める。 $n \geq 1$ であること。

2. 引数

呼出し形式:

```
ierr = c_dakher(x, y, dy, n, v, &m, &eps, &f, vw, &icon);
```

引数の説明:

x	double x[n]	入力	離散点 x_i .
y	double y[n]	入力	関数値 y_i .
dy	double dy[n]	入力	1階微係数 y'_i .
n	int	入力	離散点の個数 n .
v	double	入力	補間したい点の x 座標 v .
m	int	入力	補間に使う離散点の個数の上限 ($\leq n$).
		出力	実際に使用した離散点の個数. ("使用上の注意" a)参照)
eps	double	入力	しきい値.
		出力	補間値の絶対誤差. ("使用上の注意" b)参照)
f	double	出力	補間値.
vw	double vw[5n]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	v が離散点 x_i の一つであった.	$f = y_i$ とする.
30000	次のいずれかであった. $n < 1$ $m = 0$ $x[i-1] \geq x[i]$	$f = 0.0$ として処理を打ち切る.

3. 使用上の注意

a) m の与え方

- もし、真の関数が、 $x=v$ の近傍で高々 $2k-1$ 次の多項式で近似できることが分かっているならば、多項式補間法においても、高々 $2k-1$ 次の多項式で行うのが自然である。もし、このような状況にあるなら引数 m に k を入れる。
- 上記 1. の事実が分からないときは、引数 m に、引数 n の値と等しい値を入れる。
- もし、ある定まった個数の離散点を、収束判定することなく無条件に使う、補間値を求めたいときには引数 m に、その個数に -1 を乗じた値を入れる。(例えば、 $m = -k$ 又は $m = -n$).

b) 収束判定条件及び eps について

まず補間多項式の次数と、数値的なふるまいを考える。ここで $x=v$ の近傍での j 個の離散点を用いて補間した補間値を Z_j で表す。(離散点のとり方は、 $x=v$ に近い順とする。) 次に間差 D_j を次のように定義する。

$$D_j \equiv Z_j - Z_{j-1} \quad j = 2, \dots, m$$

ここで m は補間に使う離散点の個数の上限である。一般に、補間多項式の次数を上げていくと、 $|D_j|$ は図 dakher-1. のようなふるまいを示す。

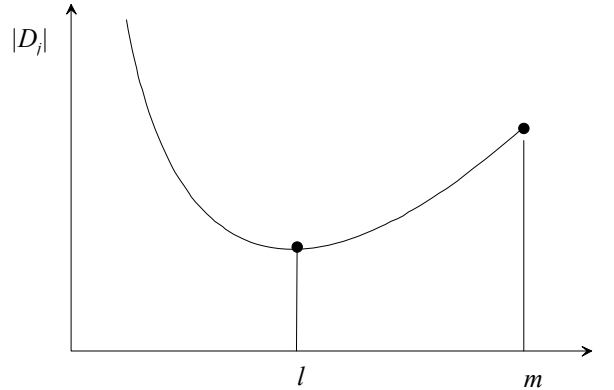


図 dakher-1. 補間多項式の次数を上げたときの $|D_j|$ のふるまい

図 dakher-1. で l は近似多項式の打ち切り誤差と計算誤差が互いに同程度になる事を示しており、一般に Z_l が数値的には最適の補間値とみなすことができる。

しかしながら、 D_j は近似関数により種々のふるまいを示す。図 dakher-2. で示されるように振動する場合もある。

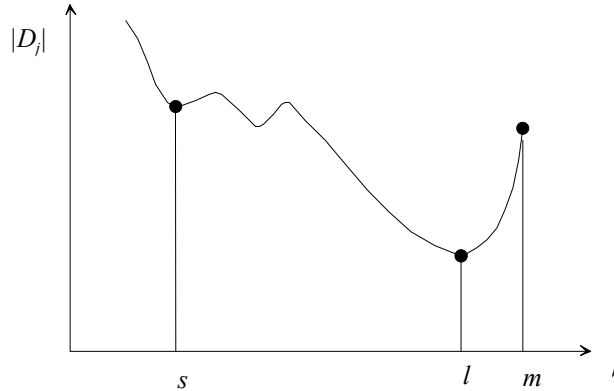


図 dakher-2. $|D_j|$ が振動する場合

この場合、補間値として Z_s を採用するのは好ましくなく Z_l を採用すべきである。この考えに基いて出力すべき補間値を次のように決定する。

$D_2, D_3, \dots, D_m$ と計算していく過程で、

- $|D_j| > |\text{eps}|$, $j = 2, 3, \dots, m$ であれば、

$$|D_l| = \min_j (|D_j|)$$

となる l を求め $Z_l, l, |D_l|$ をそれぞれ引数 f, m, eps に出力し、

- $|D_l| \leq |\text{eps}|$ となり始めたならば、その時点からは、

$$|D_l| \leq |D_{l+1}|$$

となる l を求め $Z_l, l, |D_l|$ を出力する。

もしいつまでも成立たなければ $l = m$ として $Z_m, m, |D_m|$ を出力する。

又 $\text{eps} = 0.0$ と入力すると $|D_j|$ が最小となる時点の Z_j を補間値として出力することになる。

4. 使用例

関数 $f(x) = \sin x$ を $[0, \pi]$ 区間で等間隔な 10 個の補間点を与えられたときの、ある点での補間値を求め結果を元の関数と比較する。

```
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 10

MAIN__()
{
    int ierr, icon;
    int i, n, m;
    double x[NMAX], y[NMAX], dy[NMAX], vw[5*NMAX];
    double p, h, v, f, eps, exact, pi;

    /* initialize data */
    n = NMAX;
    m = n;
    p = 0;
    pi = 2*asin(1);
    h = pi/(n-1);
    /* set function and derivative values */
    for (i=0; i<n; i++) {
        x[i] = p+i*h;
        y[i] = sin(x[i]);
        dy[i] = cos(x[i]);
    }
    eps = 1e-6;
    v = pi/2;
    exact = sin(v);
    /* interpolate */
    ierr = c_dakher(x, y, dy, n, v, &m, &eps, &f, vw, &icon);
    printf("icon = %i    f = %12.6e    m = %i    eps = %12.6e\n", icon, f, m, eps);
    eps = 1e-6;
    /* check result */
    if (fabs((f-exact)/exact) > eps)
        printf("Inaccurate result\n");
    else
        printf("Result OK\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の AKHER の項目及び[40]を参照のこと。

c_daklag

エイトケン・ラグランジュ補間
<pre>ierr = c_daklag(x, y, n, v, &m, &eps, &f, vw, &icon);</pre>

1. 機能

離散点 $x_1, x_2, \dots, x_n$ ($x_1 < x_2 < \dots < x_n$) に対して, 関数値 $y_i = f(x_i)$, $i = 1, \dots, n$ が与えられたとき, $x = v$ における補間値をエイトケン・ラグランジュ補間法により求める. $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_daklag(x, y, n, v, &m, &eps, &f, vw, &icon);
```

引数の説明:

x	double x[n]	入力	離散点 x_i .
y	double y[n]	入力	関数値 y_i .
n	int	入力	離散点の個数 n .
v	double	入力	補間したい点の x 座標 v .
m	int	入力	補間に使う離散点の個数の上限 ($\leq n$).
		出力	実際に使用した離散点の個数. ("使用上の注意" a) 参照)
eps	double	入力	しきい値.
		出力	補間値の絶対誤差. ("使用上の注意" c) 参照)
f	double	出力	補間値.
vw	double vw[4*n]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	v が離散点 x_i と一致した.	$f = y_i$ とする.
30000	次のいずれかであった. $n < 1$ $m = 0$ $x_i \geq x_{i+1}$	$f = 0.0$ として処理を打ち切る.

3. 使用上の注意

a) m の与え方

- もし, 真の関数が, $x = v$ の近傍で高々 k 次の多項式で近似できることがわかっていれば, 多項式補間法においても, 高々 k 次の多項式で行うのが自然である. もし, このような状況にあるなら, 引数 m に $k+1$ を入れる.

2. 上記 1.の事実がわからないときは, 引数 m に引数 n と等しい値を入れる.
3. もし, ある定まった個数の離散点を, 収束判定することなく無条件に使う補間値を求めたいときには, 引数 m に, その個数に -1 を乗じた値を入れる.

b) 収束判定条件について

まず補間多項式の次数と, 数値的なふるまいを考える. ここで $x=v$ の近傍での j 個の離散点を用いて補間した補間値を Z_j で表す. (離散点のとり方は, $x=v$ に近い順とする.)

次に間差 D_j を次のように定義する.

$$D_j \equiv Z_j - Z_{j-1}$$

ここで $j = 2, \dots, m$ であり, m は補間に使う離散点の個数の上限である. 一般に, 補間多項式の次数を上げていくと $|D_j|$ は図 daklag-1. のようなふるまいを示す.

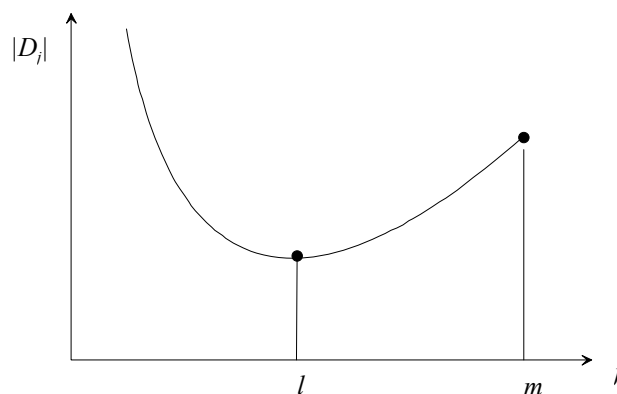


図 daklag-1. 補間多項式の次数を上げたときの $|D_j|$ のふるまい

図 daklag-1. で l は近似多項式の打ち切り誤差と計算誤差が互いに同程度になることを示しており, 一般に Z_l が数値的には最適の補間値と見なすことができる.

c) eps の与え方

次の条件を考慮する.

収束判定は b) に従って行われるが, D_j は近似関数により種々のふるまいを示す. 図 daklag-2. で示されるように振動する場合もある.

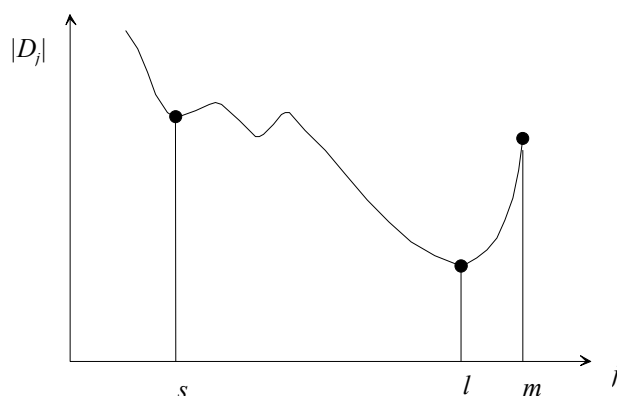


図 daklag-2. $|D_j|$ のふるまい

この場合, 補間値として Z_s を採用するのは好ましくなく Z_l を採用すべきである. この考えに基づいて出力すべき補間値を次のように決定する.

$D_2, D_3, \dots, D_m$ と計算していく過程で,

- $|D_j| > |\text{eps}|$, $j = 2, 3, \dots, m$ であれば,

$$|D_l| = \min |D_j| \quad (1)$$

となる l を求め, $Z_b, l, |D_l|$ をそれぞれ引数 f, m, eps に出力し,

- $|D_j| \leq |\text{eps}|$ となり始めたならば, その時点からは,

$$|D_l| \leq |D_{l+1}| \quad (2)$$

となる l を求め, $Z_b, l, |D_l|$ を出力する.

もし (2) がいつまでも成立たなければ $l = m$ として $Z_m, m, |D_m|$ を出力する.

又 $\text{eps} = 0.0$ と入力すると $|D_j|$ が最小となる時点の Z_j を補間値として出力することになる.

4. 使用例

[0,1] 区間で等間隔な 10 個の補間点での関数 $f(x) = \sin(x)\sqrt{x}$ の値が与えられたとき, ある点での補間値を求め結果を元の関数と比較する.

```
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 10

MAIN__()
{
    int ierr, icon;
    int i, n, m;
    double x[NMAX], y[NMAX], v, f, eps, vw[4*NMAX], h, p, exact;

    /* initialize data */
    n = NMAX;
    p = 0;
    h = 1.0/n;
    for (i=0; i<n; i++) {
        x[i] = p;
        y[i] = sin(p)*sqrt(p);
        p = p + h;
    }
    m = n;
    v = x[n/2] + (x[n/2+1]-x[n/2])/2;
    eps = 1e-6;
    exact = sin(v)*sqrt(v);
    /* interpolate */
    ierr = c_daklag(x, y, n, v, &m, &eps, &f, vw, &icon);
    printf("icon = %i   f = %12.6e   m = %i   eps = %12.6e\n", icon, f, m, eps);
    eps = 1e-6;
    /* check result */
    if (fabs((f-exact)/exact) > eps)
        printf("Inaccurate result\n");
    else
        printf("Result OK\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の AKLAG の項目及び[89]を参照のこと.

c_dakmid

2次元準エルミート補間式による補間

<pre>ierr = c_dakmid(x, nx, y, ny, fxy, k, &isw, vx, &ix, vy, &iy, &f, vw, &icon);</pre>

1. 機能

格子点 (x_i, y_j) , $i = 1, 2, \dots, n_x$, $j = 1, 2, \dots, n_y$ ($x_1 < x_2 < \dots < x_{n_x}$, $y_1 < y_2 < \dots < y_{n_y}$) に対して, 関数値 $f_{ij} = f(x_i, y_j)$ が与えられたとき, 点 $P(v_x, v_y)$ における補間値を, 双3次の区分的2次元準エルミート補間式により求める. $x_1 \leq v_x \leq x_{n_x}$, $y_1 \leq v_y \leq y_{n_y}$ 及び $n_x \geq 3, n_y \geq 3$ であること.

2. 引数

呼出し形式:

```
ierr = c_dakmid(x, nx, y, ny, fxy, k, &isw, vx, &ix, vy, &iy, &f, vw,
                &icon);
```

引数の説明:

x	double x[nx]	入力	x 方向の離散点 x_i .
nx	int	入力	x_i の個数 n_x .
y	double y[nx]	入力	y 方向の離散点 y_j .
ny	int	入力	y_j の個数 n_y .
fxy	double fxy[nx][k]	入力	関数値 f_{ij} .
k	int	入力	2次元配列 fxy の整合寸法 ($\geq ny$).
isw	int	入力	与えられた入力データ (x_i, y_j, f_{ij}) に対して初めて本関数を呼び出すときに, isw=0 と入力する. 同一の入力データに対して, 本関数を繰り返し呼び出すことにより, 一連の補間値を求めるときは, 2回目以降, isw を変更してはならない.
		出力	$x_i \leq v_x < x_{i+1}$, $y_j \leq v_y < y_{j+1}$ を満たす (i, j) に関する情報. 新たに与えられたデータ (x_i, y_j, f_{ij}) に対して補間を始めるときは, 再び isw=0 とする必要がある.
vx	double	入力	点 $P(v_x, v_y)$ の x 座標.
ix	int	入力	$x_i \leq v_x < x_{i+1}$ のとき ix には $i-1$ を入力する. ix は $x[ix] \leq vx < x[ix+1]$ を満たす. $v_x = x_{n_x}$ のときは $ix = n_x - 2$ とすること.
		出力	$x_i \leq v_x < x_{i+1}$ のとき $i-1$ を出力する. (“使用上の注意”b) 参照)
vy	double	入力	点 $P(v_x, v_y)$ の y 座標.
iy	int	入力	$y_j \leq v_y < y_{j+1}$ のとき iy には $j-1$ を入力する. iy は $y[iy] \leq vy < y[iy+1]$ を満たす. $v_y = y_{n_y}$ のときは $iy = n_y - 2$ とすること.
		出力	$y_j \leq v_y < y_{j+1}$ のとき $j-1$ を出力する. (“使用上の注意”b) 参照)
f	double	出力	補間値.
vw	double vw[50]	作業領域	同一の入力データ (x_i, y_j, f_{ij}) に対して繰り返し本関数を呼び出

アイコン int 出力 ときには、その間 vw の内容を変更してはならない。
コンディションコード。下の表を参照。

コンディションコード：

コード	意味	処理内容
0	エラーなし。	正常終了。
10000	$x[ix] \leq vx < x[ix+1]$ 又は $y[iy] \leq vy < y[iy+1]$ が満たされていない。	本関数内で左記の ix 又は iy を捜して処理を続ける。
29000	作業用領域の獲得に失敗した。	処理を打ち切る。
30000	次のいずれかであった。 $x[i] \geq x[i+1]$ なる $x[i]$ がある。 $y[j] \geq y[j+1]$ なる $y[j]$ がある。 $nx < 3$ 又は $ny < 3$ $k < ny$ $vx < x[0]$ 又は $vx > x[nx-1]$ $vy < y[0]$ 又は $vy > y[ny-1]$ - isw の指定に誤りがある。	処理を打ち切る。

3. 使用上の注意

a) 本関数の使用方法

本関数で用いる補間式は、領域 $\mathbf{R}=\{(x,y) \mid x_1 \leq x \leq x_{n_x}, y_1 \leq y \leq y_{n_y}\}$ で、 x 方向、 y 方向各々独立に 1 階までの導関数は連続であるが、2 階以上の導関数は一般に不連続となる。一方で、この補間式は、不自然な屈曲点や面が現れないのが特徴であり、“曲面のあてはめ”を効率良く実行するものである。

2 変数関数の補関値、又は微分値、積分値をより高精度で得るためには、Spline 関数による補間関数 c_dbifd3 を用いた方がよい。

本関数により、同一の入力データ (x_i, y_j, f_{ij}) のもとで、多数の補間値を求めたいときは、同一の格子内に属する点が連続するようにして本関数を呼び出すと効率がよい。なお、このとき引数 isw, vw の内容を変更してはならない。

b) ix 及び iy

引数 ix 及び iy は $x[ix] \leq vx < x[ix+1]$, $y[iy] \leq vy < y[iy+1]$ を満足していることが好ましい。もし満足していないときは、このような条件を満足するような ix, iy を探索して処理を続ける。

ここで、数学で一般に使用する添え字と C 言語の配列の添え字では、値が 1 ずれることに注意する必要がある。すなわち、C 言語の添え字は 0 から始まり、数学の添え字は 1 から始まるため ix, iy は $i-1, j-1$ を設定する。

c) icon の判定について

本関数が icon = 30000 として引数エラーとする条件は、コンディションコードの表に示した通りであるが、その中で条件の最初の 4 項目については isw = 0 と指定されたとき(すなわち、与えられた入力データ (x_i, y_j, f_{ij}) に対して、初めて呼び出されたとき)だけチェックし、2 回目以降の呼び出し時には、チェックしていない。

4. 使用例

関数 $f(x, y) = \sin(xy)\sqrt{xy}$ を区間 $[0,1] \times [0,1]$ の中の 100 個の補間点に対して、ある点での関数の値を求め、結果を元の関数と比較する。

```
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 10

MAIN__()
{
    int ierr, icon;
    int i, j, nx, ny, k, ix, iy, isw;
    double x[NMAX], y[NMAX], fxy[NMAX][NMAX], eps, vw[50], exact;
    double hx, hy, px, py, vx, vy, f;

    /* initialize data */
    nx = NMAX;
    ny = NMAX;
    k = NMAX;
    isw = 0;
    hx = 1.0/(nx-1);
    hy = 1.0/(ny-1);
    px = 0;
    for (i=0; i<nx; i++) {
        x[i] = px;
        px = px + hx;
    }
    py = 0;
    for (j=0; j<ny; j++) {
        y[j] = py;
        py = py + hy;
    }
    for (i=0; i<nx; i++)
        for (j=0; j<ny; j++) {
            px = x[i];
            py = y[j];
            fxy[i][j] = sin(px*py)*sqrt(px*py);
        }
    ix = nx/2;
    vx = x[ix] + (x[ix+1]-x[ix])/2;
    iy = ny/2;
    vy = y[iy] + (y[iy+1]-y[iy])/2;
    exact = sin(vx*vy)*sqrt(vx*vy);
    /* interpolate */
    ierr = c_dakmid(x, nx, y, ny, (double*)fxy, k, &isw,
                   vx, &ix, vy, &iy, &f, vw, &icon);
    printf("icon = %i    f = %12.6e\n", icon, f);
    eps = 1e-4;
    /* check result */
    if (fabs((f - exact)/exact) > eps)
        printf("Inaccurate result\n");
    else
        printf("Result OK\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の AKMID の項目及び[3]を参照のこと。

c_dakmin

準エルミート補間式

ierr = c_dakmin(x, y, n, c, d, e, &icon);

1. 機能

離散点 $x_1, x_2, \dots, x_n$ ($x_1 < x_2 < \dots < x_n$) に対して, 関数値 $y_i = f(x_i)$, $i = 1, \dots, n$ が与えられたとき, 準エルミート補間法を用いて, $f(x)$ に対する(1)で表される3次の区分的な補間式 $S(x)$ を求める.

$$S(x) = y_i + c_i(x - x_i) + d_i(x - x_i)^2 + e_i(x - x_i)^3 \quad (1)$$

ここで $x_i \leq x \leq x_{i+1}$, $i = 1, 2, \dots, n-1$ であり, $n \geq 3$.

2. 引数

呼出し形式:

```
ierr = c_dakmin(x, y, n, c, d, e, &icon);
```

引数の説明:

x	double x[n]	入力	離散点 x_i .
y	double y[n]	入力	関数値 y_i .
n	int	入力	離散点の個数 n .
c	double c[n-1]	出力	係数 c_i .
d	double d[n-1]	出力	係数 d_i .
e	double e[n-1]	出力	係数 e_i .
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	次のいずれかであった. $x[i] \geq x[i+1]$ $n < 3$	処理を打ち切る.

3. 使用上の注意

a) 本関数の特徴

本関数により得られる補間式は, 不自然な屈曲点(wiggles)が現れず, 人間が手で描いた曲線に最も近いという特徴を持っている. しかし, この補間式は区間 (x_1, x_n) で1階までの導関数は連続であるが, 2階以上は一般に不連続である.

もし, $f(x)$ が2次多項式であり, かつ, x_i , $i = 1, \dots, n$ が等間隔で与えられたときは, 得られる補間式は, 計算誤差がないとすると, $f(x)$ そのものである.

もし, 区間外 ($x < x_1$ 又は $x > x_n$) に対して利用したいときは, 精度は良くないが (1) における $i = 1$ 又は $i = n-1$ の式を用いればよい.

4. 使用例

関数 $f(x) = \sin(x)\sqrt{x}$ を $[0,1]$ 区間で等間隔な 10 個の補間点が与えられたときの、ある点での補間値を求め結果を元の関数と比較する。

```
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 10

MAIN__()
{
    int ierr, icon;
    int i, n, k;
    double x[NMAX], y[NMAX], c[NMAX-1], d[NMAX-1], e[NMAX-1];
    double f, eps, h, p, v, exact;

    /* initialize data */
    n = NMAX;
    p = 0;
    h = 1.0/n;
    for (i=0; i<n; i++) {
        x[i] = p;
        y[i] = sin(p)*sqrt(p);
        p = p + h;
    }
    /* calculate interpolation coefficients */
    ierr = c_dakmin(x, y, n, c, d, e, &icon);
    k = n/2;
    v = x[k] + (x[k+1]-x[k])/2;
    exact = sin(v)*sqrt(v);
    /* calculate function value using coefficients */
    h = v-x[k];
    f = y[k] + (c[k]+(d[k]+e[k]*h)*h)*h;
    printf("calculated = %12.6e    exact = %12.6e\n", f, exact);
    eps = 1e-4;
    /* check result */
    if (fabs((f-exact)/exact) > eps)
        printf("Inaccurate result\n");
    else
        printf("Result OK\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の AKMIN の項目及び[2]を参照のこと。

c_daqc8

1次元有限区間積分 (関数入力, クレンショー・カーチス型積分法)

```
ierr = c_daqc8(a, b, fun, epsa, epsr, nmin,
               nmax, &s, &err, &n, &icon);
```

1. 機能

関数 $f(x)$, a, b, ε_a 及び ε_r が与えられたとき, (1) を満たす積分の近似値 S を, 等差数列的に分点を増すクレンショー・カーチス型積分法により求める.

$$\left| S - \int_a^b f(x) dx \right| \leq \max \left(\varepsilon_a, \varepsilon_r \cdot \left| \int_a^b f(x) dx \right| \right) \quad (1)$$

2. 引数

呼出し形式:

```
ierr = c_daqc8(a, b, fun, epsa, epsr, nmin, nmax, &s, &err, &n, &icon);
```

引数の説明:

a	double	入力	積分区間の下限 a .
b	double	入力	積分区間の上限 b ($b \leq a$ でも可).
fun	function	入力	被積分関数 $f(x)$ を計算する関数名. プロトタイプ宣言 double fun (double x); 引数の説明 x double 入力 積分変数 x .
epsa	double	入力	積分の近似値 S に対する絶対誤差の上限 ε_a (≥ 0). (“使用上の注意”c)参照)
epsr	double	入力	積分の近似値 S に対する相対誤差の上限 ε_r (≥ 0). (“使用上の注意”c)参照)
nmin	int	入力	被積分関数の計算回数下限 (≥ 0), 標準値としては 15 が適当. (“使用上の注意”b)参照)
nmax	int	入力	被積分関数の計算回数上限 ($nmax \geq nmin$). 標準値としては 511 が適当. これよりも大きく指定した場合は 511 と見なす. (“使用上の注意”b)参照)
s	double	出力	積分の近似値. (“使用上の注意”c)参照)
err	double	出力	積分の近似値 S の絶対誤差の推定値. (“使用上の注意”c)参照)
n	int	出力	被積分関数の計算回数.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	丸め誤差のために要求精度が得られなかった.	s には, 得られた近似値を出力する. 精度は達成可能な限度まで達している.
20000	被積分関数の計算回数がその上限に達しても要求精度が得られなかった.	処理を打ち切る. s には, そのときまでに得られている近似値を出力するが精度は保証できない.
30000	次のいずれかであった: • $\text{epsa} < 0$ • $\text{epsr} < 0$ • $\text{nmin} < 0$ • $\text{nmax} < \text{nmin}$	処理を打ち切る.

3. 使用上の注意

a) c_daqc8 の使用方法

本ルーチンを多数回呼ぶとき, 最初だけ 511 個の定数 (積分公式の分点と重みの表) を計算し, 2 回目以降は省略する. したがって, 2 回目以降は, 計算量がその分だけ少ない.

本ルーチンは, 被積分関数 $f(x)$ が振動型の関数の場合に特に有効にはたらく. また, 滑らかな関数の場合は c_daqn9 及び c_daqe に比べ $f(x)$ の計算回数が少なく, 効率が最もよい. したがって, 滑らかな関数及び振動型の関数に対しては本ルーチンの使用を第一に勧める. 特異点を持つ関数に対しては, その特異点が積分区間 $[a, b]$ の端点だけに位置する場合は c_daqe を, また, 特異点が区間内に位置する場合及びピーク型の関数に対しては c_daqn9 を用いた方がよい.

b) nmin 及び nmax

本関数は, 被積分関数 $f(x)$ の計算関数を,

$$\text{nmin} \leq n \leq \text{nmax}.$$

の範囲に限定している. すなわち収束判定にかかわらず, $f(x)$ は少なくとも nmin 回計算され, しかも nmax 回を超えて計算されることはない. もし nmax 回以内に (1) を満たす S が求まらなかった場合には icon を 20000 として処理を打ち切る. なお nmax が 15 未満の値であったときは, 15 が与えられたものと見なす.

c) 積分の近似値 S の精度

本ルーチンは, 与えられた $\varepsilon_a, \varepsilon_r$ に対して (1) を満たす S を求める. したがって, $\varepsilon_r=0$ とおけば絶対誤差 ε_a 以内の, また $\varepsilon_a=0$ とおけば相対誤差 ε_r 以内の精度で積分の近似値を求めることになる. しかし, 関数の性質と $\varepsilon_a, \varepsilon_r$ の値によっては, この目的が達成できないこともある. 例えば, 関数の計算精度に比べて, $\varepsilon_a, \varepsilon_r$ が小さすぎる場合には, 関数の計算回数がその上限に達しない場合でも, 丸め誤差の影響のほうが大きくなり, これ以上計算を続行しても無意味である. このような状態を見出したときは icon を 10000 にして処理を打ち切る. このときの S の精度は, 使用計算機で達成可能な限度まで達している. また関数の計算回数が nmax 回以内では収束しないこともある. このときの S の値は, それまでに得られている近似値であって, その精度は保証できない. この状態は icon が 20000 の値をとることから知ることができる.

なお, いずれの場合にも, 積分の近似値 S のほかに, その絶対誤差の推定値を引数 err に出力するので精度の目安にすることができる.

4. 使用例

助変数 p を 0.1 から 0.1 刻みに 0.9 まで変えて, 積分 $f(x) = \cos(px)$ を計算する.

```

#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

double fun(double x); /* user function prototype */
double p;

MAIN__()
{
    int ierr, icon;
    int i, n, nmin, nmax;
    double a, b, epsa, epsr, err, s;

    /* initialize data */
    a = -1;
    b = 1;
    epsa = 1e-5;
    epsr = 1e-5;
    nmin = 15;
    nmax = 511;
    printf("  icon      p          s          err          n\n");
    for (i=1; i<10; i++) {
        p = (double)i/10;
        /* calculate integral */
        ierr = c_daqc8(a, b, fun, epsa, epsr, nmin, nmax, &s, &err, &n, &icon);
        printf("%6i %6.2f %12.4e %12.4e %4i\n", icon, p, s, err, n);
    }
    return(0);
}

/* user function */
double fun(double x)
{
    return cos(p*x);
}

```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の AQC8 の項目及び[21], [130]を参照のこと.

c_daqe

1次元有限区間積分 (関数入力, 二重指数関数型積分公式)

ierr = c_daqe(a, b, fun, epsa, epsr, nmin, nmax, &s, &err, &n, &icon);

1. 機能

関数 $f(x)$ 及び $a, b, \varepsilon_a, \varepsilon_r$ が与えられたとき,

$$\left| S - \int_a^b f(x) dx \right| \leq \max \left(\varepsilon_a, \varepsilon_r \cdot \left| \int_a^b f(x) dx \right| \right) \quad (1)$$

を満たす積分の近似値 S を, 高橋・森の二重指数関数型積分公式により求める.

2. 引数

呼出し形式:

```
ierr = c_daqe(a, b, fun, epsa, epsr, nmin, nmax, &s, &err, &n, &icon);
```

引数の説明:

a	double	入力	積分区間の下限 a .
b	double	入力	積分区間の上限 b ($b \leq a$ でも可).
fun	function	入力	被積分関数 $f(x)$ を計算するユーザ関数名. プロトタイプ宣言: double fun(double x[]); 引数の説明: x double 入力 x[0]は積分変数 x に対応し, x[2]は端点からの距離が設定されている. (“使用上の注意”f)参照)
epsa	double	入力	積分の近似値 S に対する絶対誤差の上限 ε_a .
epsr	double	入力	積分の近似値 S に対する相対誤差の上限 ε_r .
nmin	int	入力	被積分関数 $f(x)$ の計算回数の下限. 標準値としては 20 が適当 (“使用上の注意”d)参照).
nmax	int	入力	被積分関数 $f(x)$ の計算回数の上限. 標準値としては 641 が適当 (これよりも大きく指定した場合は 641 と見なす). (“使用上の注意”d)参照).
s	double	出力	積分の近似値 (“使用上の注意”e)参照).
err	double	出力	積分の近似値 S の絶対誤差の推定値.
n	int	出力	被積分関数 $f(x)$ の計算回数.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
10000	丸め誤差のために要求精度が得られなかった.	s には, 得られた近似値を出力する. 精度は達成可能な限度まで達している.
11000	区間の上限の付近で関数値が急激に増大していることを表す.	要求精度をゆるめて処理を続ける. s には得られた近似値を出力する. 精度は達成可能な限度まで達している.
12000	区間の下限の付近で関数値が急激に増大していることを表す.	
13000	区間の上・下限の付近で関数値が急激に増大していることを表す.	
20000	被積分関数の計算回数がその上限に達しても要求精度が得られなかった.	処理を打ち切る. s には, そのときまでに得られている近似値を出力するが精度は保証できない.
21000~23000	コード 11000~13000 の事象が発生した後, 被積分関数の計算回数がその上限に達した.	
25000	分点のための表(作業領域)が足りなくなった.	処理を打ち切る. s には, 本関数で可能な最小の刻み幅による近似値を出力する.
30000	次のいずれかであった. • $\text{epsa} < 0$ • $\text{epsr} < 0$ • $\text{nmin} < 0$ • $\text{nmax} < \text{nmin}$	処理を打ち切る.

3. 使用上の注意

a) 定数の計算

本関数を多数回呼ぶとき, 最初だけ 641 個の定数(積分公式の分点と重みの表)を計算し, 2 回目以降は省略する. したがって, 2 回目以降は, 計算量がその分だけ少ない.

b) 関数の使用条件

本関数は, 被積分関数 $f(x)$ が, 積分区間 $[a, b]$ の端付近で変化が激しい場合に, 特に有効にはたらし, 代数特異点及び対数特異点が端点だけ(片方でも両方でもよい)に位置する場合は, 本関数の使用を第一に勧める.

積分区間内に特異点を持つ場合には, そのような点で積分領域を分割して本関数を使用するか, あるいは, 関数 c_daqn9 を用いた方がよい.

ピーク型の関数に対しては関数 c_daqn9 を用いた方がよい.

c) 関数値が無限大になる場合

本関数は積分区間の両端では関数 $f(x)$ を計算しない. したがって両端で関数値が無限大 ($f(x) \rightarrow \pm\infty$) となってもかまわない. しかし区間内に無限大となる点があってはならない.

d) 引数 nmin 及び nmax

本関数は, 被積分関数 $f(x)$ の計算回数 n を

$$nmin \leq n \leq nmax$$

の範囲に限定している。すなわち収束判定にかかわらず、 $f(x)$ は少なくとも $nmin$ 回計算され、しかも $nmax$ 回を超えて計算されることはない。もし $nmax$ 回以内に(1)を満たす S が求まらなかった場合は、 $icon$ を 20000 ~ 23000 として処理を終える。

なお、 $nmax$ が極端に小さく、例えば、 $nmax = 2$ と与えられたときは、 $f(x)$ の挙動に応じて決まる一定の値まで自動的に引き上げている。

e) 積分の近似値 S の精度

本関数は、与えられた $\varepsilon_a, \varepsilon_r$ に対して(1)を満たす S を求める。したがって、 $\varepsilon_r = 0$ とおけば絶対誤差 ε_a 以内の、また $\varepsilon_a = 0$ とおけば相対誤差 ε_r 以内の精度で積分の近似値を求めることができる。しかし、関数の性質と $\varepsilon_a, \varepsilon_r$ の値によっては、この目的が達成できないこともある。例えば、関数の計算精度に比べて、 $\varepsilon_a, \varepsilon_r$ が小さ過ぎる場合には、関数の計算回数がその上限に達しない場合でも、丸め誤差の影響の方が大きくなり、これ以上計算を続行しても無意味である。このような状態を見出したときは $icon$ を 10000 にして処理を打ち切る。このときの S の精度は使用計算機で達成可能な限度まで達している。また関数の計算回数が $nmax$ 回以内では収束しないこともある。このときの S の値は、それまでに得られている近似値であって、その精度は保証できない。この状態は $icon$ が 20000 ~ 23000 の範囲の値を取ることから知ることができる。また、関数内で定める最小の刻み幅で積分しても収束しないこともあり、このとき $icon$ は 25000 の値をとる。

なお、本関数は、いずれの場合にも、積分の近似値 S のほかに、その絶対誤差の推定値を引数 err に出力するので精度の目安にすることができる。

f) 桁落ちを防ぐためのユーザ関数の記述

例えば、積分

$$I = \int_1^3 \frac{dx}{x(3-x)^{1/4}(x-1)^{3/4}}$$

における被積分関数は区間の端点 $x = 1, 3$ で特異点を持ち、しかもその点で関数値は発散するので積分値への寄与の度合は大きい。この積分を精度よく求めるには、端点近傍の関数値を精度良く計算することが重要である。しかし、端点近傍では

$$(3.0-x) \text{ 及び } (x-1.0)$$

の計算で桁落ちを起こすため関数値を精度良く計算できない。

本関数では、この桁落ちを防ぐため、被積分関数を座標変換した形で記述できるようになっている。すなわち、ユーザ関数の引数は、以下に示す仕様を持っている。

```
double fun(double x[]);
```

この関数は、double 型の値を返し、引数 x は以下のように定義される。

```
x      double x[2]      入力      x[0]は積分変数  $x$  に対応し、
                                x[1]は  $x$  の値に依存して以下のように設定されている。
                                 $AA = \min(a, b), BB = \max(a, b)$  として、
                                 $x[1] = \begin{cases} AA - x & , AA \leq x < (AA + BB) / 2 \\ BB - x & , (AA + BB) / 2 \leq x \leq BB \end{cases}$ 
```

この $x[1]$ は端点からの正確な距離である。利用者は、この $x[1]$ の値を使って関数 $f(x)$ を記述するには、次のようにすればよい。

$$f(x) = \begin{cases} f(AA - x[1]) & , x[1] < 0 \\ f(BB - x[1]) & , x[1] \geq 0 \end{cases}$$

$x[0], x[1]$ のどちらかを使うかは利用者の自由である。

4. 使用例

積分 $\int_{-1}^1 \frac{1}{\sqrt{(1-x)(1+x)}} dx$ を計算する.

```
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

double fun(double x[]); /* user function prototype */
double p;

MAIN__()
{
    int ierr, icon;
    int n, nmin, nmax;
    double a, b, epsa, epsr, err, s;

    /* initialize data */
    a = -1;
    b = 1;
    epsa = 1e-5;
    epsr = 0;
    nmin = 20;
    nmax = 641;
    /* calculate integral */
    ierr = c_dage(a, b, fun, epsa, epsr, nmin, nmax, &s, &err, &n, &icon);
    printf("   icon    s          err          n\n");
    printf("%6i %12.4e %12.4e %4i\n", icon, s, err, n);
    return(0);
}

/* user function */
double fun(double x[])
{
    double p, res;
    p = (1+x[0])*(1-x[0]);
    res = 0;
    if (p > 0)
        res = 1.0/sqrt(p);
    return(res);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の AQE の項目及び[109], [139]を参照のこと.

c_daqeh

1次元半無限区間積分 (関数入力, 二重指数関数型積分公式)

```
ierr = c_daqeh(fun, epsa, epsr, nmin, nmax,
               &s, &err, &n, &icon);
```

1. 機能

関数 $f(x)$ 及び $\varepsilon_a, \varepsilon_r$ が与えられたとき,

$$\left| S - \int_0^\infty f(x) dx \right| \leq \max \left(\varepsilon_a, \varepsilon_r \cdot \left| \int_0^\infty f(x) dx \right| \right) \quad (1)$$

を満たす積分の近似値 S を, 高橋・森の二重指数関数型積分公式により求める.

2. 引数

呼出し形式:

```
ierr = c_daqeh(fun, epsa, epsr, nmin, nmax, &s, &err, &n, &icon);
```

引数の説明:

fun	function	入力	被積分関数 $f(x)$ を計算するユーザ関数名. プロトタイプ宣言: double fun(double x); 引数の説明: x double 入力 積分する値.
epsa	double	入力	積分の近似値 S に対する絶対誤差の上限 ε_a .
epsr	double	入力	積分の近似値 S に対する相対誤差の上限 ε_r .
nmin	int	入力	被積分関数 $f(x)$ の計算回数の下限. 標準値としては 20 が適当.
nmax	int	入力	被積分関数 $f(x)$ の計算回数の上限. 標準値としては 689 が適当 (これよりも大きく指定した場合は 689 と見なす). ("使用上の注意"d)参照).
s	double	出力	積分の近似値 ("使用上の注意"e)参照).
err	double	出力	積分の近似値 S の絶対誤差の推定値.
n	int	出力	被積分関数 $f(x)$ の計算回数.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
10000	丸め誤差のために要求精度が得られなかった.	s には, 得られた近似値を出力する. 精度は達成可能な限度まで達している.
11000	$x \rightarrow 0$ の付近で関数値が急激に増大している.	要求精度をゆるめて処理を続ける. s には得られた近似値を出力する. 精度は達成可能な限度まで達している.
12000	$x \rightarrow \infty$ のとき関数値の 0 への収束が遅い.	
13000	コード 11000 と 12000 が同時に起きた.	

コード	意味	処理内容
20000	被積分関数の計算回数がその上限に達しても要求精度が得られなかった.	処理を打ち切る. s には, そのときまでに得られている近似値を出力するが精度は保証できない.
21000~23000	コード 11000~13000 の事象が発生した後, 被積分関数の計算回数がその上限に達した.	
25000	分点のための表(作業領域)が足りなくなった.	処理を打ち切る. s には, 本関数で可能な最小の刻み幅による近似値を出力する.
30000	次のいずれかであった. • $\text{epsa} < 0$ • $\text{epsr} < 0$ • $\text{nmin} < 0$ • $\text{nmax} < \text{nmin}$	処理を打ち切る.

3. 使用上の注意

a) 定数の計算

本関数を多数回呼ぶとき, 最初だけ 689 個の定数(積分公式の分点と重みの表)を計算し, 2 回目以降は省略する.したがって, 2 回目以降は計算量がその分だけ少ない.

b) 関数の使用条件

本関数は, 被積分関数 $f(x)$ が $x \rightarrow \infty$ において 0 への収束が比較的遅いものや, ガウス・ラゲールの公式が適用できないものに対しても有効に働く.

$f(x)$ が激しく振動する場合には, 高精度な積分値を得ることができない場合がある.

c) 両端の関数値

本関数は, 積分区間の下限(原点)では関数 $f(x)$ を計算しない. したがって下限での関数値が無限大 ($f(x) \rightarrow \pm\infty$) となってもかまわない. 積分区間の上端の方では, x の大きな点での関数値を必要とするので, 特に高い要求精度を与えた場合には, ユーザ関数 fun の中では, オーバーフロー, アンダーフローに対する処置が必要である.

d) 引数 nmin 及び nmax

本関数は, 被積分関数 $f(x)$ の計算回数 n を

$$\text{nmin} \leq n \leq \text{nmax}$$

の範囲に限定している. すなわち, 収束判定にかかわらず, $f(x)$ は少なくとも nmin 回計算され, しかも nmax 回を超えて計算されることはない. もし nmax 回以内に(1)を満たす S が求まらなかった場合は, icon を 20000~23000 として処理を終える.

なお, nmax が極端に小さく, 例えば, $\text{nmax} = 2$ と与えられたときは, $f(x)$ の挙動に応じて決まる一定の値まで自動的に引き上げている.

e) 積分の近似値 S の精度

本関数は, 与えられた $\varepsilon_a, \varepsilon_r$ に対して(1)を満たす S を求める. したがって, $\varepsilon_r = 0$ とおけば絶対誤差 ε_a 以内の, また $\varepsilon_a = 0$ とおけば相対誤差 ε_r 以内の精度で積分の近似値を求めることができる. しかし, 関数の性質と $\varepsilon_a, \varepsilon_r$ の値によっては, この目的が達成できないこともある. 例えば, 関数の計算精度に比べて, $\varepsilon_a, \varepsilon_r$ が小さ過ぎる場合には, 関数の計算回数がその上限に達しない場合でも, 丸め誤差の影響の方が大きくなり, これ以上計算を続行しても無意味である. このような状態を見出したときは icon を 10000 にして処理を打ち切る. このときの S の精度は使用計算機で達成可能な限度まで達している. また関数の計算回数が nmax

回以内では収束しないことがある。このときの S の値は、それまでに得られている近似値であって、その精度は保証できない。この状態は `icon` が 20000~23000 の範囲の値をとることから知ることができる。また、関数内で定める最小の刻み幅で積分しても収束しないこともあり、このとき `icon` は 25000 の値をとる。

なお、本関数は、いずれの場合にも、積分の近似値 S のほかに、その絶対誤差の推定値を引数 `err` に出力するので精度の目安にすることができる。

4. 使用例

積分 $\int_0^{\infty} e^{-x} \sin(x) dx$ を計算する。

```
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

double fun(double x); /* user function prototype */

MAIN__()
{
    int ierr, icon;
    int n, nmin, nmax;
    double epsa, epsr, err, s;

    /* initialize data */
    epsa = 1e-5;
    epsr = 0;
    nmin = 20;
    nmax = 689;
    /* calculate integral */
    ierr = c_daqeh(fun, epsa, epsr, nmin, nmax, &s, &err, &n, &icon);
    printf("  icon    s          err          n\n");
    printf("%6i %12.4e %12.4e %4i\n", icon, s, err, n);
    return(0);
}

/* user function */
double fun(double x)
{
    double res;
    if (x > 176)
        res = 0;
    else
        res = exp(-x)*sin(x);
    return(res);
}
```

5. 手法概要

本関数は高橋・森の二重指数関数型積分公式に基づく自動積分法を用いている。

この方法の原理及び、本関数の手順については“富士通 SSL II 使用手引書”の AQE の項目の手法概要を参照のこと。ただし、ここでは、積分変数 x に対して、変換、

$$x = \phi(t) = \exp\left(\frac{3}{2}\sinh(t)\right)$$

を、したがって、重み関数 $\phi(t)$ に対して

$$\phi'(t) = \frac{3}{2}\cosh(t) \cdot \exp\left(\frac{3}{2}\sinh(t)\right)$$

を採用している。

c_daqei

1次元全無限区間積分 (関数入力, 二重指数関数型積分公式)

```
ierr = c_daqei(fun, epsa, epsr, nmin, nmax,
               &s, &err, &n, &icon);
```

1. 機能

関数 $f(x)$ 及び $\varepsilon_a, \varepsilon_r$ が与えられたとき,

$$\left| S - \int_{-\infty}^{\infty} f(x) dx \right| \leq \max(\varepsilon_a, \varepsilon_r) \cdot \left| \int_{-\infty}^{\infty} f(x) dx \right| \quad (1)$$

を満たす積分の近似値 S を, 高橋・森の二重指数関数型積分公式により求める.

2. 引数

呼出し形式:

```
ierr = c_daqei(fun, epsa, epsr, nmin, nmax, &s, &err, &n, &icon);
```

引数の説明:

fun	function	入力	被積分関数 $f(x)$ を計算するユーザ関数名. プロトタイプ宣言: double fun(double x); 引数の説明: x double 入力 積分する値.
epsa	double	入力	積分の近似値 S に対する絶対誤差の上限 ε_a .
epsr	double	入力	積分の近似値 S に対する相対誤差の上限 ε_r .
nmin	int	入力	被積分関数 $f(x)$ の計算回数の下限. 標準値としては 20 が適当.
nmax	int	入力	被積分関数 $f(x)$ の計算回数の上限. 標準値としては 645 が適当 (これよりも大きく指定した場合は 645 と見なす). ("使用上の注意"d)参照).
s	double	出力	積分の近似値 ("使用上の注意"e)参照).
err	double	出力	積分の近似値 S の絶対誤差の推定値.
n	int	出力	被積分関数 $f(x)$ の計算回数.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	丸め誤差のために要求精度が得られなかった.	s には, 得られた近似値を出力する. 精度は達成可能な限度まで達している.
11000	$x \rightarrow -\infty$ のとき関数値の 0 への収束が遅い.	要求精度をゆるめて処理を続ける. s には得られた近似値を出力する. 精度は達成可能な限度まで達している.
12000	$x \rightarrow \infty$ のとき関数値の 0 への収束が遅い.	
13000	$x \rightarrow \pm\infty$ のとき関数値の 0 への収束が遅い.	

コード	意味	処理内容
20000	被積分関数の計算回数がその上限に達しても要求精度が得られなかった.	処理を打ち切る. s には, そのときまでに得られている近似値を出力するが精度は保証できない.
21000~23000	コード 11000~13000 の事象が発生した後, 被積分関数の計算回数がその上限に達した.	
25000	分点のための表(作業領域)が足りなくなった.	処理を打ち切る. s には, 本関数で可能な最小の刻み幅による近似値を出力する.
30000	次のいずれかであった. • $\text{epsa} < 0$ • $\text{epsr} < 0$ • $\text{nmin} < 0$ • $\text{nmax} < \text{nmin}$	処理を打ち切る.

3. 使用上の注意

a) 定数の計算

本関数を多数回呼ぶとき, 最初だけ 645 個の定数 (積分公式の分点と重みの表) を計算し, 2 回目以降は省略する. したがって, 2 回目以降は計算量がその分だけ少ない.

b) 関数の使用条件

本関数は, 被積分関数 $f(x)$ が $x \rightarrow \pm\infty$ において 0 への収束が比較的遅いものや, ガウス・エルミートの公式が適用できないものに対しても有効にはたらく.

$f(x)$ が原点付近で高いピークを持つもの, 激しく振動するもの場合には, 高精度な積分値を得ることができない場合がある.

c) 両端の関数値

本関数は, 積分変数 x の絶対値の大きな点での関数値 $f(x)$ を必要とするので, 殊に高い要求精度を与えた場合には, ユーザ関数 fun の中では, このような場合のオーバーフロー, アンダーフローに対する処理が必要となる.

d) 引数 nmin 及び nmax

本関数は, 被積分関数 $f(x)$ の計算回数 n を

$$\text{nmin} \leq n \leq \text{nmax}$$

の範囲に限定している. すなわち, 収束判定にかかわらず, $f(x)$ は少なくとも nmin 回計算され, しかも nmax 回を超えて計算されることはない. もし nmax 回以内に(1)を満たす S が求まらなかった場合は, icon を 20000~23000 として処理を終える.

なお, nmax が極端に小さく, 例えば, $\text{nmax} = 2$ と与えられたときは, $f(x)$ の挙動に応じて決まる一定の値まで自動的に引き上げている.

e) 積分の近似値 S の精度

本関数は, 与えられた $\varepsilon_a, \varepsilon_r$ に対して(1)を満たす S を求める. したがって, $\varepsilon_r = 0$ とおけば絶対誤差 ε_a 以内の, また $\varepsilon_a = 0$ とおけば相対誤差 ε_r 以内の精度で積分の近似値を求めることができる. しかし, 関数の性質と $\varepsilon_a, \varepsilon_r$ の値によっては, この目的が達成できないこともある. 例えば, 関数の計算精度に比べて, $\varepsilon_a, \varepsilon_r$ が小さ過ぎる場合には, 関数の計算回数とその上限に達しない場合でも, 丸め誤差の影響の方が大きくなり, これ以上計算を続行しても無意味である. このような状態を見出したときは icon を 10000 にして処理を打ち

切る. このときの S の精度は使用計算機で達成可能な限度まで達している. また関数の計算回数が $n_{\max}$ 回以内では収束しないことがある. このときの S の値は, それまでに得られている近似値であって, その精度は保証できない. この状態は $icon$ が 20000~23000 の範囲の値をとることから知ることができる. また, 関数内で定める最小の刻み幅で積分しても収束しないこともあり, このとき $icon$ は 25000 の値をとる.

なお, 本関数は, いずれの場合にも, 積分の近似値 S のほかに, その絶対誤差の推定値を引数 err に出力するので精度の目安にすることができる.

4. 使用例

積分 $\int_{-\infty}^{\infty} \frac{1}{10^{-2} + x^2} dx$ の値を計算する.

```
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

double fun(double x); /* user function prototype */

MAIN__()
{
    int ierr, icon;
    int n, nmin, nmax;
    double epsa, epsr, err, s;

    /* initialize data */
    epsa = 1e-3;
    epsr = 0;
    nmin = 20;
    nmax = 645;
    /* calculate integral */
    ierr = c_daqei(fun, epsa, epsr, nmin, nmax, &s, &err, &n, &icon);
    printf("    icon    s        err        n\n");
    printf("%6i %12.4e %12.4e %4i\n", icon, s, err, n);
    return(0);
}

/* user function */
double fun(double x)
{
    double res;
    if (fabs(x) > 1e35)
        res = 0;
    else if (fabs(x) < 1e-35)
        res = 100;
    else
        res = 1/(1e-2+x*x);
    return(res);
}
```

5. 手法概要

本関数は高橋・森の二重指数関数型積分公式に基づく自動積分法を用いている.

この方法の原理及び, 本関数の手順については“富士通 SSL II 使用手引書”の AQE の項目の手法概要を参照のこと. ただし, ここでは, 積分変数 x に対して, 変換,

$$x = \phi(t) = \sinh\left(\frac{3}{2}\sinh(t)\right)$$

を, したがって, 重み関数 $\phi(t)$ に対して

$$\phi'(t) = \frac{3}{2}\cosh(t) \cdot \cosh\left(\frac{3}{2}\sinh(t)\right)$$

を採用している.

c_daqmc8

多次元有限領域積分 (関数入力, クレンシヨウ・カーチス型積分法)
<pre>ierr = c_daqmc8(m, lsub, fun, epsa, epsr, nmin, nmax, &s, &err, &n, &icon);</pre>

1. 機能

m 重積分($1 \leq m \leq 3$)を次で定義する.

$$I = \int_{\phi_1}^{\phi_1} dx_1 \int_{\phi_2}^{\phi_2} dx_2 \dots \int_{\phi_m}^{\phi_m} dx_m f(x_1, x_2, \dots, x_m)$$

ただし, 積分の下限, 上限は,

$$\begin{aligned} \phi_1 &= a(\text{定数}) & \phi_1 &= b(\text{定数}) \\ \phi_2 &= \phi_2(x_1) & \phi_2 &= \phi_2(x_1) \\ &\vdots & &\vdots \\ \phi_m &= \phi_m(x_1, x_2, \dots, x_{m-1}) & \phi_m &= \phi_m(x_1, x_2, \dots, x_{m-1}) \end{aligned}$$

で与えられるものとする. このとき, 与えられた $\varepsilon_a, \varepsilon_r$ に対して,

$$|S - I| \leq \max(\varepsilon_a, \varepsilon_r |I|) \quad (1)$$

を満たす積分の近似値 S を, 各次元に等差数列的に分点を増すクレンシヨウ・カーチス型積分法を繰り返し適用することにより求める.

2. 引数

呼出し形式:

```
ierr = c_daqmc8(m, lsub, fun, epsa, epsr, nmin, nmax, &s, &err, &n, &icon);
```

引数の説明:

m	int	入力	積分の次元数. $1 \leq m \leq 3$.
lsub	function	入力	各積分変数の下限 ϕ_k 及び上限 ϕ_k を計算するユーザ関数名. プロトタイプ宣言: void lsub(int k, double x[], double *a, double *b);
引数の説明:			
k	int	入力	座標軸の番号 k . $1 \leq k \leq m$.
x	double x[m-1]	入力	積分する値 $x_1, x_2, \dots, x_{m-1}$ が, それぞれ x[0], x[1], ..., x[m-2] なる対応を持つ 大きさ m-1 の 1 次元配 列.
a	double	出力	下限 ϕ_k の値.
b	double	出力	上限 ϕ_k の値.

コンディションコード:

111

コード	意味	処理内容
30000	次のいずれかであった. • $\text{epsa} < 0$ • $\text{epsr} < 0$ • $\text{nmin} < 0$ • $\text{nmax} < \text{nmin}$ • $m \leq 0$ 又は $m \geq 4$	処理を打ち切る.

3. 使用上の注意

a) 定数の計算

本関数を多数回呼ぶとき, 最初だけ 511 個の定数(積分公式の分点と重みの表)を計算し, 2 回目以降は省略する. したがって, 2 回目以降は計算量がその分だけ少ない.

b) 関数の使用条件

関数は, 被積分関数 $f(x_1, x_2, \dots, x_m)$ が滑らかな場合はもちろん, 振動型関数の場合にも有効に働く.

c) 引数 nmin 及び nmax

本関数は, 被積分関数の各座標軸方向での計算回数 n_i ($i = 1, 2, \dots, m$) を

$$\text{nmin} \leq n_i \leq \text{nmax}$$

の範囲に限定している. すなわち収束判定にかかわらず $f(x_1, x_2, \dots, x_m)$ は各座標軸方向に少なくとも nmin 回計算され, しかも nmax 回を超えて計算されることはない. もし, ある座標軸方向で nmax 回以内に積分が収束しなかった場合は, icon を 200~22200 の値とする. ここで, 数字の 2 が現れる位置は, 座標軸の区別を意味し, 100, 1000, 及び 10000 の位の順に, x_3 軸, x_2 軸, 及び x_1 軸と対応する.

なお, nmax が 7 未満の値であったときは, 7 が与えられたものと見なす.

d) 積分の近似値 S の精度

本関数は, 与えられた $\varepsilon_a, \varepsilon_r$ に対して (1) を満たす S を求める. したがって, $\varepsilon_r = 0$ とおけば絶対誤差 ε_a 以内の, また $\varepsilon_a = 0$ とおけば相対誤差 ε_r 以内の精度で積分の近似値を求めることになる. しかし, 関数の性質と $\varepsilon_a, \varepsilon_r$ の値によっては, この目的が達成できないこともある. 例えば, 関数の計算精度に比べて, $\varepsilon_a, \varepsilon_r$ が小さ過ぎる場合には, 関数のある軸方向の計算回数がある上限(nmax)に達しない場合でも丸め誤差の影響の方が大きくなり, これ以上計算を続行しても無意味な状態を見出すことがある. このとき icon を 100~11100 とする. ここで数字の 1 が現れる位置は, 座標軸の区別を意味し, 100, 1000, 及び 10000 の位の順に, x_3 軸, x_2 軸, 及び x_1 軸と対応する. icon が 100~11100 のとき, すなわち, x_3 軸, x_2 軸だけで丸め誤差の影響が大きくなったときでも, 全体の積分値 S の精度にそれほど影響を及ぼさないことがあり, 得られた積分値が要求精度を満足していることもある. しかし, この確認は, 誤差の推定値 err に基づいて行われるべきである.

また, ある座標軸方向の積分が nmax 回以内の関数計算では収束しないこともある. このとき icon は 200~22200 の値をとり, それぞれの 2 の位置は座標軸の区別を意味する. icon が 200~2200 のときは, 得られた積分値が要求精度を満足していることもある.

更に, コード 100~11100 とコード 200~22200 の事象が同時に起きたときは, icon を 300~33300 とする.

なお, 本関数は, いずれの場合にも, 積分の近似値 S のほかに, 絶対誤差の推定値を引数 err に出力するので精度の目安にすることができる.

4. 使用例

助変数 p を 1 から 3 まで 1 刻みに変えて、次の積分の値を計算する。

$$\int_{-1}^1 \int_{-2}^2 \int_{-3}^3 \frac{1}{\cos(px)\cos(py)\cos(pz)} dz dy dx$$

```
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

/* function prototypes */
void lsub(int k, double x[], double *a, double *b);
double fun(double x[]);

double p;

MAIN__()
{
    int ierr, icon;
    int i, m, n, nmin, nmax;
    double epsa, epsr, err, s;

    /* initialize data */
    epsa = 1e-5;
    epsr = 1e-5;
    nmin = 7;
    nmax = 511;
    m = 3;
    printf("p      icon      s          err          n\n");
    for (i=1;i<4;i++) {
        p = (double)i;
        /* calculate integral */
        ierr = c_daqmc8(m, lsub, fun, epsa, epsr,
                       nmin, nmax, &s, &err, &n, &icon);
        printf("%3.0f %6i %12.4e %12.4e  %4i\n", p, icon, s, err, n);
    }
    return(0);
}

/* limits function */
void lsub(int k, double x[], double *a, double *b)
{
    switch (k) {
    case(1):
        *a = -1;
        *b = 1;
        break;
    case(2):
        *a = -2;
        *b = 2;
        break;
    case(3):
        *a = -3;
        *b = 3;
        break;
    }
}

/* user function */
double fun(double x[])
{
    double res;
    res = 1/(cos(p*x[0])*cos(p*x[1])*cos(p*x[2])+2);
    return(res);
}
```

5. 手法概要

本関数は、等差数列的(公差 8)に分点を追加する拡張された意味のクレンショー・カーチス型積分法を、それぞれの次元方向に繰り返す直積型多重積分法を用いている。

この方法の原理及び、本関数の手順については“富士通 SSL II 使用手引書”の AQC8, AQMC8 の項目の手法概要を参照のこと。

c_daqme

多次元積分 (関数入力, 二重指数関数型積分公式)

```
ierr = c_daqme(m, intv, lsub, fun, epsa, epsr,
               nmin, nmax, &s, &err, &n, &isf,
               &icon);
```

1. 機能

m 重積分 ($1 \leq m \leq 3$) を次で定義する.

$$I = \int_{\phi_1}^{\psi_1} \int_{\phi_2}^{\psi_2} \dots \int_{\phi_m}^{\psi_m} f(x_1, x_2, \dots, x_m) dx_m dx_{m-1} \dots dx_1.$$

ただし, 積分の下限, 上限は, 一般に,

$$\begin{aligned} \phi_1 &= a & \psi_1 &= b \\ \phi_2 &= \phi_2(x_1) & \psi_2 &= \psi_2(x_1) \\ &\dots & &\dots \\ \phi_m &= \phi_m(x_1, x_2, \dots, x_{m-1}) & \psi_m &= \psi_m(x_1, x_2, \dots, x_{m-1}) \end{aligned}$$

で与えられるが, 本関数は, このような有限領域のほかに, 半無限, あるいは全無限領域も扱う. すなわち, 各積分変数 x_k についての積分区間 $[\phi_k, \psi_k]$ は独立に $[0, \infty)$ あるいは $(-\infty, \infty)$ であってよいものとする.

以上の条件のもとに, 本関数は, 与えられた $\varepsilon_a, \varepsilon_r$ に対して,

$$|S - I| \leq \max(\varepsilon_a, \varepsilon_r |I|) \quad (1)$$

を満たす積分の近似値 S を, 高橋・森の 2 重指数関数型積分公式を繰り返し適用することにより求める.

2. 引数

呼出し形式:

```
ierr = c_daqme(m, intv, lsub, fun, epsa, epsr, nmin, nmax, &s, &err, &n, &isf,
               &icon);
```

引数の説明:

m	int	入力	積分の次元数 m .
intv	int intv[m]	入力	各積分変数についての積分区間の種類を表す情報. intv[k-1] は, k 番目の変数 x_k についての積分区間の種類を表すものとし, 以下の約束にしたがって, 1, 2, 3 のいずれかの整数を与える 1 ... 有限区間 2 ... 半無限区間 3 ... 全無限区間 例えば, 3 重積分 $I = \int_0^\infty \int_0^\pi \int_0^{2\pi} f(x_1, x_2, x_3) dx_3 dx_2 dx_1$ の場合は, intv[0] = 2, intv[1] = 1, intv[2] = 1 と与える.
lsub	function	入力	各積分変数の下限 ϕ_k 及び上限 ψ_k を計算する関数名. プロトタイプ宣言: void lsub (int k, double x[], double *a, double *b); 引数の説明: k int 入力 座標軸の番号 k . $1 \leq k \leq m$.

			x double 入力 積分変数. x[m-1] x[k-1] = $x_k, k = 1, 2, \dots, m-1$.
			a double 出力 下限 $\phi_k(x_1, x_2, \dots, x_{k-1})$ の値. b double 出力 上限 $\psi_k(x_1, x_2, \dots, x_{k-1})$ の値. ただし, 区間 $[\phi_k, \psi_k]$ が $[0, \infty)$ あるいは $(-\infty, \infty)$ の場合は, その k については, a, b に何の値も設定する必要はない.
fun	function	入力	被積分関数 $f(x_1, x_2, \dots, x_m)$ を計算する関数名. プロトタイプ宣言: double fun (double x[]); 引数の説明: x double 入力 積分変数. x[m] x[k-1] = $x_k, k=1, 2, \dots, m$. (“使用上の注意”b)参照)
epsa	double	入力	積分の近似値 S に対する絶対誤差の上限 $\varepsilon_a (\geq 0)$. (“使用上の注意”d)参照)
epsr	double	入力	積分の近似値 S に対する相対誤差の上限 $\varepsilon_r (\geq 0)$. (“使用上の注意”d)参照)
nmin	int	入力	各積分変数ごとに積分する際の被積分関数の計算回数の下限 (≥ 0). 標準値としては 20 が適当. (“使用上の注意”c)参照)
nmax	int	入力	各積分変数ごとに積分する際の被積分関数の計算回数の上限 ($\geq nmin$). 標準値としては 705 が適当 (これよりも大きく指定した場合は 705 と見なす). (“使用上の注意”c)参照)
s	double	出力	積分の近似値. (“使用上の注意”d)参照)
err	double	出力	積分の近似値 s の絶対誤差の推定値. (“使用上の注意”d)参照)
n	int	出力	被積分関数の全体としての計算回数.
isf	int	出力	icon の出力時の値が 25000 台であったとき, 被積分関数の悪条件な振舞についての情報. isf は 10 進 3 桁の正整数であり, いま, $isf = 100j_1 + 10j_2 + j_3,$ と置くと, j_3, j_2, j_1 はそれぞれ x_3 軸, x_2 軸, 及び, x_1 軸方向における被積分関数の振舞を表す. 各 j_i は独立に, 1, 2, 3 又は 0 のいずれかをとり, その意味は次のとおりである. isf = 1 積分区間の下限の付近で関数値が急激に増大しているか, あるいは, 区間が無限区間であるならば, $x_i \rightarrow -\infty$ のとき関数値の 0 への収束が遅い. isf = 2 積分区間の上限の付近で関数値が急激に増大しているか, あるいは, 区間が半無限もしくは無限区間であるならば, $x_i \rightarrow +\infty$ のとき関数値の 0 への収束が遅い. isf = 3 上記 1, 2 の事象が共に生じた. isf = 0 上記の事象は生じなかった.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
10001～10077	x_3 軸方向, 又は x_2 軸方向の積分において以下に示す事象が起きた. 1 の位は x_3 軸方向, 10 の位は x_2 軸方向の事象を表し, 各々独立に 0 ～7 の値をとる. (ただし, 共に 0 をとることはない). 1. 丸め誤差のため, その軸方向の要求精度が得られなかった. 2. その軸方向での被積分関数の計算回数が上限 (nmax) に達しても要求精度が得られなかった. 4. 関数内で定める最小の刻み幅で積分しても, その軸方向の要求精度が得られなかった. 3. 上記 1, 2 の事象が共に起きた. 5. 上記 1, 4 の事象が共に起きた. 6. 上記 2, 4 の事象が共に起きた. 7. 上記 1, 2, 4 の事象が共に起きた. 0. 上記のどの事象も起きなかった.	s には得られた近似値を出力する. 精度は要求精度を満たしていることもある.
10100～10177	x_1 軸方向の積分において, 丸め誤差のため要求精度が得られなかった. 下位 2 桁の意味は上記と同じである..	s には得られた近似値を出力する. 精度は達成可能な限度に達している.
20200～20277	x_1 軸方向の積分において, 被積分関数の計算回数が上限(nmax) に達しても要求精度が得られなかった. 下位 2 桁の意味は上記と同じである.	s には得られた近似値を出力するが, 精度は保証できない. nmax の値を大きく与えれば, 精度の改善の余地がある (nmax=705 が限度).
20400～20477	x_1 軸方向の積分において, 関数内で定める最小の刻み幅で積分しても要求精度が得られなかった. 下位 2 桁の意味は上記と同じである.	s には得られた近似値を出力する.
25000～25477	ある座標軸方向で積分しているとき, 積分区間の下限若しくは上限の付近で関数値が急激に増大している. あるいは, 積分区間が半無限若しくは無限区間の場合, 積分変数が無限遠点へ進むときの被積分関数の 0 への収束が遅い. 下位 3 桁の意味は上記と同じである.	要求精度をゆるめて処理を続ける. s には得られた近似値を出力する. 与えられた積分が, 理論的な意味で積分値が存在しない (積分可能でない) 場合も, この範囲のコードが立つので注意すること. なお, 被積分関数のこのような振舞についての詳細は引数 isf に出力するので必要ならば参照すること.
30000	次のいずれかであった: • $\text{epsa} < 0$ • $\text{epsr} < 0$ • $\text{nmin} < 0$ • $\text{nmax} < \text{nmin}$ • $m \leq 0$ 又は $m \geq 4$ • intv のある要素に, 1, 2, 3 以外の値が入力された.	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

本関数を多数回呼ぶとき、最初だけ定数表(積分公式の分点と重みの表)を計算し、2回目以降は省略する。したがって、2回目以降は計算量がその分だけ少ない。

本関数は、積分領域の境界付近で変化が激しい場合に、特に有効に働き、代数特異点及び対数特異点が境界面上に位置する場合は、本関数の使用を第一に勧める。積分領域が有限で、被積分関数が滑らかな関数及び振動型の関数の場合には関数 `c_daqmc8` を用いた方がよい。

積分領域が無限の場合で、被積分関数 $f(x_1, x_2, \dots, x_m)$ が $x_i \rightarrow \pm\infty$ において、0 への収束が比較的遅いものに対してても有効に働く。ただし、領域内部で激しく振動する場合には、高精度な積分値を得ることができない場合がある。

b) fun

本関数は、積分領域の境界面上では関数 `fun` を計算しない。したがって、その面上で関数値が無限大となってもかまわない。ただし、領域内に無限大となる点があってはならない。

ある座標軸方向 (i 番目の座標軸とする) の積分区間が無限である場合、 $|x_i|$ の大きな点での関数値 $f(x_1, x_2, \dots, x_m)$ を必要とするので、殊に高い要求精度を与えた場合には、`fun` の中では、オーバフロー、アンダフローに対する処置が必要である。

c) nmin 及び nmax

本関数は、被積分関数の各座標軸方向での計算回数 n_i ($i = 1, 2, \dots, m$) を

$$nmin \leq n_i \leq nmax.$$

の範囲に限定している。すなわち収束判定にかかわらず $f(x_1, x_2, \dots, x_m)$ は各座標軸方向に少なくとも `nmin` 回計算され、しかも `nmax` 回を超えて計算されることはない。もし、ある座標軸方向で `nmax` 回以内に積分が収束しなかった場合は、その軸が x_1 軸、 x_2 軸、 x_3 軸のいずれであるかに応じて `icon` の 100, 10, 1 の位にその情報を出力する。

なお、`nmax` が極端に小さく、例えば `nmax=2` と与えられたときは、被積分関数の挙動に応じて決まる一定の値まで自動的に引き上げている。

d) 積分の近似値 `s` の精度

本関数は、与えられた ε_a , ε_r に対して、(1) を満たす `s` を求める。したがって $\varepsilon_r=0$ とおけば、絶対誤差 ε_a 以内の、また $\varepsilon_a=0$ とおけば相対誤差 ε_r 以内の精度で積分の近似値を求めることになる。しかし、関数の性質と ε_a , ε_r の値によっては、この目的が達成できないこともある。例えば、関数の計算精度に比べて ε_a , ε_r が小さすぎる場合には、関数のある軸方向の計算回数がその上限 (`nmax`) に達しない場合でも、丸め誤差の影響の方が大きくなり、それ以上計算を実行しても無意味な状態を見出すことがある。このとき軸の種類に応じて `icon` の 100, 10, 1 の位にその情報を出力する。一般に、 x_3 軸、 x_2 軸だけで丸め誤差の影響が大きくなったときでも、全体の積分値 `s` の精度にそれほど影響を及ぼさないことがあり、得られた積分値が要求精度を満足していることもある。しかし、この確認は、誤差の推定値 `err` に基づいて行われるべきである。

また、ある座標軸方向の積分が `nmax` 回以内の関数計算では収束しないこともある。このときも、`icon` に情報を出力するが、 x_3 軸、 x_2 軸だけでこの事象が起きたときでも、得られた積分値が要求精度を満足していることもある。

また、関数内で定める最小の刻み幅で積分しても収束しないことがある。これについても、`icon` に出力するが、以前と同様に x_3 軸、 x_2 軸だけでこの事象が起きても、要求精度を満足していることもある。

本関数は、いずれの場合も、積分の近似値 `s` のほかに、絶対誤差の推定値を引数 `err` に出力するので精度の目安にすることができる。

4. 使用例

次の積分 I を計算する.

$$I = \int_0^{\infty} dx_1 \int_0^{x_1} dx_2 \int_0^{1-x_2} \frac{e^{-x_1}}{x_1 \sqrt{x_2 + x_3}} dx_3$$

```
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

/* function prototypes */
void lsub(int k, double x[], double *a, double *b);
double fun(double x[]);

MAIN__()
{
    int ierr, icon;
    int m, n, nmin, nmax, intv[3], isf;
    double epsa, epsr, err, s;

    /* initialize data */
    epsa = 1e-3;
    epsr = 1e-3;
    nmin = 20;
    nmax = 705;
    m = 3;
    intv[0] = 2;
    intv[1] = 1;
    intv[2] = 1;
    /* calculate integral */
    ierr = c_daqme(m, intv, lsub, fun, epsa, epsr,
                  nmin, nmax, &s, &err, &n, &isf, &icon);
    printf("icon = %i s = %12.4e err = %12.4e isf = %i n = %i\n",
          icon, s, err, isf, n);
    return(0);
}

/* limits function */
void lsub(int k, double x[], double *a, double *b)
{
    *a = 0;
    switch (k) {
        case(1):
            break;
        case(2):
            *b = x[0];
            break;
        case(3):
            *b = 1-x[1];
            break;
    }
}

/* user function */
double fun(double x[])
{
    double y;
    y = x[1]+x[2];
    if (y < 1e-70) return 0;
    if (x[0] > 174) return 0;
    y = x[0]*sqrt(y);
    if (y < 1e-70) return 0;
    return exp(-x[0])/y;
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の AQME の項目を参照のこと.

c_daqn9

1次元有限区間積分 (関数入力, 適応型ニュートン・コーツ9点則)

<pre>ierr = c_daqn9(a, b, fun, epsa, epsr, nmin, nmax, &s, &err, &n, &icon);</pre>

1. 機能

関数 $f(x)$ 及び $a, b, \varepsilon_a, \varepsilon_r$ が与えられたとき,

$$\left| S - \int_a^b f(x) dx \right| \leq \max \left(\varepsilon_a, \varepsilon_r \cdot \left| \int_a^b f(x) dx \right| \right) \quad (1)$$

を満たす積分の近似値 S を, ニュートン・コーツ9点則に基づく適応型手法により求める.

2. 引数

呼出し形式:

```
ierr = c_daqn9(a, b, fun, epsa, epsr, nmin, nmax, &s, &err, &n, &icon);
```

引数の説明:

a	double	入力	積分区間の下限 a .
b	double	入力	積分区間の上限 b ($b \leq a$ でも可).
fun	function	入力	被積分関数 $f(x)$ を計算するユーザ関数名. プロトタイプ宣言: double fun(double x); 引数の説明: x double 入力 積分する値.
epsa	double	入力	積分の近似値 S に対する絶対誤差の上限 ε_a .
epsr	double	入力	積分の近似値 S に対する相対誤差の上限 ε_r .
nmin	int	入力	被積分関数の計算回数の下限. 標準値としては 21 が適当. $0 \leq nmin < 150$. ("使用上の注意" c) 参照).
nmax	int	入力	被積分関数の計算回数の上限. 標準値としては 2000 が適当. $nmax > nmin + 8$. ("使用上の注意" c) 参照).
s	double	出力	積分の近似値 ("使用上の注意" d) 参照).
err	double	出力	積分の近似値 S の絶対誤差の推定値.
n	int	出力	被積分関数の計算回数.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000～13111	特異点など異常点を検出した. 次の異常点があったことを表す. 1000 の位が, 1 のときは, 代数特異点. 2 のときは, コーシー特異点. 3 のときは, その両方. 100 の位は対数特異点. 10 の位は不連続点. 1 の位はその他の異常点.	処理を続ける. 代数特異点, 対数特異点, 不連続点だけの場合は, s の値は要求精度を満たしている場合が多い. しかし, その他の異常点の場合は要求精度は得られていない.
20000～23111	被積分関数の計算回数がその上限に達しても要求精度が得られなかった. 1000 位以下の意味は, コード 10000～13111 の場合と同じ.	処理を打ち切る. s にはそのときまでに得られている近似値を出力するが, 精度は保証できない.
30000	次のいずれかであった. • $\text{epsa} < 0$ • $\text{epsr} < 0$ • $\text{nmin} < 0$ • $\text{nmin} \geq 150$ • $\text{nmax} \leq \text{nmin}+8$	処理を打ち切る.

3. 使用上の注意

a) 関数の使用条件

本関数は, 広範囲な関数に対して使用することができ, 特に, 被積分関数 $f(x)$ がピーク型の関数である場合に最も有効である. 更に $f(x)$ が代数特異点, 対数特異点, 不連続点などの異常点を積分区間 $[a, b]$ の端, 中点, 4 等分点など, 2 等分割を繰り返すことにより早期に到達できる点に持つ場合でも有効にはたらく. したがって, 性質のよく分からない関数をはじめ, ピーク型の関数及び積分区間内に特異点をもつ関数に対しては本関数の使用を第一に勧める. ただし, 精度を上げるためには, これらの特異点を積分区間の端で, しかも原点に位置するように, 必要ならば適当に変数変換することが望ましい.

特異点が積分区間の端点だけに位置する場合は関数 c_daqe を使用した方が効率の面で良い.

b) 被積分関数の値が無限大になる場合

積分区間 $[a, b]$ 内のある点で, 被積分関数 $f(x)$ が無限大 ($f(x) \rightarrow \pm\infty$) となる場合は, その点での $f(x)$ の値を適当な有限値に置き換える(例えば 0 と置く)ことが必要である.

c) 引数 nmin 及び nmax

本関数は, 被積分関数 $f(x)$ の計算回数 n を

$$\text{nmin} \leq n \leq \text{nmax}$$

範囲に限定している. すなわち, 収束判定にかかわらず, $f(x)$ は少なくとも nmin 回計算され, しかも nmax 回を超えて計算されることはない. もし nmax 回以内に(1)を満たす S が求まらなかった場合は, icon を 20000～21111 として処理を終える. なお, nmax が 21 未満の値であったときには, 21 が与えられたものと見なす.

d) 積分の近似値 S の精度

本関数は, 与えられた $\varepsilon_a, \varepsilon_r$ に対して(1)を満たす S を求める. したがって, $\varepsilon_r = 0$ とおけば絶対誤差 ε_a 以内の, また $\varepsilon_a = 0$ とおけば相対誤差 ε_r 以内の精度で積分の近似値を求めることになる. しかし, 関数の性質と

$\varepsilon_a, \varepsilon_r$ の値によっては、この目的が達成できないことがある。例えば、関数の計算精度に比べて $\varepsilon_a, \varepsilon_r$ が小さ過ぎる場合にはこのようなことが起こり、ユーザ関数の計算回数が多くなって、nmax 回以内では収束しないことがある。このときの S の値は、それまでに得られている近似値であってその精度は保証できない。この状態は icon が 20000~23111 の範囲の値をとることから知ることができる。

なお、本関数は、いずれの場合にも、積分の近似値 S のほかに、その絶対誤差の推定値を引数 err に出力するので精度の目安にすることができる。

4. 使用例

助変数 p を 0.1 から 0.9 まで 0.1 刻みに変化させて、積分 $\int_0^1 (x^{-p} + \sin(px))dx$ の値を計算する。

```
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

double fun(double x); /* user function prototype */
double p;

MAIN__()
{
    int ierr, icon;
    int i, n, nmin, nmax;
    double a, b, epsa, epsr, err, s;

    /* initialize data */
    a = 0;
    b = 1;
    epsa = 1e-4;
    epsr = 1e-4;
    nmin = 21;
    nmax = 2000;
    printf("p      icon      s          err          n\n");
    for (i=1; i<10; i++) {
        p = (double)i/10;
        /* calculate integral */
        ierr = c_daqn9(a, b, fun, epsa, epsr, nmin, nmax, &s, &err, &n, &icon);
        printf("%3.1f %6i %12.4e %12.4e %4i\n", p, icon, s, err, n);
    }
    return(0);
}

/* user function */
double fun(double x)
{
    double res;
    res = 0;
    if (x > 0)
        res = pow(x, -p) + sin(p*x);
    return(res);
}
```

5. 手法概要

本関数はニュートン・コーツ 9 点則に基づく適応型自動積分法を用いている。

この方法の原理及び、本関数の手順については“富士通 SSL II 使用手引書”の AQN9 の項目の手法概要及び [76], [134]を参照のこと。

c_dassm

行列の和 (実対称行列)
<code>ierr = c_dassm(a, b, c, n, &icon);</code>

1. 機能

$n \times n$ の実対称行列 **A** と実対称行列 **B** の和 **C** を求める.

$$\mathbf{C} = \mathbf{A} + \mathbf{B} \quad (1)$$

ここで, **C** は $n \times n$ の実対称行列である. $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dassm(a, b, c, n, &icon);
```

引数の説明:

a	double a[<i>Alen</i>]	入力	行列 A . 対称行列用圧縮格納法. 格納法の詳細については“1.3.2 対称行列の格納方法”を参照のこと. $Alen = n(n+1)/2$.
b	double b[<i>Blen</i>]	入力	行列 B . 対称行列用圧縮格納法. 格納法の詳細については“1.3.2 対称行列の格納方法”を参照のこと. $Blen = n(n+1)/2$.
c	double c[<i>Clen</i>]	入力	行列 C . 対称行列用圧縮格納法. 格納法の詳細については“1.3.2 対称行列の格納方法”を参照のこと. $Clen = n(n+1)/2$. (“使用上の注意”参照)
n	int	入力	行列 A , B 及び C の次数 n .
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
30000	$n < 1$	処理を打ち切る.

3. 使用上の注意

a) 領域の節約

配列 a あるいは b 上の内容を保存する必要のない場合は, 次のように呼び出すことにより領域が節約できる.

配列 a の内容が不要の場合:

```
ierr = c_dassm(a, b, a, n, &icon);
```

配列 b の内容が不要の場合:

```
ierr = c_dassm(a, b, b, n, &icon);
```

この場合, 行列 **C** は配列 a あるいは, b 上に得られる.

4. 使用例

実対称行列の和を求め, 結果を確認する.

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

```
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 100

MAIN__()
{
    int ierr, icon;
    int n, i, j, ij;
    double eps, err;
    double a[NMAX*(NMAX+1)/2], b[NMAX*(NMAX+1)/2], c[NMAX*(NMAX+1)/2];

    /* initialize matrices*/
    n = NMAX;
    ij = 0;
    for (i=0;i<n;i++)
        for (j=0;j<=i;j++) {
            a[ij] = i-j+1;
            b[ij++] = n-i+j-1;
        }
    /* add matrices */
    ierr = c_dassm(a, b, c, n, &icon);
    if (icon != 0) {
        printf("ERROR: c_dassm failed with icon = %d\n", icon);
        exit(1);
    }
    /* check matrix */
    eps = 1e-6;
    ij = 0;
    for (i=0;i<n;i++)
        for (j=0;j<=i;j++) {
            err = fabs((c[ij++]-n)/n);
            if (err > eps) {
                printf("WARNING: result inaccurate\n");
                exit(1);
            }
        }
    printf("Result OK\n");
    return(0);
}
```

c_dasvd1

実行列の特異値分解 (ハウスホルダー法, QR 法)

```
ierr = c_dasvd1(a, ka, m, n, isw, sig, u, ku,
                v, kv, vw, &icon);
```

1. 機能

$m \times n$ の実行列 $\mathbf{A}$ をハウスホルダー法, 及び QR 法により特異値分解する.

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

ここで, $l = \min(m, n)$ として, $\mathbf{U}$ 及び $\mathbf{V}$ は, $m \times l$ 及び $n \times l$ の行列であり,

$l = n$ ($m \geq n$) のとき,

$$\mathbf{U}^T\mathbf{U} = \mathbf{V}^T\mathbf{V} = \mathbf{V}\mathbf{V}^T = \mathbf{I}_n$$

$l = m$ ($m < n$) のとき,

$$\mathbf{U}^T\mathbf{U} = \mathbf{U}\mathbf{U}^T = \mathbf{V}^T\mathbf{V} = \mathbf{I}_m$$

である. Σ は $\Sigma = \text{diag}(\sigma_i)$, $\sigma_i \geq 0$ なる 1 次の対角行列であり, σ_i を $\mathbf{A}$ の特異値という. また, 特異値 σ_i は行列 $\mathbf{A}^T\mathbf{A}$ の固有値の正平方根であり, $\mathbf{V}$ の第 i 列は対応する固有ベクトルである. $m \geq 1, n \geq 1$ であること.

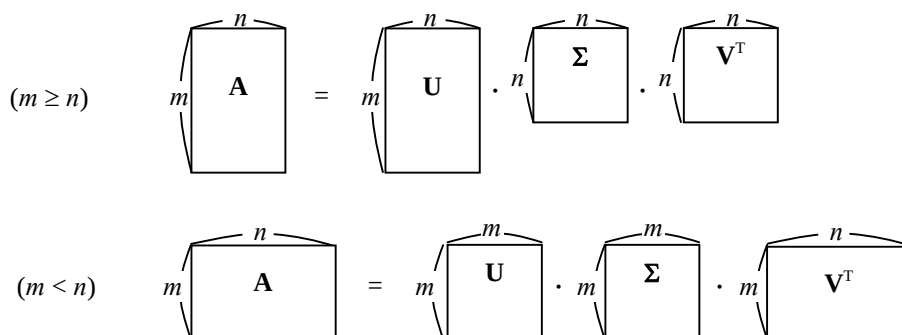


図 dasvd1-1. 特異値分解の各行列の寸法の関係

2. 引数

呼出し形式:

```
ierr = c_dasvd1((double*)a, ka, m, n, isw, sig, (double*)u, ku, (double*)v,
                kv, vw, &icon);
```

引数の説明:

a	double	入力	行列 $\mathbf{A}$.
	a[m][ka]		(“使用上の注意”d)参照)
ka	int	入力	配列 a の整合寸法 ($\geq n$).
m	int	入力	行列 $\mathbf{A}$ と $\mathbf{U}$ の行数 m .
n	int	入力	行列 $\mathbf{A}$, $\mathbf{U}$ の列数, $\mathbf{V}$ の行数 n .

isw	int	入力	制御情報. isw = $10d_1 + d_0$. ただし $0 \leq d_0, d_1 \leq 1$. d_1 0 のとき行列 U を求めない. 1 のとき行列 U を求める. d_0 0 のとき行列 V を求めない. 1 のとき行列 V を求める.
sig	double sig[Slen]	出力	行列 A の特異値. Slen = l+1. ("使用上の注意" c)参照)
u	double u[m][ku]	出力	行列 U . ("使用上の注意" b)参照)
ku	int	入力	配列 u の整合寸法($\geq n$).
v	double v[n][kv]	出力	行列 V . ("使用上の注意" b)参照)
kv	int	入力	配列 v の整合寸法($\geq \min(m+1, n)$).
vw	double vw[n+1]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
15000	ある特異値が求められなかった.	処理を打ち切る.
29000	実行時に獲得する作業用の領域が獲得できなかった.	処理を打ち切る.
30000	次のいずれかであった: $m < 1$ $n < 1$ $ka < n$ $ku < n$ $kv < \min(m+1, n)$ - isw $\neq 0, 1, 10$ 又は 11	処理を打ち切る.

3. 使用上の注意

a) 逆行列及び連立一次方程式の最小二乗ノルム解

特異値分解により求められる分解要素 **U**, **Σ** 及び **V** を用いるならば, 元の行列 **A** の一般逆行列 **A⁺** 又は, 連立 1 次方程式 **Ax = b** の最小二乗ノルム解が求められる.

しかし, このような場合, そのための専用関数 c_dginv 又は c_dlaxlm を利用の方が効果的である.

b) 行列 **U** 及び **V - u, v & isw**

特異値分解は, 応用の広い有効な方法であるが, 計算量が非常に多いことが欠点である. したがって **U**, **V** などは, 必要がなければ計算しないようにすることが望ましい. 入力引数 isw はそのためのものである. **U**, **V** を計算しないときは, これらを参照しないので, これらに対応する実引数は, 2 次元配列である必要はなく変数へのポインタでよい.

本関数では, 記憶領域の節約のために, **U** 又は **V** の中のどちらか一方を **a** の上に重ね書きができるように配慮されている. **a** を保存する必要がない場合に, **U** 又は **V** に対応する実引数として **a** を書くことにより目的を達成することができる.

c) sig

特異値は全て非負で、大きさの減少する順に格納される。ただし、icon = 15000 の場合には、非負のものが求められた特異値で、その他のものは-1となっており、特異値も大小順になっていない。

d) 行列 $A - a$

本関数は、行列 A の行数 m と列数 n の大小関係において何ら制約がない。つまり A が縦長行列、正方行列又は横長行列であっても、特異値分解を行うことができる。

4. 使用例

行列 A の特異値分解を行い、全ての特異値と対応する固有ベクトルを計算する。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define MMAX 7
#define NMAX 5

MAIN__()
{
    int ierr, icon;
    int m, n, i, j, ka, ku, kv, isw;
    double a[MMAX][NMAX], sig[NMAX], u[MMAX][NMAX], v[NMAX][NMAX], vw[NMAX];

    /* initialize system */
    m = MMAX;
    n = NMAX;
    for (i=0; i<n; i++)
        for (j=i; j<n; j++) {
            a[i][j] = n-j;
            a[j][i] = n-j;
        }
    for (i=n; i<m; i++)
        for (j=0; j<n; j++) {
            a[i][j] = 0;
            if (i%n == j) a[i][j] = 1;
        }
    ka = NMAX;
    ku = NMAX;
    kv = NMAX;
    isw = 11;
    /* singular value decomposition */
    ierr = c_dasvd1((double*)a, ka, m, n, isw, sig,
                   (double*)u, ku, (double*)v, kv, vw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dasvd1 failed with icon = %d\n", icon);
        exit(1);
    }
    /* print singular values and eigenvectors */
    for (i=0; i<n; i++) {
        printf("singular value: %10.4f\n", sig[i]);
        printf("e-vector:");
        for (j=0; j<n; j++)
            printf("%7.4f ", v[i][j]);
        printf("\n");
    }
    return(0);
}
```

5. 手法概要

ハウスホルダー法及び QR 法により特異値分解を行う。計算方法の詳細については“富士通 SSL II 使用手引書”のASVD1 及び[41]を参照のこと。

c_dbi0

第1種0次変形ベッセル関数 $I_0(x)$
<code>ierr = c_dbi0(x, &bi, &icon);</code>

1. 機能

第1種0次変形ベッセル関数 $I_0(x)$ の値,

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

を多項式及び漸近形式の近似式を用いて計算する.

2. 引数

呼出し形式:

```
ierr = c_dbi0(x, &bi, &icon);
```

引数の説明:

x	double	入力	独立変数 x .
bi	double	出力	関数値 $I_0(x)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	$ x > \log(fl_{\max})$ であった.	$bi = fl_{\max}$ とする.

3. 使用上の注意

a) 引数 x

引数 x の範囲は

$$|x| \leq \log(fl_{\max})$$

であること.

$|x|$ が上記範囲を超えると e^x の計算でオーバーフローを生じる. これを本関数内で前もって防ぐためである.

4. 使用例

$I_0(x)$ の値を x が 0 から 100 まで, 1 おきに計算し数表を作る.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
    double x, bi;
```

```
int i;

for (i=0;i<=100;i++) {
    x = (double)i;
    /* calculate Bessel function */
    ierr = c_dbi0(x, &bi, &icon);
    if (icon == 0)
        printf("x = %4.2f    bi = %e\n", x, bi);
    else
        printf("ERROR: x = %4.2f    bi = %e    icon = %i\n", x, bi, icon);
}
return(0);
}
```

5. 手法概要

変形ベッセル関数 $I_0(x)$ は, x の値によって近似式の形が異なる. $I_0(-x) = I_0(x)$ であるから, 以下では $|x|$ を x として説明する.

- $0 \leq x < 8$ のとき, べき級数展開の近似式を用いて計算する.
- $8 \leq x \leq \log(fl_{\max})$ のとき, 漸近展開の近似式を用いて計算する.

計算方法の詳細については“富士通 SSL II 使用手引書”の BI0 の項目を参照のこと.

c_dbi1

第 1 種 1 次変形ベッセル関数 $I_1(x)$
ierr = c_dbi1(x, &bi, &icon);

1. 機能

第 1 種 1 次変形ベッセル関数 $I_1(x)$ の値,

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

を多項式及び漸近形式の近似式を用いて計算する.

2. 引数

呼出し形式:

```
ierr = c_dbi1(x, &bi, &icon);
```

引数の説明:

x	double	入力	独立変数 x .
bi	double	出力	関数値 $I_1(x)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
20000	$ x > \log(fl_{\max})$ であった. $ x < -\log(fl_{\max})$ であった.	$bi = fl_{\max}$ とする. $bi = -fl_{\max}$ とする.

3. 使用上の注意

a) 引数 x

引数 x の範囲は

$$|x| \leq \log(fl_{\max})$$

であること.

$|x|$ が上記範囲を超えると e^x の計算でオーバーフローを生じる. これを本関数内で前もって防ぐためである.

4. 使用例

$I_1(x)$ の値を x が 0 から 100 まで 1 おきに計算して数表を作る.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
```

```
int ierr, icon;
double x, bi;
int i;

for (i=0;i<=100;i++) {
    x = (double)i;
    /* calculate Bessel function */
    ierr = c_dbi1(x, &bi, &icon);
    if (icon == 0)
        printf("x = %4.2f    bi = %e\n", x, bi);
    else
        printf("ERROR: x = %4.2f    bi = %e    icon = %i\n", x, bi, icon);
}
return(0);
}
```

5. 手法概要

変形ベッセル関数 $I_1(x)$ は, x の値によって近似式の形が異なる. $I_1(-x) = -I_1(x)$ であるから, 以下では, $|x|$ を x として説明する.

- $0 \leq x < 8$ のとき, べき級数展開の近似式を用いて計算する.
- $8 \leq x \leq \log(fl_{\max})$ のとき, 漸近展開の近似式を用いて計算する.

計算方法の詳細については“富士通 SSL II 使用手引書”の BI1 の項目を参照のこと.

c_dbic1

B-spline 補間式 (I)
ierr = c_dbic1(x, y, dy, n, m, c, vw, &icon);

1. 機能

離散点 $x_1, x_2, \dots, x_n (x_1 < x_2 < \dots < x_n)$ において関数値 $y_i = f(x_i), i=1, 2, \dots, n$ が与えられ, 更に両端 x_1, x_n において微係数, $y_1^{(\lambda)} = f^{(\lambda)}(x_1), y_n^{(\lambda)} = f^{(\lambda)}(x_n), \lambda=1, \dots, (m-1)/2$ が与えられたとき, m (奇数) 次の正規化された B-spline の線型結合による補間式,

$$S(x) = \sum_{j=1-m}^{n-1} c_j N_{j,m+1}(x) \quad (1)$$

の補間係数 $c_j, j=-m+1, -m+2, \dots, n-1$ を求める.

ここで求める $S(x)$ は,

$$\begin{cases} S^{(\lambda)}(x_1) = \sum_{j=1-m}^{n-1} c_j N_{j,m+1}^{(\lambda)}(x_1) = y_1^{(\lambda)}, \lambda=0, 1, \dots, (m-1)/2 \\ S(x_i) = \sum_{j=1-m}^{n-1} c_j N_{j,m+1}(x_i) = y_i, i=2, 3, \dots, n-1 \\ S^{(\lambda)}(x_n) = \sum_{j=1-m}^{n-1} c_j N_{j,m+1}^{(\lambda)}(x_n) = y_n^{(\lambda)}, \lambda=(m-1)/2, (m-1)/2-1, \dots, 0 \end{cases}$$

を満たすものである. $m \geq 3, n \geq 2$ であること.

2. 引数

呼出し形式:

```
ierr = c_dbic1(x, y, (double*)dy, n, m, c, vw, &icon);
```

引数の説明:

x	double x[n]	入力	離散点 x_i .
y	double y[n]	入力	関数値 y_i .
dy	double dy[(m-1)/2][2]	入力	両端 x_1, x_n における微係数. $dy[\lambda-1][0] = y_1^{(\lambda)}, dy[\lambda-1][1] = y_n^{(\lambda)}, \lambda=1, 2, \dots, (m-1)/2$.
n	int	入力	離散点の個数 n
m	int	入力	B-spline の次数 m . (“使用上の注意”b)参照)
c	double c[n+m-1]	出力	補間係数 c_j .
vw	double vw[Vwlen]	作業領域	$Vwlen = (n-2)m + (m+1)^2 / 2 + (m+1)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	次のいずれかであった. • m が奇数でない • $x[i] \geq x[i+1]$ なる $x[i]$ がある • $m < 3$ • $n < 2$	処理を打ち切る.

3. 使用上の注意

a) c_dbif1 との関係

本関数に続けて, 関数 c_dbif1 を呼び出すことにより, B-spline 補間式 (1) に基づく補間値あるいは微分値あるいは積分値を求めることができる. その時, 引数 x, n, m, c の値は c_dbif1 の入力となる.

b) m

次数 m は 3 又は 5 程度がよい. 元の関数がおとなしく, y_i が高精度で与えられているならば, もう少し次数を高くしてもよい. しかし, 15 程度が限界である.

4. 使用例

[0,1] 区間で等間隔な 10 個の補間点での関数 $f(x) = x^3$ の B-spline 補間式の補間係数を求め, ある点での補間値, 微分値及び積分値を求める.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define N 10
#define M 3

MAIN__()
{
    int ierr, icon;
    int i, n, m, isw;
    double x[N], y[N], c[N+M-1], dy[1][2], vw[36];
    double p, h, v, f;

    /* initialize data */
    n = N;
    m = M;
    p = 0;
    h = 1.0/(n-1);
    /* set function values */
    for (i=0;i<n;i++) {
        x[i] = p+i*h;
        y[i] = x[i]*x[i]*x[i];
    }
    /* set derivative values at end-points */
    dy[0][0] = 3*x[0]*x[0];
    dy[0][1] = 3*x[n-1]*x[n-1];

    /* calculate B-spline interpolation coefficients */
    ierr = c_dbic1(x, y, (double*)dy, n, m, c, vw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dbic1 failed with icon = %d\n", icon);
        exit(1);
    }
    i = 4;
    v = 0.5;
    for (isw=-1;isw<=m;isw++) {
        /* calculate value at point */
        ierr = c_dbif1(x, n, m, c, isw, v, &i, &f, vw, &icon);
        if (icon >= 20000) {
```

```
        printf("ERROR: c_dbif1 failed with icon = %d\n", icon);
        exit(1);
    }
    if (isw == -1)
        printf("icon = %i    integral = %12.6e\n", icon, f);
    else if (isw == 0)
        printf("icon = %i    value = %12.6e\n", icon, f);
    else
        printf("icon = %i    derivative %i = %12.6e\n", icon, isw, f);
    }
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の BIC1 の項目を参照のこと。

c_dbic2

B-spline 補間式 (II)
ierr = c_dbic2(x, y, dy, n, m, c, vw, &icon);

1. 機能

離散点 $x_1, x_2, \dots, x_n (x_1 < x_2 < \dots < x_n)$ において関数値 $y_i = f(x_i), i=1, 2, \dots, n$ が与えられ, 更に両端 x_1, x_n において微係数 $y_1^{(\lambda)} = f^{(\lambda)}(x_1), y_n^{(\lambda)} = f^{(\lambda)}(x_n), \lambda = (m+1)/2, (m+1)/2+1, \dots, m-1$ が与えられたとき, m (奇数) 次の正規化された B-spline の線型結合による補間式,

$$S(x) = \sum_{j=1-m}^{n-1} c_j N_{j,m+1}(x) \quad (1)$$

の補間係数 $c_j, j=-m+1, -m+2, \dots, n-1$ を求める.

ここで求める $S(x)$ は,

$$\begin{cases} S^{(\lambda)}(x_1) = \sum_{j=1-m}^{n-1} c_j N_{j,m+1}^{(\lambda)}(x_1) = y_1^{(\lambda)}, \lambda = (m+1)/2, (m+1)/2+1, \dots, m-1 \\ S(x_i) = \sum_{j=1-m}^{n-1} c_j N_{j,m+1}(x_i) = y_i, i=1, 2, \dots, n \\ S^{(\lambda)}(x_n) = \sum_{j=1-m}^{n-1} c_j N_{j,m+1}^{(\lambda)}(x_n) = y_n^{(\lambda)}, \lambda = m-1, m-2, \dots, (m+1)/2 \end{cases}$$

を満たすものである. $m \geq 3, n \geq (m+1)/2$ であること.

2. 引数

呼出し形式:

```
ierr = c_dbic2 (x, y, (double*)dy, n, m, c, vw, &icon);
```

引数の説明:

x	double x[n]	入力	離散点 x_i .
y	double y[n]	入力	関数値 y_i .
dy	double dy[(m-1)/2][2]	入力	両端 x_1, x_n における微係数. $dy[\lambda-(m+1)/2][0] = y_1^{(\lambda)}, dy[\lambda-(m+1)/2][1] = y_n^{(\lambda)},$ $\lambda = (m+1)/2, (m+1)/2+1, \dots, m-1.$
n	int	入力	離散点の個数 n .
m	int	入力	B-spline の次数 m . (“使用上の注意”b)参照)
c	double c[n+m-1]	出力	補間係数 c_j .
vw	double vw[Vwlen]	作業領域	$Vwlen = m(n+m-3) + 2(m+1).$
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
30000	次のいずれかであった. • m が奇数でない • $x[i] \geq x[i+1]$ なる $x[i]$ がある • $m < 3$ • $n < (m+1)/2$	処理を打ち切る.

3. 使用上の注意

a) c_dbif2 との関係

本関数に続けて, 関数 c_dbif2 を呼び出すことにより, B-spline 補間式 (1) に基づく補間値あるいは微分値あるいは積分値を求めることができる. その時, 引数 x, n, m, c の値は c_dbif2 の入力となる.

b) m

次数 m は 3 又は 5 程度がよい. 元の関数がおとなしく, y_i が高精度で与えられているならば, もう少し次数を高くしてもよい. しかし, 15 程度が限界である.

4. 使用例

[0,1] 区間で等間隔な 10 個の補間点での関数 $f(x) = x^3$ の B-spline 補間式の補間係数を求め, ある点での補間値, 微分値及び積分値を求める.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define N 10
#define M 3

MAIN__()
{
    int ierr, icon;
    int i, n, m, isw;
    double x[N], y[N], c[N+M-1], dy[1][2], vw[38];
    double p, h, v, f;

    /* initialize data */
    n = N;
    m = M;
    p = 0;
    h = 1.0/(n-1);
    /* set function values */
    for (i=0;i<n;i++) {
        x[i] = p+i*h;
        y[i] = x[i]*x[i]*x[i];
    }
    /* set derivative values at end-points */
    dy[0][0] = 3*x[0]*x[0];
    dy[0][1] = 3*x[n-1]*x[n-1];
    /* calculate B-spline interpolation coefficients */
    ierr = c_dbic2(x, y, (double*)dy, n, m, c, vw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dbic2 failed with icon = %d\n", icon);
        exit(1);
    }
    i = 4;
    v = 0.5;
    for (isw=-1;isw<=m;isw++) {
        /* calculate value at point */
        ierr = c_dbif2(x, n, m, c, isw, v, &i, &f, vw, &icon);
```

```
    if (icon >= 20000) {
        printf("ERROR: c_dbif2 failed with icon = %d\n", icon);
        exit(1);
    }
    if (isw == -1)
        printf("icon = %i    integral = %12.6e\n", icon, f);
    else if (isw == 0)
        printf("icon = %i    value = %12.6e\n", icon, f);
    else
        printf("icon = %i    derivative %i = %12.6e\n", icon, isw, f);
    }
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の BIC2 の項目を参照のこと.

c_dbic3

B-spline 補間式(III)
ierr = c_dbic3(x, y, n, m, c, xt, vw, &icon);

1. 機能

離散点 $x_1, x_2, \dots, x_n$ ($x_1 < x_2 < \dots < x_n$) に対して, 関数値 $y_i = f(x_i)$, $i = 1, 2, \dots, n$ が与えられたとき,

$$\begin{aligned}\xi_1 &= x_1 \\ \xi_i &= x_{i+(m-1)/2}, \quad i = 2, 3, \dots, n-m \\ \xi_{n-m+1} &= x_n\end{aligned}$$

を接点とする m (奇数) 次の正規化された B-spline の線型結合 による補間式,

$$S(x) = \sum_{j=1-m}^{n-m} c_j N_{j,m+1}(x) \quad (1)$$

の補間係数 c_j , $j = -m+1, -m+2, \dots, n-m$ を求める. $m \geq 3$, $n \geq m+2$ であること.

2. 引数

呼出し形式:

```
ierr = c_dbic3(x, y, n, m, c, xt, vw, &icon);
```

引数の説明:

x	double x[n]	入力	離散点 x_i .
y	double y[n]	入力	関数値 y_i .
n	int	入力	離散点の個数 n .
m	int	入力	B-spline の次数 m . (“使用上の注意”b) 参照)
c	double c[n]	出力	補間係数 c_j .
xt	double xt[n-m+1]	出力	節点列 ξ_i .
vw	double vw[m*n+2]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	次のいずれかであった. m が奇数でない. $n < m + 2$ $x[i] \geq x[i+1]$ なる $x[i]$ がある. $m < 3$	処理を打ち切る.

3. 使用上の注意

a) c_dbif3 との関係

本関数に続けて、関数 c_dbic3 を呼び出すことにより、B-spline 補間式(1)に基づく補間値あるいは微分値あるいは積分値を求めることができる。その時、引数 x, n, m, c 及び xt の値は関数 c_dbic3 の入力となる。

b) m

次数 m は 3 又は 5 程度がよい。元の関数がおとなしく、 y_i が高精度で与えられているならばもう少し次数を高くしてもよい。しかし 15 を超えると良くないといわれている。

4. 使用例

[0,1] 区間で等間隔な 10 個の補間点での関数 $f(x) = \sin(x)\sqrt{x}$ の値が与えられたとき、ある点での補間値を求める。

```
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define N 10
#define M 3

MAIN__()
{
    int ierr, icon;
    int i, n, m, isw;
    double x[N], y[N], c[N], xt[N-M+1], vw[M*N+2];
    double p, h, v, f;

    /* initialize data */
    n = N;
    m = M;
    isw = 0;
    p = 0;
    h = 1.0/n;
    for (i=0; i<n; i++) {
        x[i] = p;
        y[i] = sin(p)*sqrt(p);
        p = p + h;
    }
    /* calculate B-spline interpolation coefficients */
    ierr = c_dbic3(x, y, n, m, c, xt, vw, &icon);
    i = n/2;
    v = x[i] + (x[i+1]-x[i])/2;
    for (isw=-1; isw<=m; isw++) {
        /* calculate value at point */
        ierr = c_dbif3(x, n, m, c, xt, isw, v, &i, &f, vw, &icon);
        if (isw == -1)
            printf("icon = %i    integral = %12.6e\n", icon, f);
        else if (isw == 0)
            printf("icon = %i    value = %12.6e\n", icon, f);
        else
            printf("icon = %i    derivative %i = %12.6e\n", icon, isw, f);
    }
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の BIC3 の項目を参照のこと。

c_dbic4

B-spline 補間式 (IV)
ierr = c_dbic4 (x, y, n, m, c, vw, &icon);

1. 機能

離散点 $x_1, x_2, \dots, x_n$ ($x_1 < x_2 < \dots < x_n$) に対して, 周期 $(x_n - x_1)$ の周期関数値 $y_i = f(x_i)$, $i=1, 2, \dots, n$ (ただし $y_1=y_n$) が与えられたとき, m (奇数) 次の正規化された B-spline の線型結合による補間式,

$$S(x) = \sum_{j=1-m}^{n-1} c_j N_{j,m+1}(x) \quad (1)$$

の補間係数 $c_j, j=-m+1, -m+2, \dots, n-1$ を求める.

ここで求める $S(x)$ は $f(x)$ と同じく周期 $(x_n - x_1)$ の周期関数で, 境界条件 $S^{(\lambda)}(x_1) = S^{(\lambda)}(x_n), \lambda = 0, 1, \dots, m-1$ を満たすものである. $m \geq 3, n \geq m+2$ であること.

2. 引数

呼出し形式:

```
ierr = c_dbic4 (x, y, n, m, c, vw, &icon);
```

引数の説明:

x	double x[n]	入力	離散点 x_i .
y	double y[n]	入力	関数値 y_i . $y_1 = y_n$ であること. もし $y_1 \neq y_n$ であったときは y_n の方を採用する.
n	int	入力	離散点の個数 n .
m	int	入力	B-spline の次数 m . ("使用上の注意"b)参照)
c	double c[n+m-1]	出力	補間係数 c_j .
vw	double vw[Vwlen]	作業領域	$Vwlen = (n-1)(2m-1) + m + 1$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	次のいずれかであった. m が奇数でない $x[i] \geq x[i+1]$ なる $x[i]$ がある $m < 3$ 又は $n < m+2$	処理を打ち切る.

3. 使用上の注意

a) c_dbif4 との関係

本関数に続けて, 関数 c_dbif4 を呼び出すことにより, B-spline 補間式 (1) に基づく補間値あるいは微分値あるいは積分値を求めることができる. そのとき, 引数 x, n, m, c の値は c_dbif4 の入力となる.

b) m

次数 m は 3 又は 5 程度がよい. 元の関数がおとなしく, y_i が高精度で与えられているならば, もう少し次数を高くしてもよい. しかし 15 を超えると良くないといわれている.

4. 使用例

$[0, 2\pi]$ 区間で等間隔な 10 個の補間点での関数 $f(x) = \sin x$ の B-spline 補間式の補間係数を求め, ある点での補間値, 微分値及び積分値を求める.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define N 10
#define M 3

MAIN__()
{
    int ierr, icon;
    int i, n, m, isw;
    double x[N], y[N], c[N+M-1], vw[49];
    double p, h, v, f, pi;

    /* initialize data */
    n = N;
    m = M;
    p = 0;
    pi = 2*asin(1);
    h = 2*pi/(n-1);
    /* set function values */
    for (i=0; i<n; i++) {
        x[i] = p+i*h;
        y[i] = sin(x[i]);
    }
    /* calculate B-spline interpolation coefficients */
    ierr = c_dbic4(x, y, n, m, c, vw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dbic4 failed with icon = %d\n", icon);
        exit(1);
    }
    i = 4;
    v = pi;
    for (isw=-1; isw<=m; isw++) {
        /* calculate value at point */
        ierr = c_dbif4(x, n, m, c, isw, v, &i, &f, vw, &icon);
        if (icon >= 20000) {
            printf("ERROR: c_dbif4 failed with icon = %d\n", icon);
            exit(1);
        }
        if (isw == -1)
            printf("icon = %i    integral = %12.6e\n", icon, f);
        else if (isw == 0)
            printf("icon = %i    value = %12.6e\n", icon, f);
        else
            printf("icon = %i    derivative %i = %12.6e\n", icon, isw, f);
    }
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の BIC4 の項目を参照のこと.

c_dbicd1

B-spline 2 次元補間式 (I-I)

```
ierr = c_dbicd1(x, nx, y, ny, fxy, k, m, c,
                vw, &icon);
```

1. 機能

xy 平面上の格子点 (x_i, y_j) , $(x_1 < x_2 < \dots < x_{n_x}, y_1 < y_2 < \dots < y_{n_y})$ において関数値, $f_{ij} = f(x_i, y_j)$ が与えられ, 更に周囲において偏微係数 $f_{i,j}^{(\lambda,\mu)}$, $i=1, n_x, j=1, n_y, \lambda=1,2,\dots, (m-1)/2, \mu=1,2,\dots, (m-1)/2$ が与えられたとき, 双 m (奇数) 次の B-spline 2 次元補間式,

$$S(x, y) = \sum_{\beta=1-m}^{n_y-1} \sum_{\alpha=1-m}^{n_x-1} c_{\alpha,\beta} N_{\alpha,m+1}(x) N_{\beta,m+1}(y) \quad (1)$$

の補間係数 $c_{\alpha,\beta}$ を求める. $m \geq 3, n_x \geq 2, n_y \geq 2$ であること.

2. 引数

呼出し形式:

```
ierr = c_dbicd1(x, nx, y, ny, (double*)fxy, k, m, (double*)c, vw, &icon)
```

引数の説明:

x	double x[nx]	入力	x 方向の離散点 x_i .
nx	int	入力	x_i の個数 n_x .
y	double y[ny]	入力	y 方向の離散点 y_j .
ny	int	入力	y_j の個数 n_y .
fxy	double fxy[Fxylen][k]	入力	関数値などの $\hat{f}_{i,j}$. $Fxylen = n_x + m - 1$. (“使用上の注意”a)参照)
k	int	入力	2 次元配列 fxy 及び c の整合寸法 ($\geq ny + m - 1$).
m	int	入力	B-spline の次数 m . (“使用上の注意”c)参照)
c	double c[Clen][k]	出力	補間係数 $c_{\alpha,\beta}$. $Clen = n_x + m - 1$.
vw	double vw[Vwlen]	作業領域	$Vwlen = (\max(n_x, n_y) + 1)(m + 2) - 3 + (m + 1)^2 / 2$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	次のいずれかであった. • m が奇数でない • $x[i] \geq x[i+1]$ なる $x[i]$ がある • $y[j] \geq y[j+1]$ なる $y[j]$ がある • $m < 3$ • $n_x < 2$ 又は $n_y < 2$	処理を打ち切る.

3. 使用上の注意

a) fxy

配列 fxy に関数値などの $\hat{f}_{i,j}$ を入力する. $\hat{f}_{i,j}$ は次のように定義する(但し $\ell = (m-1)/2$).

$$\begin{aligned}
 \hat{f}_{i,j} &= f_{1,1}^{(\ell+1-i,\ell+1-j)} & i=1,2,\dots,\ell+1; \quad j=1,2,\dots,\ell+1; \\
 \hat{f}_{i,j} &= f_{i-\ell,1}^{(0,\ell+1-j)} & i=\ell+2,\ell+3,\dots,\ell+n_x-1; \quad j=1,2,\dots,\ell+1; \\
 \hat{f}_{i,j} &= f_{n_x,1}^{(i-\ell-n_x,\ell+1-j)} & i=\ell+n_x,\ell+n_x+1,\dots,2\ell+n_x; \quad j=1,2,\dots,\ell+1; \\
 \hat{f}_{i,j} &= f_{1,j-\ell}^{(\ell+1-i,0)} & i=1,2,\dots,\ell+1; \quad j=\ell+2,\ell+3,\dots,\ell+n_y-1; \\
 \hat{f}_{i,j} &= f_{i-\ell,j-\ell} & i=\ell+2,\ell+3,\dots,\ell+n_x-1; \quad j=\ell+2,\ell+3,\dots,\ell+n_y-1; \\
 \hat{f}_{i,j} &= f_{n_x,j-\ell}^{(i-\ell-n_x,0)} & i=\ell+n_x,\ell+n_x+1,\dots,2\ell+n_x; \quad j=\ell+2,\ell+3,\dots,\ell+n_y-1; \\
 \hat{f}_{i,j} &= f_{1,n_y}^{(\ell+1-i,j-\ell-n_y)} & i=1,2,\dots,\ell+1; \quad j=\ell+n_y,\ell+n_y+1,\dots,2\ell+n_y; \\
 \hat{f}_{i,j} &= f_{i-\ell,n_y}^{(0,j-\ell-n_y)} & i=\ell+2,\ell+3,\dots,\ell+n_x-1; \quad j=\ell+n_y,\ell+n_y+1,\dots,2\ell+n_y; \\
 \hat{f}_{i,j} &= f_{n_x,n_y}^{(i-\ell-n_x,j-\ell-n_y)} & i=\ell+n_x,\ell+n_x+1,\dots,2\ell+n_x; \quad j=\ell+n_y,\ell+n_y+1,\dots,2\ell+n_y.
 \end{aligned}$$

$\hat{f}_{i,j}$ を行列の要素として並べたものを以下に示す.

	$j = 1$	$j = \ell + 1$	$j = \ell + 2$	$j = \ell + n_y - 1$	$j = \ell + n_y$	$j = 2\ell + n_y$
$i = 1$	(x_1, y_1) における 関数値, 偏微係数	$(x_1, y_{j-\ell})$ における 関数値, 偏微係数		(x_1, y_{n_y}) における 関数値, 偏微係数		
$i = \ell + 1$ $i = \ell + 2$		$(x_{i-\ell}, y_{j-\ell})$ における関数値 $2 \leq i - \ell \leq n_x - 1,$ $2 \leq j - \ell \leq n_y - 1$		$(x_{i-\ell}, y_{n_y})$ における 関数値, 偏微係数		
$i = \ell + n_x - 1$ $i = \ell + n_x$		$(x_{n_x}, y_{j-\ell})$ における 関数値, 偏微係数		(x_{n_x}, y_{n_y}) における 関数値, 偏微係数		
$i = 2\ell + n_x$						

b) c_dbifd1 との関係

本関数に続けて, 関数 c_dbifd1 を呼び出すことにより, B-spline 2次元補間式 (1) に基づく補間値あるいは偏微分値あるいは二重積分値を求めることができる. その時, 引数 x, nx, y, ny, k, m, c の値は c_dbifd1 の入力となる.

c) m

次数 m は 3 又は 5 程度がよい. 元の関数がおとなしく, $f_{i,j}^{(\lambda,\mu)}$ が高精度で与えられているならば, もう少し次数を高くしてもよい. しかし 15 が限界である.

4. 使用例

区間 $[0,1] \times [0,1]$ で 100 個の補間点での関数 $f(x, y) = x^3 y^3$ の B-spline 補間式の補間係数を求め, ある点での補間値, 微分値及び積分値を求める.

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

```

```
#define N 10
#define M 3

/* function prototype for initializer */
void gen(double x[], double y[], int n, double fxy[][N+M-1]);

MAIN__()
{
    int ierr, icon;
    int i, j, nx, ny, m, k, ix, iy, iswx, iswy;
    double x[N], y[N], fxy[N+M-1][N+M-1], c[N+M-1][N+M-1];
    double vw[60];
    double hx, hy, px, py, vx, vy, f;

    /* initialize data */
    nx = N;
    ny = N;
    m = M;
    k = N+M-1;
    hx = 1.0/(nx-1);
    hy = 1.0/(ny-1);
    px = 0;
    for (i=0;i<nx;i++) {
        x[i] = px+i*hx;
    }
    py = 0;
    for (j=0;j<ny;j++) {
        y[j] = py+j*hy;
    }
    /* generate function and derivative values in fxy */
    gen(x, y, nx, fxy);

    /* calculate B-spline interpolation coefficients */
    ierr = c_dbicd1(x, nx, y, ny, (double*)fxy, k,
                   m, (double*)c, vw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dbicd1 failed with icon = %d\n", icon);
        exit(1);
    }
    ix = 4;
    vx = 0.5;
    iy = 4;
    vy = 0.5;
    for (iswx=-1;iswx<=m;iswx++) {
        iswy = iswx;
        /* calculate value at point */
        ierr = c_dbifd1(x, nx, y, ny, m, (double*)c, k,
                       iswx, vx, &ix, iswy, vy, &iy, &f, vw, &icon);
        if (icon >= 20000) {
            printf("ERROR: c_dbifd1 failed with icon = %d\n", icon);
            exit(1);
        }
        if (iswx == -1)
            printf("icon = %i    integral = %12.6e\n", icon, f);
        else if (iswx == 0)
            printf("icon = %i    value = %12.6e\n", icon, f);
        else
            printf("icon = %i    derivative %i = %12.6e\n", icon, iswx, f);
    }
    return(0);
}

/* generate function and derivative values for f=x^3y^3 */
void gen(double x[], double y[], int n, double fxy[][N+M-1])
{
    double y1, yn, x1, xn, fx, fy;
    int i, j;

    /* corner points; df/dxdy values */
    fxy[0][0] = 9*x[0]*x[0]*y[0]*y[0];
    fxy[n+1][0] = 9*x[n-1]*x[n-1]*y[0]*y[0];
    fxy[0][n+1] = 9*x[0]*x[0]*y[n-1]*y[n-1];
    fxy[n+1][n+1] = 9*x[n-1]*x[n-1]*y[n-1]*y[n-1];

    /* partial derivatives on edges: df/dx, df/dy */
    y1 = y[0]*y[0]*3;
```

```
yn = y[n-1]*y[n-1]*3;
x1 = x[0]*x[0]*3;
xn = x[n-1]*x[n-1]*3;

/* edges; fx.df/dy or fy.df/dx */
for (i=0;i<n;i++) {
    fx = x[i]*x[i]*x[i];
    fy = y[i]*y[i]*y[i];
    fxy[i+1][0] = y1*fx;
    fxy[0][i+1] = x1*fy;
    fxy[n+1][i+1] = xn*fy;
    fxy[i+1][n+1] = yn*fx;
}

/* central area; function values */
for (i=0;i<n;i++) {
    fx = x[i]*x[i]*x[i];
    for (j=0;j<n;j++) {
        fxy[i+1][j+1] = fx*y[j]*y[j]*y[j];
    }
}
return;
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の BICD1 の項目を参照のこと.

c_dbicd3

B-spline 2 次元補間式 (III-III)
ierr = c_dbicd3(x, nx, y, ny, fxy, k, m, c, xt, vw, &icon);

1. 機能

xy 平面上の格子点 (x_i, y_j) , $(x_1 < x_2 < \dots < x_{n_x}, y_1 < y_2 < \dots < y_{n_y})$ において関数値, $f_{ij} = f(x_i, y_j)$ が与えられたとき, 双 m (奇数)次の B-spline 2 次元補間式,

$$S(x, y) = \sum_{\beta=1-m}^{n_y-m} \sum_{\alpha=1-m}^{n_x-m} c_{\alpha,\beta} N_{\alpha,m+1}(x) N_{\beta,m+1}(y) \quad (1)$$

の補間係数 $c_{\alpha,\beta}$ を求める.

ここで $S(x, y)$ の節点は, x 方向には(2), y 方向には(3)のように採る.

$$\xi_i = \begin{cases} x_1 & i = 1 \\ x_{i+(m-1)/2} & i = 2, 3, \dots, n_x - m \\ x_{n_x} & i = n_x - m + 1 \end{cases} \quad (2)$$

$$\eta_j = \begin{cases} y_1 & j = 1 \\ y_{j+(m-1)/2} & j = 2, 3, \dots, n_y - m \\ y_{n_y} & j = n_y - m + 1 \end{cases} \quad (3)$$

$m \geq 3, n_x \geq m+2, n_y \geq m+2$ であること.

2. 引数

呼出し形式:

```
ierr = c_dbicd3(x, nx, y, ny, (double*)fxy, k, m, (double*)c, xt, vw,
                &icon);
```

引数の説明:

x	double x[nx]	入力	x 方向の離散点 x_i .
nx	int	入力	x_i の個数 n_x .
y	double y[ny]	入力	y 方向の離散点 y_j .
ny	int	入力	y_j の個数 n_y .
fxy	double fxy[nx][k]	入力	関数値 f_{ij} . fxy[i-1][j-1]に f_{ij} を入力する.
k	int	入力	fxy 及び c の整合寸法 ($\geq ny$).
m	int	入力	B-spline の次数 m . (“使用上の注意”b) 参照)
c	double c[nx][k]	出力	補間係数 $c_{\alpha,\beta}$. c[i-1][j-1]に $c_{-m+i, -m+j}$ を出力する.
xt	double xt[Xilen]	出力	接点列 $\{\xi_i\}$ 及び $\{\eta_j\}$. $\{\xi_i\}, \{\eta_j\}$ の順に出力する. $Xilen = (nx - m + 1) + (ny - m + 1)$.
vw	double	作業領域	$Vwlen = (\max(nx, ny) - 2) * m + 2 * (m + 1) + 2 * \max(nx,$

vw[Vwlen] ny)
 icon int 出力 コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
29000	作業用領域の獲得に失敗した.	処理を打ち切る.
30000	次のいずれかであった. • m が奇数でない. • $nx < m + 2$ 又は $ny < m + 2$ • $x[i] \geq x[i+1]$ なる $x[i]$ がある. • $y[j] \geq y[j+1]$ なる $y[j]$ がある. • $m < 3$	処理を打ち切る.

3. 使用上の注意

a) c_dbicd3 との関係

本関数に続けて, 関数 c_dbicd3 を呼び出すことにより, B-spline 2 次元補間式(1)に基づく補間値あるいは偏微分値あるいは二重積分値を求めることができる. そのとき, 引数 x, nx, y, ny, m, c 及び xt の値は関数 c_dbicd3 の入力となる.

b) m

次数 m は 3 又は 5 程度がよい. 元の関数がおとなしく, f_{ij} が高精度で与えられているならばもう少し次数を高くしてもよい. しかし 15 を超えると良くないといわれている.

4. 使用例

[0,1]×[0,1] 区間で等間隔な 100 個の補間点での関数 $f(x, y) = \sin(xy)\sqrt{xy}$ の値が与えられたとき, ある点での補間値を求める.

```
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define N 10
#define M 3

MAIN__()
{
  int ierr, icon;
  int i, j, nx, ny, m, k, ix, iy, iswx, iswy;
  double x[N], y[N], fxy[N][N], c[N][N], xt[2*(N-M+1)];
  double vw[(N-2)*M+2*(M+1)+2*N];
  double hx, hy, px, py, vx, vy, f;

  /* initialize data */
  nx = N;
  ny = N;
  m = M;
  hx = 1.0/(nx-1);
  hy = 1.0/(ny-1);
  px = 0;
  for (i=0;i<nx;i++) {
    x[i] = px;
    px = px + hx;
  }
  py = 0;
  for (j=0;j<ny;j++) {
```

```
        y[j] = py;
        py = py + hy;
    }
    for (i=0;i<nx;i++)
        for (j=0;j<ny;j++) {
            px = x[i];
            py = y[j];
            fxy[i][j] = sin(px*py)*sqrt(px*py);
        }
    k = N;
    /* calculate B-spline interpolation coefficients */
    ierr = c_dbicd3(x, nx, y, ny, (double*)fxy, k,
                   m, (double*)c, xt, vw, &icon);

    ix = nx/2;
    vx = x[ix] + (x[ix+1]-x[ix])/2;
    iy = ny/2;
    vy = y[iy] + (y[iy+1]-y[iy])/2;
    for (iswx=-1;iswx<m;iswx++) {
        iswy = iswx;
        /* calculate value at point */
        ierr = c_dbifd3(x, nx, y, ny, m, (double*)c, k, xt,
                       iswx, vx, &ix, iswy, vy, &iy, &f, vw, &icon);
        if (iswx == -1)
            printf("icon = %i    integral = %12.6e\n", icon, f);
        else if (iswx == 0)
            printf("icon = %i    value = %12.6e\n", icon, f);
        else
            printf("icon = %i    derivative %i = %12.6e\n", icon, iswx, f);
    }
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の BICD3 の項目を参照のこと.

c_dbif1

B-spline 補間式 (I) による補間, 数値微分, 数値積分

ierr = c_dbif1(x, n, m, c, isw, v, &i, &f, vw, &icon);

1. 機能

離散点 $x_1, x_2, \dots, x_n (x_1 < x_2 < \dots < x_n)$ において関数値 $y_i = f(x_i)$, $i=1, 2, \dots, n$ が与えられ, 更に両端 x_1, x_n において微係数, $y_1^{(\lambda)} = f^{(\lambda)}(x_1)$, $y_n^{(\lambda)} = f^{(\lambda)}(x_n)$, $\lambda = 1, \dots, (m-1)/2$ が与えられたとき $x=v \in [x_1, x_n]$ における補間値あるいは微分値あるいは x_1 から v までの積分値を求める.

ただし, 本関数を利用する前に, 関数 c_dbic1 により B-spline 補間式,

$$S(x) = \sum_{j=1-m}^{n-1} c_j N_{j,m+1}(x) \quad (1)$$

における補間係数 $c_j, j=-m+1, -m+2, \dots, n-1$ が計算されているとする. ここで m は奇数で B-spline $N_{j,m+1}(x)$ の次数である. $x_1 \leq v \leq x_n, m \geq 3, n \geq 2$ であること.

2. 引数

呼出し形式:

```
ierr = c_dbif1(x, n, m, c, isw, v, &i, &f, vw, &icon);
```

引数の説明:

x	double x[n]	入力	離散点 x_i .
n	int	入力	離散点の個数 n .
m	int	入力	B-spline の次数 m .
c	double c[n+m-1]	入力	補間係数 c_j (c_dbic1 の出力).
isw	int	入力	計算の種類を与える. 0 補間値 $F = S(v)$ を求める. λ λ 階微分値 $F = S^{(\lambda)}(v)$ を求める. ($1 \leq \lambda \leq m$) -1 積分値 $F = \int_{x_1}^v S(x)dx$ を求める.
v	double	入力	補間値などを求めたい点 v .
i	int	入力	$x[i] \leq v < x[i+1]$ を満たす i . $v = x_n$ のときは $i = n-2$ とする.
f	double	出力	$x[i] \leq v < x[i+1]$ を満たす i . (“使用上の注意”b)参照)
		出力	補間値あるいは λ 階微分値あるいは積分値. (引数“isw”の項参照)
vw	double vw[m+1]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	$x[i] \leq v < x[i+1]$ が満たされていない.	左記の i をさがして処理を続ける.
30000	次のいずれかであった. $v < x[0]$ 又は $v > x[n-1]$ $isw < -1$ 又は $isw > m$	処理を打ち切る.

3. 使用上の注意

a) c_dbic1 との関係

本関数は, 関数 c_dbic1 により求められた B-spline 補間式 (1) に基づいて補間値あるいは微分値あるいは積分値を求めるものである. したがって, 本関数を呼び出す前に, c_dbic1 を呼び出して補間式 (1) を求め, 続けて本関数を呼び出すことにより補間値を求めることができる. そのとき, 引数 x , n , m , c は c_dbic1 のそれらと同一でなければならない.

b) i

引数 i は $x[i] \leq v < x[i+1]$ を満足していることが好ましい. もし満足されていないときは $x[i] \leq v < x[i+1]$ を満足するような i を探索して処理を続ける.

ここで, 数学で一般に使用する添え字と C 言語の配列の添え字では, 値が 1 ずれることに注意する必要がある. すなわち, C 言語の添え字は 0 から始まり, 数学の添え字は 1 から始まるため i は $i-1$ を設定する.

4. 使用例

区間[0,1]で等間隔な 10 個の補間点での関数 $f(x) = x^3$ の B-spline 補間式の補間係数を求め, ある点での補間値, 微分値及び積分値を求める.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define N 10
#define M 3

MAIN__()
{
    int ierr, icon;
    int i, n, m, isw;
    double x[N], y[N], c[N+M-1], dy[1][2], vw[36];
    double p, h, v, f;

    /* initialize data */
    n = N;
    m = M;
    p = 0;
    h = 1.0/(n-1);
    /* set function values */
    for (i=0;i<n;i++) {
        x[i] = p+i*h;
        y[i] = x[i]*x[i]*x[i];
    }
    /* set derivative values at end-points */
    dy[0][0] = 3*x[0]*x[0];
    dy[0][1] = 3*x[n-1]*x[n-1];

    /* calculate B-spline interpolation coefficients */
    ierr = c_dbic1(x, y, (double*)dy, n, m, c, vw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dbic1 failed with icon = %d\n", icon);
        exit(1);
    }
    i = 4;
```



```
v = 0.5;
for (isw=-1;isw<=m;isw++) {
    /* calculate value at point */
    ierr = c_dbif1(x, n, m, c, isw, v, &i, &f, vw, &icon);
    if (icon >= 20000) {
        printf("ERROR: c_dbif1 failed with icon = %d\n", icon);
        exit(1);
    }
    if (isw == -1)
        printf("icon = %i    integral = %12.6e\n", icon, f);
    else if (isw == 0)
        printf("icon = %i    value = %12.6e\n", icon, f);
    else
        printf("icon = %i    derivative %i = %12.6e\n", icon, isw, f);
}
return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の BIF1 の項目を参照のこと。

c_dbif2

B-spline 補間式 (II) による補間, 数値微分, 数値積分

ierr = c_dbif2 (x, n, m, c, isw, v, &i, &f, vw, &icon);

1. 機能

離散点 $x_1, x_2, \dots, x_n (x_1 < x_2 < \dots < x_n)$ において関数値 $y_i = f(x_i)$, $i=1, 2, \dots, n$ が与えられ, 更に両端 x_1, x_n において微係数 $y_1^{(\lambda)} = f^{(\lambda)}(x_1)$, $y_n^{(\lambda)} = f^{(\lambda)}(x_n)$, $\lambda = (m+1)/2, (m+1)/2+1, \dots, m-1$ が与えられたとき $x=v \in [x_1, x_n]$ における補間値あるいは微分値あるいは x_1 から v までの積分値を求める.

ただし, 本関数を利用する前に, c_dbic2 により B-spline 補間式,

$$S(x) = \sum_{j=1-m}^{n-1} c_j N_{j,m+1}(x) \quad (1)$$

における補間係数 $c_j, j=-m+1, -m+2, \dots, n-1$ が計算されているとする. ここで m は奇数で B-spline $N_{j,m+1}(x)$ の次数である. $x_1 \leq v \leq x_n, m \geq 3, n \geq (m+1)/2$ であること.

2. 引数

呼出し形式:

```
ierr = c_dbif2 (x, n, m, c, isw, v, &i, &f, vw, &icon);
```

引数の説明:

x	double x[n]	入力	離散点 x_i .
n	int	入力	離散点の個数 n .
m	int	入力	B-spline の次数 m .
c	double c[n+m-1]	入力	補間係数 c_i (c_dbic2 の出力).
isw	int	入力	計算の種類を与える. 0 補間値 $F = S(v)$ を求める. λ λ 階微分値 $F = S^{(\lambda)}(v)$ を求める. ($1 \leq \lambda \leq m$) -1 積分値 $F = \int_{x_1}^v S(x)dx$ を求める.
v	double	入力	補間値などを求めたい点 v .
i	int	入力	$x[i] \leq v < x[i+1]$ を満たす i . $v = x_n$ のときは $i = n-2$ とする.
f	double	出力	$x[i] \leq v < x[i+1]$ を満たす i . (“使用上の注意”b)参照)
vw	double vw[m+1]	作業領域	補間値あるいは λ 階微分値あるいは積分値. (“isw”の項参照)
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	$x[i] \leq v < x[i+1]$ が満たされていない.	左記の i をさがして処理を続ける.
30000	次のいずれかであった. $v < x[0]$ 又は $v > x[n-1]$ $isw < -1$ 又は $isw > m$	処理を打ち切る.

3. 使用上の注意

a) c_dbic2 との関係

本関数は, 関数 c_dbic2 により求められた B-spline 補間式 (1) に基づいて補間値あるいは微分値あるいは積分値を求めるものである. したがって, 本関数を呼び出す前に, c_dbic2 を呼び出して補間式 (1) を求め, 続けて本関数を呼び出すことにより補間値などを求めることができる. その時, 引数 x , n , m , c は c_dbic2 のそれらと同一でなければならない.

b) i

引数 i は $x[i] \leq v < x[i+1]$ を満足していることが好ましい. もし満足されていないときは $x[i] \leq v < x[i+1]$ を満足するような i を探索して処理を続ける.

ここで, 数学で一般に使用する添え字と C 言語の配列の添え字では, 値が 1 ずれることに注意する必要がある. すなわち, C 言語の添え字は 0 から始まり, 数学の添え字は 1 から始まるため i は $i-1$ を設定する.

4. 使用例

区間[0,1]で等間隔な 10 個の補間点での関数 $f(x) = x^3$ の B-spline 補間式の補間係数を求め, ある点での補間値, 微分値及び積分値を求める.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define N 10
#define M 3

MAIN__()
{
    int ierr, icon;
    int i, n, m, isw;
    double x[N], y[N], c[N+M-1], dy[1][2], vw[38];
    double p, h, v, f;

    /* initialize data */
    n = N;
    m = M;
    p = 0;
    h = 1.0/(n-1);
    /* set function values */
    for (i=0;i<n;i++) {
        x[i] = p+i*h;
        y[i] = x[i]*x[i]*x[i];
    }
    /* set derivative values at end-points */
    dy[0][0] = 3*x[0]*x[0];
    dy[0][1] = 3*x[n-1]*x[n-1];
    /* calculate B-spline interpolation coefficients */
    ierr = c_dbic2(x, y, (double*)dy, n, m, c, vw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dbic2 failed with icon = %d\n", icon);
        exit(1);
    }
    i = 4;
    v = 0.5;
```

```
for (isw=-1;isw<=m;isw++) {
    /* calculate value at point */
    ierr = c_dbif2(x, n, m, c, isw, v, &i, &f, vw, &icon);
    if (icon >= 20000) {
        printf("ERROR: c_dbif2 failed with icon = %d\n", icon);
        exit(1);
    }
    if (isw == -1)
        printf("icon = %i    integral = %12.6e\n", icon, f);
    else if (isw == 0)
        printf("icon = %i    value = %12.6e\n", icon, f);
    else
        printf("icon = %i    derivative %i = %12.6e\n", icon, isw, f);
}
return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の BIF2 の項目を参照のこと。

c_dbif3

B-spline 補間式 (III)による補間, 数値微分, 数値積分

ierr = c_dbif3(x, n, m, c, xt, isw, v, &i, &f, vw, &icon);

1. 機能

離散点 $x_1, x_2, \dots, x_n$ ($x_1 < x_2 < \dots < x_n$) に対して関数値 $y_i = f(x_i)$, $i = 1, 2, \dots, n$ が与えられたとき, $x = v$ における補間値あるいは微分値あるいは x_1 から v までの積分値を求める.

ただし, 関数を利用する前に, c_dbic3 により, 節点列 ξ_i , $i = 1, 2, \dots, n-m+1$ 及び B-spline 補間式,

$$S(x) = \sum_{j=1-m}^{n-m} c_j N_{j,m+1}(x) \quad (1)$$

における補間係数 c_j , $j = -m+1, -m+2, \dots, n-m$ が計算されているとする. ここで m は奇数で, B-spline $N_{j,m+1}(x)$ の次数である. $m \geq 3$, $x_1 \leq v \leq x_n$ かつ $n \geq m+2$ であること.

2. 引数

呼出し形式:

```
ierr = c_dbif3(x, n, m, c, xt, isw, v, &i, &f, vw, &icon);
```

引数の説明:

x	double x[n]	入力	離散点 x_i .
n	int	入力	離散点の個数 n .
m	int	入力	B-spline の次数 m .
c	double c[n]	入力	補間係数 c_j (c_dbic3 の出力).
xt	double xt[n-m+1]	入力	節点列 ξ_i (c_dbic3 の出力).
isw	int	入力	計算の種類を与える. 0 補間値 $F = S(v)$ を求める. l l 階微分値 $F = S^{(l)}(v)$ を求める. $1 \leq l \leq m$. -1 積分値 $F = \int_{x_1}^v S(x)dx$ を求める.
v	double	入力	補間値などを求めたい点 v .
i	int	入力	$x_i \leq v < x_{i+1}$ のとき i には $i-1$ を入力する. i は $x[i] \leq v < x[i+1]$ を満たす. $v = x_n$ のときは $i = n-2$ とすること.
		出力	$x_i \leq v < x_{i+1}$ のとき $i-1$ を出力する. (“使用上の注意”b)参照)
f	double	出力	補間値あるいは l 階微分値あるいは積分値 (引数“isw”の項参照).
vw	double [2*m+2]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	$x[i] \leq v < x[i+1]$ が満たされていない.	本関数内で左記の i をさがして処理を続ける.
30000	次のいずれかであった. $v < x[0]$ 又は $v > x[n-1]$ $isw < -1$ 又は $isw > m$	処理を打ち切る.

3. 使用上の注意

a) c_dbic3 との関係

本関数は、関数 c_dbic3 により、求められた B-spline 補間式(1)に基づいて、補間値あるいは微分値あるいは積分値を求めるものである。

したがって、本関数を呼び出す前に、関数 c_dbic3 を呼び出して補間式(1)を求め、続けて本関数を呼び出すことにより補間値などを求めることができる。そのとき、引数 x , n , m , c 及び xt は c_dbic3 のそれらと同一でなければならない。

b) i

引数 i は $x[i] \leq v < x[i+1]$ を満足していることが好ましい。もし満足していないときは、このような条件を満足するような i を探索して処理を続ける。

ここで、数学で一般に使用する添え字と C 言語の配列の添え字では、値が 1 ずれることに注意する必要がある。すなわち、C 言語の添え字は 0 から始まり、数学の添え字は 1 から始まるため i は $i-1$ を設定する。

4. 使用例

[0,1] 区間で等間隔な 10 個の補間点での関数 $f(x) = \sin(x)\sqrt{x}$ の値が与えられたとき、ある点での補間値を求める。

```
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define N 10
#define M 3

MAIN__()
{
    int ierr, icon;
    int i, n, m, isw;
    double x[N], y[N], c[N], xt[N-M+1], vw[M*N+2];
    double p, h, v, f;

    /* initialize data */
    n = N;
    m = M;
    isw = 0;
    p = 0;
    h = 1.0/n;
    for (i=0;i<n;i++) {
        x[i] = p;
        y[i] = sin(p)*sqrt(p);
        p = p + h;
    }
    /* calculate B-spline interpolation coefficients */
```

```
ierr = c_dbic3(x, y, n, m, c, xt, vw, &icon);
i = n/2;
v = x[i] + (x[i+1]-x[i])/2;
for (isw=-1;isw<=m;isw++) {
    /* calculate value at point */
    ierr = c_dbif3(x, n, m, c, xt, isw, v, &i, &f, vw, &icon);
    if (isw == -1)
        printf("icon = %i    integral = %12.6e\n", icon, f);
    else if (isw == 0)
        printf("icon = %i    value = %12.6e\n", icon, f);
    else
        printf("icon = %i    derivative %i = %12.6e\n", icon, isw, f);
}
return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の BIF3 の項目を参照のこと。

c_dbif4

B-spline 補間式 (IV) による補間, 数値微分, 数値積分

ierr = c_dbif4(x, n, m, c, isw, v, &i, &f, vw, &icon);

1. 機能

離散点 $x_1, x_2, \dots, x_n (x_1 < x_2 < \dots < x_n)$ に対して, 周期 $(x_n - x_1)$ の周期関数値 $y_i = f(x_i), i=1, 2, \dots, n$ (ただし $y_1 = y_n$) が与えられたとき, $x=v \in [x_1, x_n]$ における補間値あるいは微分値あるいは x_1 から v までの積分値を求める.

ただし, 本関数を利用する前に, 関数 c_dbic4 により, 周期条件を満たす B-spline 補間式,

$$S(x) = \sum_{j=1-m}^{n-1} c_j N_{j,m+1}(x) \quad (1)$$

における補間係数 $c_j, j = -m+1, -m+2, \dots, n-1$ が計算されているとする. ここで m は奇数で B-spline $N_{j,m+1}(x)$ の次数である. $x_1 \leq v \leq x_n, m \geq 3, n \geq m+2$ であること.

2. 引数

呼出し形式:

```
ierr = c_dbif4 (x, n, m, c, isw, v, &i, &f, vw, &icon);
```

引数の説明:

x	double x[n]	入力	離散点 x_i .
n	int	入力	離散点の個数 n .
m	int	入力	B-spline の次数 m .
c	double c[n+m-1]	入力	補間係数 c_j (c_dbic4 の出力).
isw	int	入力	計算の種類を与える. 0 補間値 $F = S(v)$ を求める. λ λ 階微分値 $F = S^{(\lambda)}(v)$ を求める. ($1 \leq \lambda \leq m$) -1 積分値 $F = \int_{x_1}^v S(x)dx$ を求める.
v	double	入力	補間値などを求めたい点 v .
i	int	入力	$x[i] \leq v < x[i+1]$ を満たす i . $v = x_n$ のときは $i = n-2$ とする.
f	double	出力	$x[i] \leq v < x[i+1]$ を満たす i . (“使用上の注意”b)参照)
		出力	補間値あるいは λ 階微分値あるいは積分値. (引数“isw”の項参照)
vw	double vw[m+1]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	$x[i] \leq v < x[i+1]$ が満たされていない.	左記の i をさがして処理を続ける.
30000	次のいずれかであった. $v < x[0]$ 又は $v > x[n-1]$ $isw < -1$ 又は $isw > m$	処理を打ち切る.

3. 使用上の注意

a) c_dbic4 との関係

本関数は, 関数 c_dbic4 により求められた B-spline 補間式 (1) に基づいて, 補間値あるいは微分値あるいは積分値を求めるものである. したがって, 本関数を呼び出す前に, 関数 c_dbic4 を呼び出して補間式 (1) を求め, 続けて本関数を呼び出すことにより補間値などを求めることができる. そのとき, 引数 x , n , m , c は c_dbic4 のそれらと同一でなければならない.

b) i

引数 i は $x[i] \leq v < x[i+1]$ を満足していることが好ましい. もし満足されていないときは $x[i] \leq v < x[i+1]$ を満足するような i を探索して処理を続ける.

ここで, 数学で一般に使用する添え字と C 言語の配列の添え字では, 値が 1 ずれることに注意する必要がある. すなわち, C 言語の添え字は 0 から始まり, 数学の添え字は 1 から始まるため i は $i-1$ を設定する.

4. 使用例

区間 $[0, 2\pi]$ で等間隔な 10 個の補間点での関数 $f(x) = \sin x$ の B-spline 補間式の補間係数を求め, ある点での補間値, 微分値及び積分値を求める.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define N 10
#define M 3

MAIN__()
{
    int ierr, icon;
    int i, n, m, isw;
    double x[N], y[N], c[N+M-1], vw[49];
    double p, h, v, f, pi;

    /* initialize data */
    n = N;
    m = M;
    p = 0;
    pi = 2*asin(1);
    h = 2*pi/(n-1);
    /* set function values */
    for (i=0; i<n; i++) {
        x[i] = p+i*h;
        y[i] = sin(x[i]);
    }
    /* calculate B-spline interpolation coefficients */
    ierr = c_dbic4(x, y, n, m, c, vw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dbic4 failed with icon = %d\n", icon);
        exit(1);
    }
    i = 4;
    v = pi;
    for (isw=-1; isw<=m; isw++) {
        /* calculate value at point */

```

```
ierr = c_dbif4(x, n, m, c, isw, v, &i, &f, vw, &icon);
if (icon >= 20000) {
    printf("ERROR: c_dbif4 failed with icon = %d\n", icon);
    exit(1);
}
if (isw == -1)
    printf("icon = %i    integral = %12.6e\n", icon, f);
else if (isw == 0)
    printf("icon = %i    value = %12.6e\n", icon, f);
else
    printf("icon = %i    derivative %i = %12.6e\n", icon, isw, f);
}
return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の BIF4 の項目を参照のこと.

c_dbifd1

B-spline 2 次元補間式 (I-I) による補間, 数値微分, 数値積分

```
ierr = c_dbifd1(x, nx, y, ny, m, c, k, iswx,
                vx, &ix, iswy, vy, &iy, &f, vw,
                &icon);
```

1. 機能

xy 平面上の格子点 (x_i, y_j) , $(x_1 < x_2 < \dots < x_{n_x}, y_1 < y_2 < \dots < y_{n_y})$ において関数値 $f_{ij} = f(x_i, y_j)$ が与えられ, 更に周囲において偏微係数,

$$\begin{aligned} f_{1,j}^{(\lambda,0)} &= f^{(\lambda,0)}(x_1, y_j), & f_{n_x,j}^{(\lambda,0)} &= f^{(\lambda,0)}(x_{n_x}, y_j) \\ f_{i,1}^{(0,\mu)} &= f^{(0,\mu)}(x_i, y_1), & f_{i,n_y}^{(0,\mu)} &= f^{(0,\mu)}(x_i, y_{n_y}) \\ f_{1,1}^{(\lambda,\mu)} &= f^{(\lambda,\mu)}(x_1, y_1), & \dots & f_{n_x,1}^{(\lambda,\mu)} = f^{(\lambda,\mu)}(x_{n_x}, y_1) \\ f_{1,n_y}^{(\lambda,\mu)} &= f^{(\lambda,\mu)}(x_1, y_{n_y}), & f_{n_x,n_y}^{(\lambda,\mu)} &= f^{(\lambda,\mu)}(x_{n_x}, y_{n_y}) \end{aligned}$$

が与えられたとき点 $P(v_x, v_y)$ における補間値あるいは偏微分値あるいは領域 $\{(x,y) \mid x_1 \leq x \leq v_x, y_1 \leq y \leq v_y\}$ 上の二重積分値を求める.

ただし, 本関数を利用する前に, 関数 c_dbicd1 により B-spline2 次元補間式,

$$S(x, y) = \sum_{\beta=1-m}^{n_y-1} \sum_{\alpha=1-m}^{n_x-1} c_{\alpha,\beta} N_{\alpha,m+1}(x) N_{\beta,m+1}(y) \quad (1)$$

における補間係数 $c_{\alpha,\beta}$ が計算されているとする. ここで m は奇数で B-spline $N_{\alpha,m+1}(x)$ 及び $N_{\beta,m+1}(y)$ の次数である. $x_1 \leq v_x \leq x_{n_x}$, $y_1 \leq v_y \leq y_{n_y}$, $m \geq 3$ 及び $n_x \geq 2$, $n_y \geq 2$ であること.

2. 引数

呼出し形式:

```
ierr = c_dbifd1(x, nx, y, ny, m, (double*)c, k, iswx, vx, &ix, iswy, vy, &iy,
                &f, vw, &icon);
```

引数の説明:

x	double x[nx]	入力	x 方向の離散点 x_i
nx	int	入力	x_i の個数 n_x .
y	double y[ny]	入力	y 方向の離散点 y_j .
ny	int	入力	y_j の個数 n_y .
m	int	入力	B-spline の次数 m .
c	double c[Clen][k]	入力	補間係数 $c_{\alpha,\beta}$ (c_dbicd1 の出力). $Clen = n_x + m - 1$
k	int	入力	2 次元配列 c の整合寸法 ($\geq ny + m - 1$).
iswx	int	入力	x 方向に関する計算の種類を与える. $-1 \leq iswx \leq m$. (引数 f の項参照)
vx	double	入力	点 $P(v_x, v_y)$ の x 座標.
ix	int	入力	$x[ix] \leq vx < x[ix+1]$ を満たす ix. $v_x = x_{n_x}$ のときは $ix = n_x - 2$ とする.
		出力	$x[ix] \leq vx < x[ix+1]$ を満たす ix. (“使用上の注意”b)参照)
iswy	int	入力	y 方向に関する計算の種類を与える.

			$-1 \leq \text{iswy} \leq m$. (“f”の項参照)
vy	double	入力	点 $P(v_x, v_y)$ の y 座標.
iy	int	入力	$y[\text{iy}] \leq v_y < y[\text{iy}+1]$ を満たす整数. $v_y = y_{n_y}$ のときは $\text{iy} = n_y - 2$ とする.
		出力	$y[\text{iy}] \leq v_y < y[\text{iy}+1]$ を満たす整数. (“使用上の注意”b)参照)
f	double	出力	補間値あるいは偏微分値あるいは積分値. $\text{iswx} = \lambda, \text{iswy} = \mu$ とすると λ, μ の組合せごとに次の値を出力する. $\lambda, \mu \geq 0$ のとき $f = \frac{\partial^{\lambda+\mu}}{\partial x^\lambda \partial y^\mu} S(v_x, v_y)$ 補間値は $\lambda = \mu = 0$ とすることにより得られる. $\lambda = -1, \mu \geq 0$ のとき $f = \int_{x_1}^{v_x} \frac{\partial^\mu}{\partial y^\mu} S(x, v_y) dx$ $\lambda \geq 0, \mu = -1$ のとき $f = \int_{y_1}^{v_y} \frac{\partial^\lambda}{\partial x^\lambda} S(v_x, y) dy$ $\lambda = \mu = -1$ のとき $f = \int_{y_1}^{v_y} \int_{x_1}^{v_x} S(x, y) dx dy$
vw	double vw[Vwlen]	作業領域	$Vwlen = 4(m+1) + \max(n_x, n_y) + m - 1$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	$x[\text{ix}] \leq v_x < x[\text{ix}+1]$ 又は $y[\text{iy}] \leq v_y < y[\text{iy}+1]$ が満たされていない.	ルーチン内で左記の ix 又は iy をさがして処理を続ける.
30000	次のいずれかであった. $v_x < x[0]$ 又は $v_x > x[n_x-1]$ $v_y < y[0]$ 又は $v_y > y[n_y-1]$ $\text{iswx} < -1$ 又は $\text{iswx} > m$ $\text{iswy} < -1$ 又は $\text{iswy} > m$	処理を打ち切る.

3. 使用上の注意

a) c_dbicd1 との関係

本関数は、関数 c_dbicd1 により求められた B-spline2 次元補間式 (1) に基づいて、補間値あるいは偏微分値あるいは二重積分値を求めるものである。したがって、本関数を呼び出す前に c_dbicd1 を呼び出して補間式 (1) を求め、続けて本関数を呼び出すことにより補間値などを求めることができる。そのとき、引数 x, nx, y, ny, k, m, c の値は c_dbicd1 のそれらと同一でなければならない。

b) ix 及び iy

引数 ix, iy はそれぞれ $x[\text{ix}] \leq v_x < x[\text{ix}+1]$, $y[\text{iy}] \leq v_y < y[\text{iy}+1]$ を満足していることが望ましい。もし満足していないときは、これらを満足するような ix, iy を探索して処理を続ける。

ここで、数学で一般に使用する添え字と C 言語の配列の添え字では、値が 1 ずれることに注意する必要がある。すなわち、C 言語の添え字は 0 から始まり、数学の添え字は 1 から始まるため i_x, i_y は $i-1, j-1$ を設定する。

4. 使用例

区間 $[0,1] \times [0,1]$ で等間隔な 10 個の補間点での関数 $f(x,y) = x^3 y^3$ の B-spline 補間式の補間係数を求め、ある点での補間値、微分値及び積分値を求める。

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define N 10
#define M 3

/* function prototype for initializer */
void gen(double x[], double y[], int n, double fxy[][N+M-1]);

MAIN__()
{
    int ierr, icon;
    int i, j, nx, ny, m, k, ix, iy, iswx, iswy;
    double x[N], y[N], fxy[N+M-1][N+M-1], c[N+M-1][N+M-1];
    double vw[60];
    double hx, hy, px, py, vx, vy, f;

    /* initialize data */
    nx = N;
    ny = N;
    m = M;
    k = N+M-1;
    hx = 1.0/(nx-1);
    hy = 1.0/(ny-1);
    px = 0;
    for (i=0; i<nx; i++) {
        x[i] = px+i*hx;
    }
    py = 0;
    for (j=0; j<ny; j++) {
        y[j] = py+j*hy;
    }
    /* generate function and derivative values in fxy */
    gen(x, y, nx, fxy);

    /* calculate B-spline interpolation coefficients */
    ierr = c_dbifd1(x, nx, y, ny, (double*)fxy, k,
                   m, (double*)c, vw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dbifd1 failed with icon = %d\n", icon);
        exit(1);
    }
    ix = 4;
    vx = 0.5;
    iy = 4;
    vy = 0.5;
    for (iswx=-1; iswx<=m; iswx++) {
        iswy = iswx;
        /* calculate value at point */
        ierr = c_dbifd1(x, nx, y, ny, m, (double*)c, k,
                       iswx, vx, &ix, iswy, vy, &iy, &f, vw, &icon);
        if (icon >= 20000) {
            printf("ERROR: c_dbifd1 failed with icon = %d\n", icon);
            exit(1);
        }
        if (iswx == -1)
            printf("icon = %i    integral = %12.6e\n", icon, f);
        else if (iswx == 0)
            printf("icon = %i    value = %12.6e\n", icon, f);
        else
            printf("icon = %i    derivative %i = %12.6e\n", icon, iswx, f);
    }
}
```

```
    return(0);
}

/* generate function and derivative values for f=x^3y^3 */
void gen(double x[], double y[], int n, double fxy[][N+M-1])
{
    double y1, yn, x1, xn, fx, fy;
    int i, j;

    /* corner points; df/dxdy values */
    fxy[0][0] = 9*x[0]*x[0]*y[0]*y[0];
    fxy[n+1][0] = 9*x[n-1]*x[n-1]*y[0]*y[0];
    fxy[0][n+1] = 9*x[0]*x[0]*y[n-1]*y[n-1];
    fxy[n+1][n+1] = 9*x[n-1]*x[n-1]*y[n-1]*y[n-1];

    /* partial derivatives on edges: df/dx, df/dy */
    y1 = y[0]*y[0]*3;
    yn = y[n-1]*y[n-1]*3;
    x1 = x[0]*x[0]*3;
    xn = x[n-1]*x[n-1]*3;

    /* edges; fx.df/dy or fy.df/dx */
    for (i=0;i<n;i++) {
        fx = x[i]*x[i]*x[i];
        fy = y[i]*y[i]*y[i];
        fxy[i+1][0] = y1*fx;
        fxy[0][i+1] = x1*fy;
        fxy[n+1][i+1] = xn*fy;
        fxy[i+1][n+1] = yn*fx;
    }

    /* central area; function values */
    for (i=0;i<n;i++) {
        fx = x[i]*x[i]*x[i];
        for (j=0;j<n;j++) {
            fxy[i+1][j+1] = fx*y[j]*y[j]*y[j];
        }
    }
    return;
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の BIFD1 の項目を参照のこと。

c_dbifd3

B-spline 2 次元補間式 (III-III) による補間, 数値微分, 数値積分

```
ierr = c_dbifd3(x, nx, y, ny, m, c, k, xt,
               iswx, vx, &ix, iswy, vy, &iy, &f,
               vw, &icon);
```

1. 機能

xy 平面上の格子点 (x_i, y_j) , $(x_1 < x_2 < \dots < x_{n_x}, y_1 < y_2 < \dots < y_{n_y})$ において関数値, $f_{ij} = f(x_i, y_j)$ が与えられたとき, 点 $P(v_x, v_y)$ における補間値あるいは偏微分値あるいは領域 $[x_1 \leq x \leq v_x, y_1 \leq y \leq v_y]$ 上の二重積分値を求める.

ただし, 本関数を利用する前に, 関数 c_bicd3 により, x 方向の節点列 $\{\xi_i\}$ 及び y 方向の節点列 $\{\eta_j\}$, 更に B-spline 2 次元補間式,

$$S(x, y) = \sum_{\beta=1-m}^{n_y-m} \sum_{\alpha=1-m}^{n_x-m} c_{\alpha,\beta} N_{\alpha,m+1}(x) N_{\beta,m+1}(y) \quad (1)$$

における補間係数 $c_{\alpha,\beta}$ が計算されているとする. ここで m は奇数で B-spline $N_{\alpha,m+1}(x)$ 及び $N_{\beta,m+1}(y)$ の次数である. $x_1 \leq v_x \leq x_{n_x}, y_1 \leq v_y \leq y_{n_y}, m \geq 3$ 及び $n_x \geq m+2, n_y \geq m+2$ であること.

2. 引数

呼出し形式:

```
ierr = c_dbifd3(x, nx, y, ny, m, (double*)c, k, xt, iswx, vx, &ix, iswy, vy,
               &iy, &f, vw, &icon);
```

引数の説明:

x	double x[nx]	入力	x 方向の離散点 x_i .
nx	int	入力	x_i の個数 n_x .
y	double y[ny]	入力	y 方向の離散点 y_j .
ny	int	入力	y_j の個数 n_y .
m	int	入力	B-spline の次数 m .
c	double c[nx][k]	入力	補間係数 $c_{\alpha,\beta}$ (c_dbicd3 の出力).
k	int	入力	配列 c の整合寸法 ($\geq ny$).
xt	double xt[Xilen]	入力	x 方向, y 方向の節点列 (c_dbicd3 の出力). $Xilen = (nx-m+1) + (ny-m+1)$.
iswx	int	入力	x 方向に関する計算の種類を与える. $-1 \leq iswx \leq m$. (引数“f”の項参照)
vx	double	入力	点 $P(v_x, v_y)$ の x 座標.
ix	int	入力	$x_i \leq v_x < x_{i+1}$ のとき ix には $i-1$ を入力する. ix は $x[ix] \leq v_x < x[ix+1]$ を満たす. $v_x = x_{n_x}$ のときは $ix = n_x - 2$ とする.
		出力	$x_i \leq v_x < x_{i+1}$ のとき $i-1$ を出力する. (“使用上の注意”b) 参照)
iswy	int	入力	y 方向に関する計算の種類を与える.

			-1 ≤ iswy ≤ m. (引数“f”の項参照)
vy	double	入力	点 $P(v_x, v_y)$ の y 座標.
iy	int	入力	$y_j \leq v_y < y_{j+1}$ のとき iy には $j-1$ を入力する. iy は $y[iy] \leq v_y < y[iy+1]$ を満たす. $v_y = y_{n_y}$ のときは $iy = n_y - 2$ とする.
		出力	$y_j \leq v_y < y_{j+1}$ のとき $j-1$ を出力する. (“使用上の注意”b) 参照)
f	double	出力	補間値あるいは偏微分値あるいは積分値. $iswx = \lambda, iswy = \mu$, とすると λ, μ の組み合わせごとに次の値を出力する. $0 \leq \lambda, \mu$ のとき, $f = \frac{\partial^{\lambda+\mu}}{\partial x^\lambda \partial y^\mu} S(v_x, v_y)$ 補間値は $\lambda = \mu = 0$ とすることにより得られる. $\lambda = -1, 0 \leq \mu$ のとき, $f = \int_{x_1}^{v_x} \frac{\partial^\mu}{\partial y^\mu} S(x, v_y) dx$ $\lambda \geq 0, \mu = -1$ のとき, $f = \int_{y_1}^{v_y} \frac{\partial^\lambda}{\partial x^\lambda} S(v_x, y) dy$ $\lambda = \mu = -1$ のとき, $f = \int_{y_1}^{v_y} dy \int_{x_1}^{v_x} S(x, y) dx$
vw	double vw[Vwlen]	作業領域	$Vwlen = 4 \cdot (m+1) + \max(nx, ny)$
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	$x[ix] \leq vx < x[ix+1]$ 又は $y[iy] \leq vy < y[iy+1]$ が満たされていない.	本関数内で左記の ix 又は iy をさがして処理を続ける.
29000	作業用の領域が確保できなかった.	処理を打ち切る.
30000	次のいずれかであった. $vx < x[0]$ 又は $vx > x[nx-1]$ $vy < y[0]$ 又は $vy > y[ny-1]$ $iswx < -1$ 又は $iswx > m$ $iswy < -1$ 又は $iswy > m$	処理を打ち切る.

3. 使用上の注意

a) c_dbicd3 との関係

本関数は、関数 c_dbicd3 により、求められた B-spline 2 次元補間式(1) に基づいて、補間値あるいは偏微分値あるいは二重積分値を求めるものである。

したがって、本関数を呼び出す前に、関数 `c_dbicd3` を呼び出して補間式(1) を求め、続けて本関数を呼び出すことにより補間値などを求めることができる。そのとき、引数 `x`, `nx`, `y`, `ny`, `k`, `m`, `c` 及び `xt` は `c_dbicd3` のそれらと同一でなければならない。

b) `ix` 及び `iy`

引数 `ix` 及び `iy` はそれぞれ $x[ix] \leq vx < x[ix+1]$, $y[iy] \leq vy < y[iy+1]$ を満足していることが好ましい。もし満足されていないときは、これらを満足するような `ix`, `iy` を探索して処理を続ける。

ここで、数学で一般に使用する添え字と C 言語の配列の添え字では、値が 1 ずれることに注意する必要がある。すなわち、C 言語の添え字は 0 から始まり、数学の添え字は 1 から始まるため `ix`, `iy` は $i-1, j-1$ を設定する。

4. 使用例

$[0,1] \times [0,1]$ 区間で等間隔な 100 個の補間点での関数 $f(x,y) = \sin(xy)\sqrt{xy}$ の値が与えられたとき、ある点での補間値を求める。

```
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define N 10
#define M 3

MAIN__()
{
    int ierr, icon;
    int i, j, nx, ny, m, k, ix, iy, iswx, iswy;
    double x[N], y[N], fxy[N][N], c[N][N], xt[2*(N-M+1)];
    double vw[(N-2)*M+2*(M+1)+2*N];
    double hx, hy, px, py, vx, vy, f;

    /* initialize data */
    nx = N;
    ny = N;
    m = M;
    hx = 1.0/(nx-1);
    hy = 1.0/(ny-1);
    px = 0;
    for (i=0; i<nx; i++) {
        x[i] = px;
        px = px + hx;
    }
    py = 0;
    for (j=0; j<ny; j++) {
        y[j] = py;
        py = py + hy;
    }
    for (i=0; i<nx; i++)
        for (j=0; j<ny; j++) {
            px = x[i];
            py = y[j];
            fxy[i][j] = sin(px*py)*sqrt(px*py);
        }
    k = N;
    /* calculate B-spline interpolation coefficients */
    ierr = c_dbicd3(x, nx, y, ny, m, (double*)fxy, k,
                    m, (double*)c, xt, vw, &icon);

    ix = nx/2;
    vx = x[ix] + (x[ix+1]-x[ix])/2;
    iy = ny/2;
    vy = y[iy] + (y[iy+1]-y[iy])/2;
    for (iswx=-1; iswx<m; iswx++) {
        iswy = iswx;
        /* calculate value at point */
        ierr = c_dbifd3(x, nx, y, ny, m, (double*)c, k, xt,
                       iswx, vx, &ix, iswy, vy, &iy, &f, vw, &icon);
    }
}
```

```
    if (iswx == -1)
        printf("icon = %i    integral = %12.6e\n", icon, f);
    else if (iswx == 0)
        printf("icon = %i    value = %12.6e\n", icon, f);
    else
        printf("icon = %i    derivative %i = %12.6e\n", icon, iswx, f);
}
return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の BIFD3 の項目を参照のこと.

c_dbin

第1種整数次変形ベッセル関数 $I_n(x)$
ierr = c_dbin(x, n, &bi, &icon);

1. 機能

第1種整数次変形ベッセル関数 $I_n(x)$ の値,

$$I_n(x) = \sum_{k=0}^{\infty} \frac{(x/2)^{2k+n}}{k!(n+k)!}$$

をテイラー展開式及び漸化式を用いて計算する.

2. 引数

呼出し形式:

```
ierr = c_dbin(x, n, &bi, &icon);
```

引数の説明:

x	double	入力	独立変数 x .
n	int	入力	$I_n(x)$ の次数 n .
bi	double	出力	関数値 $I_n(x)$.
icon	int	出力	コンディションコード. 下の表を参照.

ただし, $n=0$ 又は, $n=1$ のときはそれぞれ関数 c_dbi0, c_dbi1 の icon に準ずる.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
20000	次のいずれかであった. $x > 100$ $1/8 \leq x < 1$ かつ $n \geq 19 x + 29$ $1 \leq x < 10$ かつ $n \geq 4.7 x + 43$ $10 \leq x \leq 100$ かつ $n \geq 183 x + 71$	bi = 0 とする.

3. 使用上の注意

a) 引数 x 及び 引数 n

計算途中のオーバーフロー, アンダフローを前もって防ぐため, コンディションコードで示した範囲の x 及び n はエラーとする.

b) $I_0(x)$ 及び $I_1(x)$ の計算について

$I_0(x)$ 及び $I_1(x)$ を計算するには, 本関数よりもそれぞれ, 関数 c_dbi0, 関数 c_dbi1 を直接用いた方がよい.

4. 使用例

$I_n(x)$ の値を x が 0 から 10 まで 1 おきに, n が 20 から 30 まで 10 おきに計算し数表を作る.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
    double x, bi;
    int i, n;

    for (n=20;n<=30;n=n+10)
        for (i=0;i<=10;i++) {
            x = (double)i;
            /* calculate Bessel function */
            ierr = c_dbin(x, n, &bi, &icon);
            if (icon == 0)
                printf("x = %4.2f    n = %i    bi = %e\n", x, n, bi);
            else
                printf("ERROR: x = %4.2f    n = %i    bi = %e    icon = %i\n",
                    x, n, bi, icon);
        }
    return(0);
}
```

5. 手法概要

ベッセル関数 $I_n(x)$ は, x の値によって計算方法が異なる. $I_{-n}(x) = I_n(x)$, $I_n(-x) = -I_n(x)$ であるから, 以下では $|n|, |x|$ をそれぞれ n, x として説明する.

- $0 \leq x < 1/8$ のとき, テイラー展開式を用いる.
- $1/8 \leq x \leq 100$ のとき, 漸化式を用いる.

計算方法の詳細については“富士通 SSL II 使用手引書”の BIN の項目及び [123], [133]を参照のこと.

c_dbir

第1種実数変形ベッセル関数 $I_\nu(x)$
<code>ierr = c_dbir(x, v, &bi, &icon);</code>

1. 機能

第1種実数変形ベッセル関数 $I_\nu(x)$ の値,

$$I_\nu(x) = \left(\frac{x}{2}\right)^\nu \sum_{k=0}^{\infty} \frac{(x^2/4)^k}{k! \Gamma(\nu + k + 1)} \quad (1)$$

をべき級数展開式(1)及び漸化式を用いて計算する.

2. 引数

呼出し形式:

```
ierr = c_dbir(x, v, &bi, &icon);
```

引数の説明:

x	double	入力	独立変数 x .
v	double	入力	$I_\nu(x)$ の次数 ν .
bi	double	出力	関数値 $I_\nu(x)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
20000	$x > \log(fl_{\max})$ であった.	bi = 0 とする.
30000	$x < 0$ 又は $\nu < 0$ であった.	bi = 0 とする.

3. 使用上の注意

a) 引数 x 及び 引数 ν

計算途中のオーバーフロー, アンダフローを前もって防ぐため, コンディションコードで示した範囲の x 及び ν はエラーとする.

b) $I_0(x)$ 及び $I_1(x)$ の計算について

$I_0(x)$ 及び $I_1(x)$ を計算するには, 本関数よりもそれぞれ, 関数 c_dbir0, 関数 c_dbir1 を直接用いた方がよい.

c) 評価順序

$I_\nu(x), I_{\nu+1}(x), I_{\nu+2}(x), \dots, I_{\nu+M}(x)$ の値をいっせいに必要とするときには, 本関数により $I_{\nu+M}(x)$ と $I_{\nu+M-1}(x)$ を求め, 漸化式を反復適用して $I_{\nu+M-2}(x), I_{\nu+M-3}(x), \dots, I_\nu(x)$ と次数の低いものを求めていくとよい. 逆に, 本関数により $I_\nu(x)$ と $I_{\nu+1}(x)$ を求め, 漸化式により $I_{\nu+2}(x), I_{\nu+3}(x), \dots, I_{\nu+M}(x)$ と次数の高いものを求めていくことは不安定が生ずるので避けなければならない.

4. 使用例

$I_\nu(x)$ の値を x が 0 から 10 まで 1 おきに, ν が 0.4 から 0.6 まで 0.2 おきに計算し数表を作る.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
    double v, x, bi;
    int nv, i;

    for (i=0;i<=10;i++) {
        x = (double)i;
        for (nv=40;nv<=60;nv=nv+20) {
            v = (double)nv/100;
            /* calculate Bessel function */
            ierr = c_dbir(v, x, &bi, &icon);
            if (icon == 0)
                printf("x = %5.2f    v = %5.2f    bi = %e\n", x, v, bi);
            else
                printf("ERROR: x = %5.2f    v = %5.2f    bi = %e    icon = %i\n",
                    x, v, bi, icon);
        }
    }
    return(0);
}
```

5. 手法概要

ベッセル関数 $I_\nu(x)$ は, x の値によって計算方法が異なる.

- $0 \leq x \leq 1$ のとき, べき級数展開式(1)を用いる.
- $1 < x \leq \log(f_{\max})$ のとき, 漸化式を用いる.
 m を適当に大きな整数 (x, ν 及び要求精度によって決まる), δ を十分に小さい定数 (使用する計算機の正の最小数) とする. n 及び α を次式のように決める.

$$\nu = n + \alpha \quad (n: \text{整数}, 0 \leq \alpha < 1)$$

初期値を

$$G_{\alpha+m+1}(x) = 0, \quad G_{\alpha+m}(x) = \delta$$

として, $k = m, m-1, \dots, 1$ に対して漸化式

$$G_{\alpha+k-1}(x) = \frac{2(\alpha+k)}{x} G_{\alpha+k}(x) + G_{\alpha+k+1}(x)$$

を反復適用する. このとき, $I_\nu(x)$ は,

$$I_\nu(x) \approx \frac{\frac{1}{2} \left(\frac{x}{2}\right)^\alpha \frac{\Gamma(2\alpha+1)}{\Gamma(\alpha+1)} e^x G_{\alpha+n}(x)}{\sum_{k=0}^m \frac{(\alpha+k)\Gamma(2\alpha+k)}{k!} G_{\alpha+k}(x)}$$

として求められる.

計算方法の詳細については“富士通 SSL II 使用手引書”の BIR の項目及び[133]を参照のこと.

c_dbj0

第1種0次ベッセル関数 $J_0(x)$
<code>ierr = c_dbj0(x, &bj, &icon);</code>

1. 機能

第1種0次ベッセル関数 $J_0(x)$ の値

$$J_0(x) = \sum_{k=0}^{\infty} \frac{(-1)^k (x/2)^{2k}}{(k!)^2}$$

を有理式及び、漸近形式の近似式を用いて計算する。

2. 引数

呼出し形式:

```
ierr = c_dbj0(x, &bj, &icon);
```

引数の説明:

x	double	入力	独立変数 x .
bj	double	出力	関数値 $J_0(x)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
20000	$ x \geq t_{\max}$ であった.	$bj = 0$ とする.

3. 使用上の注意

a) 引数 x

引数 x の範囲は

$$|x| < t_{\max}$$

であること.

$|x|$ が大きくなると $\sin(x - \frac{\pi}{4})$, $\cos(x - \frac{\pi}{4})$ の値が不正確になるためである.

4. 使用例

$J_0(x)$ の値を x が 0 から 100 まで 1 おきに計算し数表を作る.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
    double x, bj;
```

```
int i;

for (i=0;i<=100;i++) {
    x = (double)i;
    /* calculate Bessel function */
    ierr = c_dbj0(x, &bj, &icon);
    if (icon == 0)
        printf("x = %4.2f    bj = %f\n", x, bj);
    else
        printf("ERROR: x = %4.2f    bj = %f    icon = %i\n", x, bj, icon);
}
return(0);
}
```

5. 手法概要

ベッセル関数 $J_0(x)$ は, x の値によって近似式の形が異なる. $J_0(-x) = J_0(x)$ であるから, 以下では $|x|$ を x として説明する.

- $0 \leq x \leq 8$ のとき, べき級数展開の有理近似式を用いて計算する.
- $x > 8$ のとき, 漸近展開の近似式を用いて計算する.

計算方法の詳細については“富士通 SSL II 使用手引書”のBJ0の項目及び[48]を参照のこと.

c_dbj1

第1種1次ベッセル関数 $J_1(x)$
<code>ierr = c_dbj1(x, &bj, &icon);</code>

1. 機能

第1種1次ベッセル関数 $J_1(x)$ の値

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

を有理式及び、漸近形式の近似式を用いて計算する。

2. 引数

呼出し形式:

```
ierr = c_dbj1(x, &bj, &icon);
```

引数の説明:

x	double	入力	独立変数 x .
bj	double	出力	関数値 $J_1(x)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	$ x \geq t_{\max}$ であった.	$bj = 0$ とする.

3. 使用上の注意

a) 引数 x

引数 x の範囲は

$$|x| < t_{\max}$$

であること.

$|x|$ が大きくなると $\sin(x - \frac{3\pi}{4})$, $\cos(x - \frac{3\pi}{4})$ の値が不正確になるためである.

4. 使用例

$J_1(x)$ の値を x が 0 から 100 まで 1 おきに計算し数表を作る.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
```

```
double x, bj;
int i;

for (i=0;i<=100;i++) {
    x = (double)i;
    /* calculate Bessel function */
    ierr = c_dbj1(x, &bj, &icon);
    if (icon == 0)
        printf("x = %4.2f    bj = %f\n", x, bj);
    else
        printf("ERROR: x = %4.2f    bj = %f    icon = %i\n", x, bj, icon);
}
return(0);
}
```

5. 手法概要

ベッセル関数 $J_1(x)$ は, x の値によって近似式の形が異なる. $J_1(-x) = -J_1(x)$ であるから, 以下では $|x|$ を x として説明する.

- $0 \leq x \leq 8$ のとき, べき級数展開の有理近似式を用いて計算する.
- $x > 8$ のとき, 漸近展開の近似式を用いて計算する.

計算方法の詳細については“富士通 SSL II 使用手引書”のBJ1の項目及び[48]を参照のこと.

c_dbjn

第1種整数次ベッセル関数 $J_n(x)$
ierr = c_dbjn(x, n, &bj, &icon);

1. 機能

第1種整数次ベッセル関数 $J_n(x)$ の値

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

をテーラー展開式及び、漸近形式を用いて計算する。

2. 引数

呼出し形式:

```
ierr = c_dbjn(x, n, &bj, &icon);
```

引数の説明:

x	double	入力	独立変数 x .
n	int	入力	$J_n(x)$ の次数 n .
bj	double	出力	関数値 $J_n(x)$.
icon	int	出力	コンディションコード. 下の表を参照.

ただし, $n=0$ 又は, $n=1$ のときはそれぞれ関数 c_dbj0, c_dbj1 の icon に準ずる.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	次のいずれかであった. $x > 100$ $1/8 \leq x < 1$ かつ $n \geq 19 x + 29$ $1 \leq x < 10$ かつ $n \geq 4.7 x + 43$ $10 \leq x \leq 100$ かつ $n \geq 1.83 x + 71$	bj = 0 とする.

3. 使用上の注意

a) 引数 x 及び 引数 n

計算途中のオーバーフロー, アンダフローを前もって防ぐため, コンディションコードで示した範囲の x 及び n はエラーとする.

b) $J_0(x)$ 及び $J_1(x)$ の計算について

$J_0(x)$ 及び $J_1(x)$ を計算するには, 本関数よりもそれぞれ, 関数 c_dbj0, 関数 c_dbj1 を直接用いた方がよい.

4. 使用例

$J_n(x)$ の値を x が 0 から 10 まで 1 おきに, n が 20 から 30 まで 10 おきに計算し数表を作る.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
    double x, bj;
    int i, n;

    for (n=20;n<=30;n=n+10)
        for (i=0;i<=10;i++) {
            x = (double)i;
            /* calculate Bessel function */
            ierr = c_dbjn(x, n, &bj, &icon);
            if (icon == 0)
                printf("x = %4.2f    n = %i    bj = %e\n", x, n, bj);
            else
                printf("ERROR: x = %4.2f    n = %i    bj = %e    icon = %i\n",
                    x, n, bj, icon);
        }
    return(0);
}
```

5. 手法概要

ベッセル関数 $J_n(x)$ は, x の値によって計算方法が異なる. $J_{-n}(x) = J_n(-x) = (-1)^n J_n(x)$ であるから, 以下では $|x|$ を x として説明する.

- $0 \leq x < 1/8$ のとき, テイラー展開式を用いる.
- $1/8 \leq x \leq 100$ のとき, 漸化式を用いる.

計算方法の詳細については“富士通 SSL II 使用手引書”の BJN の項目及び [123], [133]を参照のこと.

c_dbjr

第1種実数次ベッセル関数 $J_\nu(x)$
<code>ierr = c_dbjr(x, v, &bj, &icon);</code>

1. 機能

第1種実数次ベッセル関数 $J_\nu(x)$ の値

$$J_\nu(x) = \left(\frac{x}{2}\right)^\nu \sum_{k=0}^{\infty} \frac{(-x^2/4)^k}{k! \Gamma(\nu + k + 1)} \quad (1)$$

をべき級数展開式(1), 漸化式及び, 漸近展開式を用いて計算する.

2. 引数

呼出し形式:

```
ierr = c_dbjr(x, v, &bj, &icon);
```

引数の説明:

x	double	入力	独立変数 x .
v	double	入力	$J_\nu(x)$ の次数 ν .
bj	double	出力	関数値 $J_\nu(x)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
20000	次のいずれかであった. • $x > 100$ かつ $\nu > 15$ • $x \geq t_{\max}$	$bj = 0$ とする.
30000	$x < 0$ 又は $\nu < 0$ であった.	$bj = 0$ とする.

3. 使用上の注意

a) 引数 x 及び 引数 ν

x が大きくなると漸近展開式の中の $\sin(|x| - (\frac{1}{2}\nu + \frac{1}{4})\pi)$ 及び $\cos(|x| - (\frac{1}{2}\nu + \frac{1}{4})\pi)$ の値が正確に計算できなくなる.

b) $J_0(x)$ 及び $J_1(x)$ の計算について

$J_0(x)$ 及び $J_1(x)$ を計算するには, 本関数よりもそれぞれ, 関数 c_dbj0, 関数 c_dbj1 を直接用いた方がよい.

c) 評価順序

$J_\nu(x), J_{\nu+1}(x), J_{\nu+2}(x), \dots, J_{\nu+M}(x)$ の値をいっせいに必要とするときには, 本関数により $J_{\nu+M}(x)$ と $J_{\nu+M-1}(x)$ を求め, 漸化式を反復適用して $J_{\nu+M-2}(x), J_{\nu+M-3}(x), \dots, J_\nu(x)$ と次数の低いものを求めていくとよい. 逆に, 本関数により $J_\nu(x)$ と $J_{\nu+1}(x)$ を求め, 漸化式により $J_{\nu+2}(x), J_{\nu+3}(x), \dots, J_{\nu+M}(x)$ と次数の高いものを求めていくことは不安定が生ずるので避けなければならない.

4. 使用例

$J_\nu(x)$ の値を x が 0 から 10 まで 1 おきに, ν が 0.4 から 0.6 まで 0.2 おきに計算し数表を作る.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
    double v, x, bj;
    int nv, i;

    for (i=0;i<=10;i++) {
        x = (double)i;
        for (nv=40;nv<=60;nv=nv+20) {
            v = (double)nv/100;
            /* calculate Bessel function */
            ierr = c_dbjr(v, x, &bj, &icon);
            if (icon == 0)
                printf("x = %5.2f    v = %5.2f    bj = %e\n", x, v, bj);
            else
                printf("ERROR: x = %5.2f    v = %5.2f    bj = %e    icon = %i\n",
                    x, v, bj, icon);
        }
    }
    return(0);
}
```

5. 手法概要

ベッセル関数 $J_\nu(x)$ は, x 及び ν の値によって計算方法が異なる.

- $0 \leq x < 1$ のとき, ベキ級数展開式(1)を用いる.
- $1 \leq x < 30$, 又は, $30 \leq x \leq 100$ かつ $\nu > 0.115x + 4$ のとき, 漸化式を用いる.
 m を適当に大きな整数 (x, ν 及び要求精度によって決まる), δ を十分に小さい定数 (使用する計算機の正の最小数) とする. n 及び α を次式のように決める.

$$\nu = n + \alpha \quad (n: \text{整数}, 0 \leq \alpha < 1)$$

初期値を

$$F_{\alpha+m+1}(x) = 0, \quad F_{\alpha+m}(x) = \delta$$

として, $k = m, m-1, \dots, 1$ に対して漸化式

$$F_{\alpha+k-1}(x) = \frac{2(\alpha+k)}{x} F_{\alpha+k}(x) - F_{\alpha+k+1}(x)$$

を反復適用する. このとき, $J_\nu(x)$ は

$$J_\nu(x) \approx \frac{\left(\frac{x}{2}\right)^\alpha F_{\alpha+n}(x)}{\sum_{k=0}^{m/2} \frac{(\alpha+2k)\Gamma(\alpha+k)}{k!} F_{\alpha+2k}(x)}$$

として求められる.

- $100 < x < t_{\max}$ かつ $\nu \leq 15$, 又は $30 \leq x \leq 100$ かつ $\nu \leq 0.115x + 4$ のとき, 漸近展開式を用いる.

計算方法の詳細については“富士通 SSL II 使用手引書”のBJRの項目及び[133]を参照のこと.

c_dbk0

第2種0次変形ベッセル関数 $K_0(x)$
ierr = c_dbk0(x, &bk, &icon);

1. 機能

第2種0次変形ベッセル関数 $K_0(x)$ の値,

$$K_0(x) = \sum_{k=1}^{\infty} \frac{(x/2)^{2k}}{(k!)^2} \left(\sum_{m=1}^k \frac{1}{m} \right) - I_0(x) [\gamma + \log(x/2)] \quad (1)$$

を多項式及び漸近形式の近似式を用いて計算する. ただし, $x > 0$ であること.

($I_0(x)$: 第1種0次変形ベッセル関数, γ : オイラー一定数)

2. 引数

呼出し形式:

```
ierr = c_dbk0(x, &bk, &icon);
```

引数の説明:

x	double	入力	独立変数 x .
bk	double	出力	関数値 $K_0(x)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
20000	$x > \log(fl_{\max})$ であった.	bk = 0 とする.
30000	$x \leq 0$ であった.	bk = 0 とする.

3. 使用上の注意

a) 引数 x

引数 x の範囲は

$$0 < x \leq \log(fl_{\max})$$

であること.

x が上記範囲を超えると e^{-x} の計算でアンダフローする. これを本関数内で前もって防ぐためである.

4. 使用例

$K_0(x)$ の値を x が 0 から 100 まで 1 おきに計算し数表を作る.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
```

```
{
  int ierr, icon;
  double x, bk;
  int i;

  for (i=1;i<=100;i++) {
    x = (double)i;
    /* calculate Bessel function */
    ierr = c_dbk0(x, &bk, &icon);
    if (icon == 0)
      printf("x = %4.2f    bk = %e\n", x, bk);
    else
      printf("ERROR: x = %4.2f    bk = %e    icon = %i\n", x, bk, icon);
  }
  return(0);
}
```

5. 手法概要

変形ベッセル関数 $K_0(x)$ は, x の値によって近似式の形が異なる.

- $0 < x < 2$ のとき, べき級数展開の近似式を用いて計算する.
- $2 \leq x \leq \log(f_{\max})$ のとき, 漸近展開の近似式を用いて計算する.

計算方法の詳細については“富士通 SSL II 使用手引書”のBK0の項目を参照のこと.

c_dbk1

第2種1次変形ベッセル関数 $K_1(x)$
ierr = c_dbk1(x, &bk, &icon);

1. 機能

第2種1次変形ベッセル関数 $K_1(x)$ の値,

$$K_1(x) = I_1(x) \left[\gamma + \log(x/2) \right] + \frac{1}{x} - \frac{1}{2} \sum_{k=0}^{\infty} \frac{(x/2)^{2k+1}}{k!(k+1)!} \left(\sum_{m=1}^k \frac{1}{m} + \sum_{m=1}^{k+1} \frac{1}{m} \right) \quad (1)$$

を多項式及び漸近形式の近似式を用いて計算する. ただし, $x > 0$ であること.

($I_1(x)$: 第1種1次変形ベッセル関数, γ : オイラー定数)

2. 引数

呼出し形式:

```
ierr = c_dbk1(x, &bk, &icon);
```

引数の説明:

x	double	入力	独立変数 x .
bk	double	出力	関数値 $K_1(x)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
20000	$x > \log(fl_{\max})$ であった.	bk = 0 とする.
30000	$x \leq 0$ であった.	bk = 0 とする.

3. 使用上の注意

a) 引数 x

引数 x の範囲は

$$0 < x \leq \log(fl_{\max})$$

であること.

x が上記範囲を超えると e^{-x} の計算でアンダフローを生じる. これを本関数内で前もって防ぐためである.

4. 使用例

$K_1(x)$ の値を x が 0 から 100 まで 1 おきに計算し数表を作る.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
```

```
{
  int ierr, icon;
  double x, bk;
  int i;

  for (i=1;i<=100;i++) {
    x = (double)i;
    /* calculate Bessel function */
    ierr = c_dbk1(x, &bk, &icon);
    if (icon == 0)
      printf("x = %4.2f    bk = %e\n", x, bk);
    else
      printf("ERROR: x = %4.2f    bk = %e    icon = %i\n", x, bk, icon);
  }
  return(0);
}
```

5. 手法概要

変形ベッセル関数 $K_1(x)$ は, x の値によって近似式の形が異なる.

- $0 < x < 2$ のとき, べき級数展開の近似式を用いて計算する.
- $2 \leq x \leq \log(f_{\max})$ のとき, 漸近展開の近似式を用いて計算する.

計算方法の詳細については“富士通 SSL II 使用手引書”のBK1の項目を参照のこと.

c_dbkn

第2種整数次変形ベッセル関数 $K_n(x)$
<code>ierr = c_dbkn(x, n, &bk, &icon);</code>

1. 機能

第2種整数次変形ベッセル関数 $K_n(x)$ の値,

$$K_n(x) = (-1)^{n+1} I_n(x) [\gamma + \log(x/2)] + \frac{1}{2} \sum_{k=0}^{n-1} \frac{(-1)^k (n-k-1)!}{k!} (x/2)^{2k-n} \\ + \frac{(-1)^n}{2} \sum_{k=0}^{\infty} \frac{(x/2)^{n+2k}}{k!(n+k)!} (\phi_k + \phi_{k+n}) \quad (1)$$

を漸化式を用いて計算する。ただし、上式において、 $x > 0$ であり、 γ はオイラー一定数、 $I_n(x)$ は第1種整数次変形ベッセル関数、また、

$$\phi_0 = 0 \\ \phi_L = \sum_{m=1}^L \frac{1}{m} \quad (L \geq 1)$$

である。

2. 引数

呼出し形式:

```
ierr = c_dbkn(x, n, &bk, &icon);
```

引数の説明:

x	double	入力	独立変数 x .
n	int	入力	$K_n(x)$ の次数 n .
bk	double	出力	関数値 $K_n(x)$.
icon	int	出力	コンディションコード. 下の表を参照.

ただし、 $n=0$ 又は、 $n=1$ のときはそれぞれ関数 `c_dbk0`, `c_dbk1` の `icon` に準ずる。

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	$x > \log(fl_{\max})$ であった.	<code>bk = 0</code> とする.
30000	$x \leq 0$ であった.	<code>bk = 0</code> とする.

3. 使用上の注意

a) 引数 x

引数 x の範囲は

$$0 < x \leq \log(f_{\max})$$

であること.

x が上記範囲を超えると e^{-x} の計算でアンダフローする. これを本関数内で前もって防ぐためである.

b) $K_0(x)$ 及び $K_1(x)$ の計算について

$K_0(x)$ 及び $K_1(x)$ を計算するには, 本関数よりもそれぞれ, 関数 c_dbk0, 関数 c_dbk1 を直接用いた方がよい.

4. 使用例

$K_n(x)$ の値を x が 0 から 10 まで 1 おきに, n を 20 から 30 まで 10 おきに計算し数表を作る.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
    double x, bk;
    int i, n;

    for (n=20;n<=30;n=n+10)
        for (i=1;i<=10;i++) {
            x = (double)i;
            /* calculate Bessel function */
            ierr = c_dbkn(x, n, &bk, &icon);
            if (icon == 0)
                printf("x = %4.2f    n = %i    bk = %e\n", x, n, bk);
            else
                printf("ERROR: x = %4.2f    n = %i    bk = %e    icon = %i\n",
                    x, n, bk, icon);
        }
    return(0);
}
```

5. 手法概要

ベッセル関数 $K_n(x)$ は漸化式を用いて計算する. ただし, $K_0(x)$ 及び $K_1(x)$ は, それぞれ Fortran ルーチンである DBK0, DBK1 を利用している.

計算方法の詳細については“富士通 SSL II 使用手引書”の BKN の項目を参照のこと.

c_dbkr

第2種実数変形ベッセル関数 $K_\nu(x)$
<code>ierr = c_dbkr(x, v, &bk, &icon);</code>

1. 機能

第2種実数変形ベッセル関数 $K_\nu(x)$ の値

$$K_\nu(x) = \frac{\pi [I_{-\nu}(x) - I_\nu(x)]}{2 \sin(\nu\pi)} \quad (1)$$

を吉田・二宮による方法で計算する.

$I_\nu(x)$ は第1種変形ベッセル関数である. $x > 0$ であること.

2. 引数

呼出し形式:

```
ierr = c_dbkr(x, v, &bk, &icon);
```

引数の説明:

x	double	入力	独立変数 x .
v	double	入力	$K_\nu(x)$ の次数 ν .
bk	double	出力	関数値 $K_\nu(x)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	$x = 0$ であった. 又は, bk がオーバーフローするほど大きな値となった.	bk には浮動小数点数の最大値を出力する.
30000	$x < 0$ であった.	bk = 0 とする.

3. 使用上の注意

a) $K_0(x)$ 及び $K_1(x)$ の計算について

$K_0(x)$ 及び $K_1(x)$ を計算するには, 本関数よりもそれぞれ, 関数 c_dbk0, 関数 c_dbk1 を直接用いた方が効率が良い.

b) 評価順序

$K_\nu(x), K_{\nu+1}(x), K_{\nu+2}(x), \dots, K_{\nu+M}(x)$ の値をいっせいに必要とするときには, 本関数により $K_\nu(x)$ と $K_{\nu+1}(x)$ を求め, 次の漸化式を反復適用して $K_{\nu+2}(x), K_{\nu+3}(x), \dots, K_{\nu+M}(x)$ と次数の高いものへと順次求めていくとよい.

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

c) 計算の効率化

x が大きいところでは, 続けて同じ v の値で, 本関数が呼ばれるときには, 計算の共通部分を通過し, 能率良く $K_v(x)$ の値を計算できるようにしてある.

4. 使用例

$K_v(x)$ の値を x が 0 から 10 まで 1 おきに, v を 0.4 から 0.6 まで 0.2 おきに計算し数表を作る.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
    double v, x, bk;
    int nv, i;

    for (i=1; i<=10; i++) {
        x = (double)i;
        for (nv=40; nv<=60; nv=nv+20) {
            v = (double)nv/100;
            /* calculate Bessel function */
            ierr = c_dbkr(v, x, &bk, &icon);
            if (icon == 0)
                printf("x = %5.2f    v = %5.2f    bk = %e\n", x, v, bk);
            else
                printf("ERROR: x = %5.2f    v = %5.2f    bk = %e    icon = %i\n",
                    x, v, bk, icon);
        }
    }
    return(0);
}
```

5. 手法概要

第 2 種実数変形ベッセル関数 $K_v(x)$ は吉田・二宮による方法を使用して計算する.

計算方法の詳細については“富士通 SSL II 使用手引書”の BKR の項目及び[143]を参照のこと.

c_dblnc

実行列の平衡化

ierr = c_dblnc(a, k, n, dv, &icon);

1. 機能

n 次の実行列 $\mathbf{A}$ を (1) の対角相似変換により, 平衡化する.

$$\tilde{\mathbf{A}} = \mathbf{D}^{-1} \mathbf{A} \mathbf{D} \quad (1)$$

ここで, 平衡化するとは, 変換後の実行列 $\tilde{\mathbf{A}}$ について第 i 行 ($i=1, \dots, n$) 要素のノルムの和と, 第 i 列要素のノルムの和をほぼ等しくすることである. また, $\mathbf{D}$ は対角行列である. $n \geq 1$ なること.

2. 引数

呼出し形式:

```
ierr = c_dblnc((double *) a, k, n, dv, &icon);
```

引数の説明:

a	double	入力	実行列 $\mathbf{A}$.
	a[n][k]	出力	平衡化後の実行列 $\tilde{\mathbf{A}}$.
k	int	入力	配列 a の整合寸法 ($\geq n$).
n	int	入力	実行列 $\mathbf{A}$ 及び $\tilde{\mathbf{A}}$ の次数 n .
dv	double dv[n]	出力	スケーリングファクタ ($\mathbf{D}$ の対角要素).
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	$n=1$ であった.	平衡化は行わない.
30000	次のいずれかであった: • $n < 1$ • $k < n$	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

行列の各要素の大きさが不揃いな場合, この行列の固有値及び固有ベクトルの計算値の精度が悪くなりやすい. 本関数は, これを防ぐためのものである.

行列の各要素の大きさがほぼ揃っているときには, 本関数により何等の変換も行われないので, 本関数の実行を省略した方がよい.

本関数は行列の平衡化にあたり, 行又は列の要素が, 対角要素を除いてすべて零となるような行 (又は列) がある場合, その行 (又は列) と対応する列 (又は行) については平衡化を省略する.

b) 実行列 $\mathbf{A}$ の固有ベクトル $\mathbf{x}$

実行列 $\mathbf{A}$ の固有ベクトル $\mathbf{x}$ を求めるときは, 本関数により平衡化した行列 $\tilde{\mathbf{A}}$ の固有ベクトル $\tilde{\mathbf{x}}$ に対して, (2) の逆変換を行う必要がある.

$$\mathbf{x} = \mathbf{D} \tilde{\mathbf{x}}. \quad (2)$$

(2) の逆変換は関数 c_dhbk1 により実行できる.

4. 使用例

n 次の実行列 A を平衡化した後、関数 `c_dhes1` により実ヘッセンベルグ行列に変換し、固有値を関数 `c_dhsqr` で求め、固有ベクトルを `c_dhvec` 及び `c_dhbk1` で求める。

```
#include <stdio.h>
#include <stdlib.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 5

MAIN__()
{
    int ierr, icon;
    int n, i, j, k, m, mk, ind[NMAX];
    double a[NMAX][NMAX], pv[NMAX], aw[NMAX+4][NMAX];
    double er[NMAX], ei[NMAX], ev[NMAX][NMAX];
    double dv[NMAX];

    /* initialize matrix */
    n = NMAX;
    k = NMAX;
    mk = NMAX;
    for (i=0; i<n; i++) {
        a[i][i] = n-i;
        for (j=0; j<i; j++) {
            a[i][j] = n-i;
            a[j][i] = n-i;
        }
    }
    /* balance matrix A */
    ierr = c_dblnc((double*)a, k, n, dv, &icon);
    if (icon > 10000 ) {
        printf("ERROR: c_dblnc failed with icon = %i\n", icon);
        exit (1);
    }
    /* reduce matrix to Hessenberg form */
    ierr = c_dhes1((double*)a, k, n, pv, &icon);
    if (icon != 0 ) {
        printf("ERROR: c_dhes1 failed with icon = %i\n", icon);
        exit (1);
    }
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            aw[i][j] = a[i][j];
    /* find eigenvalues */
    ierr = c_dhsqr((double*)aw, k, n, er, ei, &m, &icon);
    if (icon >= 20000 ) {
        printf("ERROR: c_dhsqr failed with icon = %i\n", icon);
        exit (1);
    }
    for (i=0; i<m; i++) ind[i] = 1;
    /* find eigenvectors for given eigenvalues */
    ierr = c_dhvec((double*)a, k, n, er, ei,
                  ind, m, (double*)ev, mk, (double*)aw, &icon);
    if (icon >= 20000 ) {
        printf("ERROR: c_dhvec failed with icon = %i\n", icon);
        exit (1);
    }
    /* back transformation to find e-vectors of A */
    ierr = c_dhbk1((double*)ev, k, n, ind, m, (double*)a, pv, dv, &icon);
    if (icon > 10000 ) {
        printf("ERROR: c_dhbk1 failed with icon = %i\n", icon);
        exit (1);
    }
    printf("icon = %i\n", icon);
    /* print eigenvalues and eigenvectors */
    i = 0;
    k = 0;
    while (i<m) {
        if (ind[i] == 0) i++;
        else if (ei[i] == 0) {
            /* real eigenvector */
            printf("eigenvalue: %12.4f\n", er[i]);
            printf("eigenvector:");
        }
    }
}
```



```

        for (j=0;j<n;j++)
            printf("%7.4f ", ev[k][j]);
        printf("\n");
        i++;
        k++;
    }
    else {
        /* complex eigenvector pair */
        printf("eigenvalue: %7.4f+i*%7.4f\n", er[i], ei[i]);
        printf("eigenvector: ");
        for (j=0;j<n;j++)
            printf("%7.4f+i*%7.4f ", ev[k][j], ev[k+1][j]);
        printf("\n");
        printf("eigenvalue: %7.4f+i*%7.4f\n", er[i+1], ei[i+1]);
        printf("eigenvector: ");
        for (j=0;j<n;j++)
            printf("%7.4f+i*%7.4f ", ev[k][j], -ev[k+1][j]);
        printf("\n");
        i = i+2;
        k = k+2;
    }
}
return(0);
}

```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の BLNC の項目及び[119]を参照のこと.

c_dbmdmx

MDM ^T 分解された実対称バンド行列の連立 1 次方程式

ierr = c_dbmdmx(b, fa, n, nh, mh, ip, ivw, &icon);

1. 機能

MDM^T 分解された実対称バンド行列を係数に持つ連立 1 次方程式,

$$\mathbf{P}^{-1} \mathbf{M} \mathbf{D} \mathbf{M}^T \mathbf{P}^{-T} \mathbf{x} = \mathbf{b} \quad (1)$$

を解く. ただし, $\mathbf{M}$ は $n \times n$, バンド幅 $\tilde{h}$ の単位下バンド行列, $\mathbf{D}$ は高々次数 2 の対称なブロックから成る対称ブロック対角行列, $\mathbf{P}$ は置換行列 (係数行列を MDM^T 分解するときのピボットティングによる行の入換えを行う), $\mathbf{b}$ は n 次元の実定数ベクトル, $\mathbf{x}$ は n 次元の解ベクトルである. ここで, $d_{k+1,k} \neq 0$ なら $m_{k+1,k} = 0$ である. ただし, $\mathbf{M}=(m_{ij})$, $\mathbf{D}=(d_{ij})$, $n > \tilde{h} \geq 0$ であること.

2. 引数

呼出し形式:

```
ierr = c_dbmdmx(b, fa, n, nh, mh, ip, ivw, &icon);
```

引数の説明:

b	double b[n]	入力	定数ベクトル $\mathbf{b}$.
		出力	解ベクトル $\mathbf{x}$.
fa	double fa[Falen]	入力	行列 $\mathbf{D} + (\mathbf{M} - \mathbf{I})$. 対称バンド行列用圧縮格納法で与える. 格納法の詳細については“1.3.4 バンド行列の格納方法”を参照のこと. $Falen = n(h_m + 1) - h_m(h_m + 1)/2$.
n	int	入力	行列 $\mathbf{M}$ 及び $\mathbf{D}$ の次数 n .
nh	int	入力	行列 $\mathbf{M}$ のバンド幅 $\tilde{h}$. (“使用上の注意”a)参照)
mh	int	入力	行列 $\mathbf{M}$ の最大許容バンド幅 h_m ($n > mh \geq nh$). (“使用上の注意”a)参照)
ip	int ip[n]	入力	ピボットティングによる行の入換えの履歴を示すトランスポジションベクトル. (“使用上の注意”a)参照)
ivw	int ivw[n]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	係数行列が非正則であった.	処理を打ち切る.
30000	次のいずれかであった. • $nh < 0$ • $mh < nh$ • $mh \geq n$ • ip に誤りがあった.	処理を打ち切る.

3. 使用上の注意

a) $\mathbf{fa}$, $\mathbf{nh}$, $\mathbf{ip}$, $\mathbf{mh}$ 及び $\mathbf{MDM}^T$ 分解

連立 1 次方程式を解く場合, 関数 `c_dsbmdm` を呼び出して, 係数行列を $\mathbf{MDM}^T$ 分解させてから, 本関数を呼び出せば方程式を解くことができる. しかし, 通常は関数 `c_dlsbix` を呼び出せば, 一度に解が求められる.

本関数の引数 $\mathbf{fa}$, $\mathbf{nh}$, $\mathbf{ip}$, $\mathbf{mh}$ は, `c_dsbmdm` の引数 $\mathbf{a}$, $\mathbf{nh}$, $\mathbf{ip}$, $\mathbf{mh}$ と同じである.

b) 行列式

行列 $\mathbf{A}$ の行列式は, 演算後の $\mathbf{D}$ の行列式に等しい. (`c_dsbmdm` の使用例参照)

c) 固有値

行列 $\mathbf{A}$ の正, 負固有値の数を各々求めることができる. (`c_dsbmdm` の使用例参照)

4. 使用例

実対称バンド行列を $\mathbf{MDM}^T$ 分解させてから, 連立 1 次方程式を解く.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define max(a,b) ((a) > (b) ? (a) : (b))
#define min(a,b) ((a) < (b) ? (a) : (b))

#define NMAX 100
#define NHMAX 50

MAIN__()
{
    int ierr, icon;
    int n, nh, mh, i, j, ij, jmin;
    double epsz, eps;
    double a[NMAX*(NHMAX+1)-NHMAX*(NHMAX+1)/2], b[NMAX], x[NMAX];
    int ivw[NMAX], ip[NMAX];

    /* initialize matrix */
    n = NMAX;
    nh = 2;
    mh = NHMAX;
    ij = 0;
    for (i=0;i<n;i++) {
        jmin = max(i-mh, 0);
        for (j=jmin;j<=i;j++)
            if (i-j == 0)
                a[ij++] = 10;
            else if (i-j == 1)
                a[ij++] = -3;
            else if (i-j == 2)
                a[ij++] = -6;
            else
                a[ij++] = 0;
    }
    epsz = 1e-6;
    /* initialize RHS vector */
    for (i=0;i<n;i++)
        x[i] = i+1;
    /* initialize constant vector b = a*x */
    ierr = c_dmsbv(a, n, mh, x, b, &icon);
    /* MDM decomposition of system */
    ierr = c_dsbmdm(a, n, &nh, mh, epsz, ip, ivw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dsbmdm failed with icon = %d\n", icon);
        exit(1);
    }
    /* solve decomposed system of equations */
    ierr = c_dbmdmx(b, a, n, nh, mh, ip, ivw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dbmdmx failed with icon = %d\n", icon);
    }
}
```

```
        exit(1);
    }
    /* check solution vector */
    eps = 1e-6;
    for (i=0;i<n;i++)
        if (fabs((x[i]-b[i])/b[i]) > eps) {
            printf("WARNING: result inaccurate\n");
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の BMDMX の項目を参照のこと。

c_dbsc1

B-spline 平滑化式 (固定節点)
<pre>ierr = c_dbsc1(x, y, w, n, m, xt, nt, c, r, &rnor, vw, ivw, &icon);</pre>

1. 機能

離散点 $x_1, x_2, \dots, x_n$ に対して観測値 $y_1, y_2, \dots, y_n$, 重み関数値 $w_i = w(x_i)$, $i = 1, 2, \dots, n$ 及び Spline 関数の節点 $\xi_1, \xi_2, \dots, \xi_{n_t}$ が与えられたとき, これに対する最小二乗近似の意味の B-spline 平滑化式を求める. すなわち, 求めるべき m (奇数でも偶数でもよい)次の B-spline 平滑化式を,

$$\bar{S}(x) = \sum_{j=1-m}^{n_t-1} c_j N_{j,m+1}(x) \quad (1)$$

とし, このとき重み付き残差の 2 乗和,

$$\delta_m^2 = \sum_{i=1}^n w_i [y_i - \bar{S}(x_i)]^2 \quad (2)$$

を最小にするような平滑化係数 c_j を求める.

ここで, 節点列 $\{\xi_j\}$ の張る区間 $I_\xi = [\min(\xi_j), \max(\xi_j)]$ は, n 個の離散点のすべてを含む必要はない. 例えば, 図 dbsc1-1. のように, I_ξ を離散点の張る区間 $I_x = [\min(x_i), \max(x_i)]$ の部分区間として指定することができる. この場合は(1)の $\bar{S}(x)$ は I_ξ に含まれる離散点(個数を n_e とする)だけを対象とする平滑化式となり, (2)の和をとる際も I_ξ に含まれる離散点だけが対象となる.

$w_i \geq 0$, $m \geq 1$, $n_t \geq 3$, $\xi_j \neq \xi_k$ ($j \neq k$), $n_e \geq n_t + m - 1$ であること.

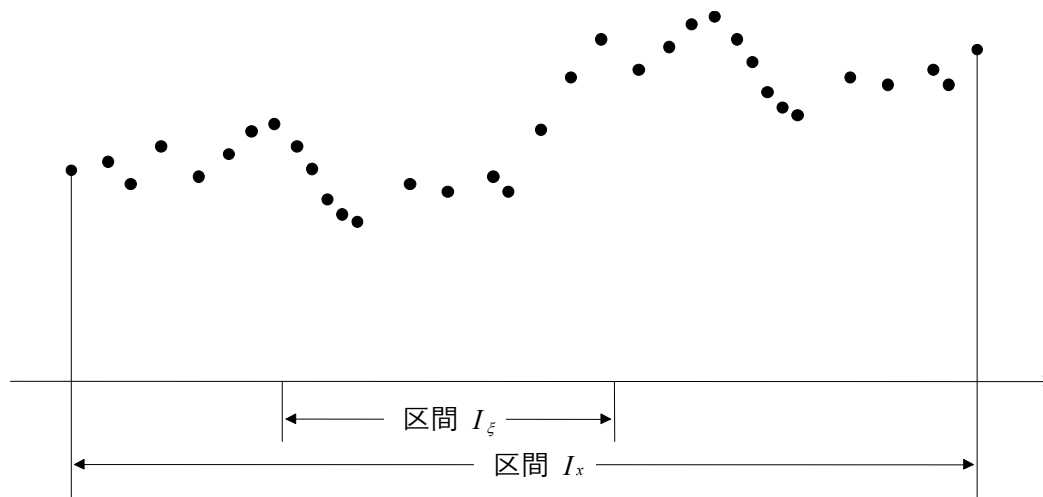


図 dbsc1-1. 平滑化式を求める区間 I_ξ の例

2. 引数

呼出し形式:

```
ierr = c_dbsc1(x, y, w, n, m, xt, nt, c, r, &rnor, vw, ivw, &icon);
```

引数の説明:

x	double x[n]	入力	離散点 x_i .
y	double y[n]	入力	観測値 y_i .
w	double w[n]	入力	重み関数値.
n	int	入力	離散点の個数 n .
m	int	入力	B-spline の次数 m . (“使用上の注意”b) 参照)
xt	double xt[nt]	入力 出力	節点 ξ_j . (“使用上の注意”c) 参照) もし入力時に $xt[0] < xt[1] < \dots < xt[nt-1]$ が満たされていなかったときは, 出力時にこのように並べ換える.
nt	int	入力	節点の個数 n_t .
c	double c[nt+m-1]	出力	平滑化係数 c_j .
r	double r[n]	出力	残差 $y_i - \bar{S}(x_i)$.
rnor	double	出力	重み付き残差の 2 乗和 δ_m^2 .
vw	double vw[Vwlen]	作業領域	$Vwlen = (nt+m)*(m+1)$.
ivw	int ivw[n]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
30000	次のいずれかであった. • w に負のものがある. • $m < 1$ • $xt[i] = xt[j]$ ($i \neq j$) • $nt < 3$ • $n_e < nt + m - 1$	処理を打ち切る.

3. 使用上の注意

a) c_dbf1 との関係

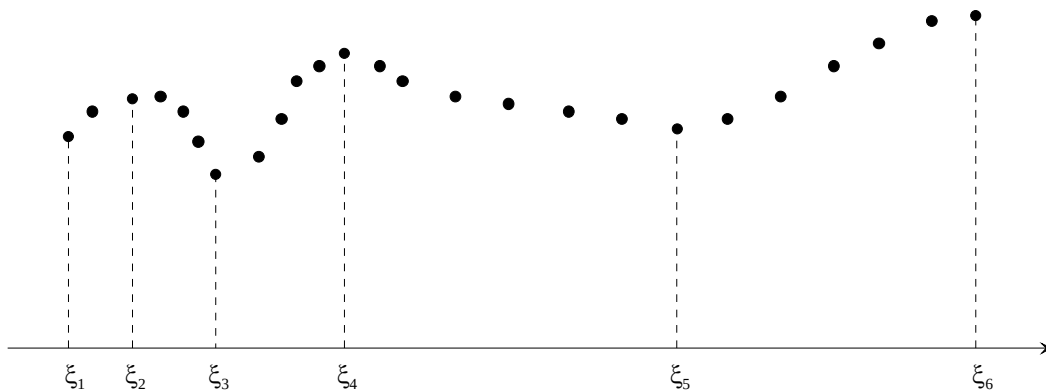
本関数に続けて, 関数 c_dbf1 を呼び出すことにより, B-spline 平滑化式(1)に基づく平滑値あるいは微分値あるいは積分値を求めることができる. そのとき, 引数 m, xt, nt 及び c の値は関数 c_dbf1 の入力となる.

b) m

次数 m は 3 程度が良く, 5 を超えるのは良くない. これは, 平滑化係数 c_j を求める際の正規方程式が, m が大きくなるにつれ, 悪条件になるからである.

c) xt

節点 ξ_j の採り方(位置決め)は観測値の挙動に応じて決めることが大切である. 一般的に言えば, 観測値がピークになる点あるいは急激に変化する点には必ず節点を選ぶべきである. 逆に滑らかに変化している領域には節点を選ぶべきではない. (図 dbsc1-2.参照)

図 dbsc1-2. 節点 ξ_j の例

4. 使用例

[0,1] 区間で等間隔な 10 個の補間点での関数 $f(x)=x^3$ の値が与えられたとき, ある点での補間値を求め結果を元の関数と比較する.

```
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define N 10
#define M 3
#define NT 5

MAIN__()
{
    int ierr, icon;
    int i, n, m, nt, isw, ivw[N];
    double x[N], y[N], w[N], c[NT+M-1], xt[NT], r[N], vw[(NT+M)*(M+1)];
    double p, h, v, f, rnor;

    /* initialize data */
    n = N;
    m = M;
    nt = NT;
    isw = 0;
    p = 0;
    h = 1.0/n;
    for (i=0;i<n;i++) {
        w[i] = 10;
        x[i] = p;
        y[i] = pow(p,3);
        p = p + h;
    }
    p = 0;
    h = 1.0/nt;
    for (i=0;i<nt;i++) {
        xt[i] = p;
        p = p + h;
    }
    /* calculate B-spline smoothing coefficients */
    ierr = c_dbsc1(x, y, w, n, m, xt, nt, c, r, &rnor, vw, ivw, &icon);
    i = nt/2;
    v = xt[i] + (xt[i+1]-xt[i])/2;
    for (isw=-1;isw<=m;isw++) {
        /* calculate value at point */
        ierr = c_dbsf1(m, xt, nt, c, isw, v, &i, &f, vw, &icon);
        if (isw == -1)
            printf("icon = %i    integral = %12.6e\n", icon, f);
        else if (isw == 0)
            printf("icon = %i    value = %12.6e\n", icon, f);
        else
            printf("icon = %i    derivative %i = %12.6e\n", icon, isw, f);
    }
}
```

```
    }  
    return(0);  
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の BSC1 の項目を参照のこと。

c_dbsc2

B-spline 平滑化式(節点追加方式)

```
ierr = c_dbsc2(x, y, s, n, m, xt, &nt, nl,
               rnot, c, rnor, vw, ivw, &icon);
```

1. 機能

離散点 $x_1, x_2, \dots, x_n$ において観測値 $y_1, y_2, \dots, y_n$, 観測誤差 $\sigma_1, \sigma_2, \dots, \sigma_n$, 残差二乗和の許容値 δ_i^2 , 初期節点列 $\xi_1, \xi_2, \dots, \xi_{n_s}$ が与えられたとき, この節点列に適当な節点を段階的に追加して観測値を最小二乗の意味で近似する場合, 残差二乗和が許容値以下となるような m 次の B-spline 平滑化式を求める. すなわち, 最終的な節点の個数を n_b それに対応する残差二乗和を $\delta_{n_b}^2$ とするとき,

$$\delta_{n_t}^2 = \sum_{i=1}^n \frac{1}{\sigma_i^2} \{y_i - \bar{S}(x_i)\}^2 \leq \delta_i^2 \quad (1)$$

が満足されるような m 次の B-spline 平滑化式.

$$\bar{S}(x) = \sum_{j=1}^{n_t-1} c_j N_{j,m+1}(x) \quad (2)$$

の平滑化係数 c_j を求める.

なお, 本関数は, 係数 c_j のほか, 最終的な節点列 $\xi_1, \xi_2, \dots, \xi_{n_t}$, 節点追加の各段階における残差二乗和 (3), 及び統計量 (4), (5) も出力する.

$$\delta_{n_r}^2 = \sum_{i=1}^n \frac{1}{\sigma_i^2} \{y_i - \bar{S}(x_i)\}^2, \quad (3)$$

(ただし $\bar{S}(x)$ は $\xi_1, \xi_2, \dots, \xi_{n_r}$ を節点列とする m 次の B-spline 平滑化式)

$$\bar{\sigma}_{n_r}^2 = \delta_{n_r}^2 / \{n - (n_r + m - 1)\}, \quad (4)$$

$$AICr = n \log \delta_{n_r}^2 + 2(n_r + m - 1) \quad (5)$$

ここで, $n_r = n_s, n_s + 1, \dots, n_t$, $\sigma_i \geq 0$, $m \geq 1$, $n_s \geq 2$ 及び初期節点列 ξ_j に対して, $\min_j(\xi_j) \leq \min_i(x_i)$, $\max_j(\xi_j) \geq \max_i(x_i)$ であること.

2. 引数

呼出し形式:

```
ierr = c_dbsc2(x, y, s, n, m, xt, &nt, nl, rnot, c, (double*)rnor, vw, ivw,
               &icon);
```

引数の説明:

x	double x[n]	入力	離散点 x_i .
y	double y[n]	入力	観測値 y_i .
s	double s[n]	入力	観測誤差 σ_i . (“使用上の注意”b)参照)
n	int	入力	離散点の個数 n .
m	int	入力	B-spline の次数 m . (“使用上の注意”c)参照)
xt	double xt[nl]	入力	初期節点 $\xi_i, i = 1, 2, \dots, n_s$. (“使用上の注意”d)参照)
		出力	求められた平滑化式の節点 $\xi_i, i = 1, 2, \dots, n_t$. 結果は $\xi_1 < \xi_2 < \dots < \xi_{n_t}$ の順に並べられる.

nt	int	入力	初期節点の個数 n_s .
		出力	求められた平滑化式の節点の個数 n_t .
nl	int	入力	節点の個数の上限 ($\geq n_s$). (“使用上の注意”e)参照)
rnot	double	入力	残差二乗和の許容値 δ_t^2 . 標準値としては $\delta_t^2 = n$ が適当.
c	double	出力	平滑化係数 $c_j, j=1-m, 2-m, \dots, n_t-1$. c_j は $c[j+m-1]$ に格納される.
rnor	double rnor[Rlen][3]	出力	節点追加の各段階における $\delta_{n_r}^2, \bar{\sigma}_{n_r}^2, AICr$ の値. $\delta_{n_r}^2$ は $rnor[n_r - n_s][0]$ に, $\bar{\sigma}_{n_r}^2$ は $rnor[n_r - n_s][1]$ に, $AICr$ は $rnor[n_r - n_s][2]$ に格納される. $Rlen = nl - n_s + 1$. (“使用上の注意”f)参照)
vw	double vw[Vwlen]	作業領域	$Vwlen = (m+1) + (m+2)(nl+m)$.
ivw	int ivw[Ivwlen]	作業領域	$Ivwlen = n + nl + m$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	節点の個数がその上限に達しても収束判定 (1) が満たされなかった.	最後に得られた平滑化式を出力する.
30000	次のいずれかであった. $\sigma_i \leq 0$ $m < 1$ $xt[i] = xt[j] \ (i \neq j)$ $n_s < 2$ $nl < n_s$ $\min_j(\xi_j) > \min_i(x_i)$ 又は $\max_j(\xi_j) < \max_i(x_i)$	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

本関数に続けて, 関数 c_dbf1 を呼び出すことにより, B-spline 平滑化式 (2) に基づく平滑値, 微分値あるいは積分値を求めることができる. そのとき, 引数 m, xt, nt, c の値は関数 c_dbf1 の入力となる.

b) s

観測誤差 σ_i は, 観測値 y_i が持つ誤差の見積りである. 例えば, y_i の値が, d_i 桁有効であるならば, σ_i は $10^{-d_i} |y_i|$ と与えればよい. σ_i は, 関数内で, $\bar{S}(x)$ をどの程度 y_i に近づけるかを示す量として使われる. σ_i が大きいと y_i へ近づける程度は小さい.

c) m

次数 m (奇数でも偶数でもよい) は 3 程度が良く, 5 を超えるのは良くない. これは, 平滑化係数 c_j を求める際の正規方程式が, m が大きくなるにつれ, 悪条件になるからである.

d) xt

初期節点列 $\xi_j, j=1, 2, \dots, n_s$ は通常, $n_s = 2, \xi_1 = \min_i(x_i), \xi_{n_s} = \max_i(x_i)$ と与えればよい.

e) nl

節点列の個数の上限 nl は, $n/2$ 程度に与えるのがよい (節点の個数が増えると, 正規方程式が悪条件となる). なお, 本関数は, 節点の個数がその上限に達しても (1) が満足されないときは, icon=10000 として処理を打ち切る.

f) rnor

rnor に出力される情報は, 節点を追加してゆく過程での各種評価基準値の履歴である. これらの履歴は, 得られた平滑化式の良悪の確認に使える. すなわち, これら評価基準値は, 通常, 節点の追加と共に減少するが, その変化は徐々に緩やかになる. 特に $\bar{\sigma}_{n_r}^2$ 及び $AICr$ が殆んど変化しなくなったときは, 平滑化式が良好になったことの現れである. したがって, 利用者は, rnor の内容を印刷することにより得られた平滑化式の良好さを確かめるのがよい. もし, 節点の個数が n_r よりも少ない段階で, 良好な平滑化式が得られていたことが分かれば, その段階での $\delta_{n_r}^2$ を新たに許容値 δ_t^2 として, 再度, 本関数を呼び出せば, その段階での平滑化式を求めることができる.

4. 使用例

区間[0,1]で等間隔な 10 個の補間点での関数 $f(x) = x^3$ の B-spline 補間式の補間係数を求め, ある点での補間値, 微分値及び積分値を求める.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define N 10
#define M 3
#define NTMAX 5
#define NT 3

MAIN__()
{
    int ierr, icon;
    int i, n, m, nt, nl, isw, ivw[N+NTMAX+M];
    double x[N], y[N], s[N], rnor[NTMAX-NT+1][3];
    double c[NTMAX+M-1], xt[NTMAX], vw[(M+1)+(M+2)*(NTMAX+M)];
    double p, h, v, f, rnot;

    /* initialize data */
    n = N;
    m = M;
    nt = NT;
    nl = NTMAX;
    rnot = n;
    p = 0.1;
    h = 0.8/(n-1);
    for (i=0; i<n; i++) {
        x[i] = p+i*h;
        y[i] = pow(x[i],3);
        s[i] = 1e-6*fabs(y[i]); /* make up some error values */
    }
    p = 0;
    h = 1.0/(nt-1);
    for (i=0; i<nt; i++) {
        xt[i] = p+i*h;
    }
    /* calculate B-spline smoothing coefficients */
    ierr = c_dbsc2(x, y, s, n, m, xt, &nt, nl, rnot,
                  c, (double*)rnor, vw, ivw, &icon);
    if (icon >= 20000) {
        printf("ERROR: c_dbsc2 failed with icon = %d\n", icon);
        exit(1);
    }
    i = 1;
    v = 0.5;
    for (isw=-1; isw<=m; isw++) {
        /* calculate value at point */
        ierr = c_dbsf1(m, xt, nt, c, isw, v, &i, &f, vw, &icon);
```

```
    if (icon >= 20000) {
        printf("ERROR: c_dbsf1 failed with icon = %d\n", icon);
        exit(1);
    }
    if (isw == -1)
        printf("icon = %i    integral = %12.6e\n", icon, f);
    else if (isw == 0)
        printf("icon = %i    value = %12.6e\n", icon, f);
    else
        printf("icon = %i    derivative %i = %12.6e\n", icon, isw, f);
    }
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の BSC2 の項目を参照のこと.

c_dbscd2

B-spline 2 次元平滑化式 (節点追加方式)

```
ierr = c_dbscd2(x, nx, y, ny, fxy, kf, sx, sy,
               m, xt, &nxt, yt, &nyt, nxl, nyl,
               rnot, c, kc, rnor, vw, ivw,
               &icon);
```

1. 機能

xy 平面上の格子点 (x_i, y_j) ; $i=1,2,\dots,n_x, j=1,2,\dots,n_y$ において, 観測値 $f_{ij} = f(x_i, y_j)$, 観測誤差 $\sigma_{ij} = \sigma_{x_i} \cdot \sigma_{y_j}$, 残差二乗和の許容値 δ_t^2 , 及び x 方向, y 方向の初期節点列 $\xi_1, \xi_2, \dots, \xi_{n_s}$; $\eta_1, \eta_2, \dots, \eta_{\ell_s}$ が与えられたとき, これら x 方向, y 方向に適当な節点を段階的に追加し観測値を最小二乗の意味で近似する場合, 残差二乗和が許容値以下となるような双 m 次の B-spline 平滑化式を求める. すなわち, x 方向, y 方向の最終的な節点の個数を n_t, ℓ_t , そのときの残差二乗和を $\delta_{n_t+\ell_t}^2$ とするとき,

$$\delta_{n_t+\ell_t}^2 = \sum_{j=1}^{n_y} \sum_{i=1}^{n_x} \frac{1}{(\sigma_{x_i} \cdot \sigma_{y_j})^2} \{f_{ij} - \bar{S}(x_i, y_j)\}^2 \leq \delta_t^2 \quad (1)$$

が満足されるような双 m 次の B-spline 平滑化式,

$$\bar{S}(x, y) = \sum_{\beta=1-m}^{\ell_t-1} \sum_{\alpha=1-m}^{n_t-1} c_{\alpha,\beta} N_{\alpha,m+1}(x) N_{\beta,m+1}(y), \quad (2)$$

の平滑化係数 $c_{\alpha,\beta}$ を求める.

なお, 本関数は, 係数 $c_{\alpha,\beta}$ のほか, x 方向の節点列 $\xi_1, \xi_2, \dots, \xi_{n_s}$, y 方向の節点列 $\eta_1, \eta_2, \dots, \eta_{\ell_s}$, 節点追加の各段階における残差二乗和 (3), 及び統計量 (4), (5) を出力する.

$$\delta_{n_r+\ell_r}^2 = \sum_{j=1}^{n_y} \sum_{i=1}^{n_x} \frac{1}{(\sigma_{x_i} \cdot \sigma_{y_j})^2} \{f_{ij} - \bar{S}(x_i, y_j)\}^2 \quad (3)$$

(ただし, $\bar{S}(x, y)$ は $\xi_1, \xi_2, \dots, \xi_{n_r}$ 及び $\eta_1, \eta_2, \dots, \eta_{\ell_r}$ をそれぞれ x, y 方向の節点列とする双 m 次の B-spline 平滑化式) .

$$\bar{\sigma}_{n_r+\ell_r}^2 = \delta_{n_r+\ell_r}^2 / \{n_x \cdot n_y - (n_r + m - 1)(\ell_r + m - 1)\}, \quad (4)$$

$$AIC_r = n_x \cdot n_y \log \delta_{n_r+\ell_r}^2 + 2(n_r + m - 1)(\ell_r + m - 1). \quad (5)$$

ここで, $n_r + \ell_r = n_s + \ell_s, n_s + \ell_s + 1, \dots, n_t + \ell_t$, $\sigma_{x_i} > 0$, $\sigma_{y_j} > 0$, $m \geq 1$, $n_s \geq 2$, $\ell_s \geq 2$, 及び, 初期節点列 $\xi_1, \xi_2, \dots, \xi_{n_s}$, $\eta_1, \eta_2, \dots, \eta_{\ell_s}$ に対して,

$$\begin{aligned} \min_j(\xi_j) &\leq \min_i(x_i), \quad \max_j(\xi_j) \geq \max_i(x_i) \\ \min_j(\eta_j) &\leq \min_i(y_i), \quad \max_j(\eta_j) \geq \max_i(y_i). \end{aligned}$$

であること.

2. 引数

呼出し形式:

```
ierr = c_dbscd2(x, nx, y, ny, (double*)fxy, kf, sx, sy, m, xt, &nxt, yt, &nyt,
               nxl, nyl, rnot, (double*)c, kc, (double*)rnot, vw, ivw, &icon);
```

引数の説明:

x	double x[nx]	入力	x 方向の離散点 x_i .
nx	int	入力	x_i の個数 n_x .
y	double y[ny]	入力	y 方向の離散点 y_j .
ny	int	入力	y_j の個数 n_y .
fxy	double fxy[nx][kf]	入力	観測値 f_{ij} .
kf	int	入力	配列 fxy の整合寸法 ($\geq ny$).
sx	double sx[nx]	入力	x 方向の観測誤差 σ_{x_i} .
sy	double sy[ny]	入力	y 方向の観測誤差 σ_{y_j} .
m	int	入力	B-spline の次数 m . (“使用上の注意”b)参照)
xt	double xt[nxl]	入力 出力	x 方向の初期節点 $\xi_i, i=1,2,\dots,n_s$. (“使用上の注意”c)参照) 求められた平滑化式の x 方向の節点列 $\xi_i, i=1,2,\dots,n_t$. 結果は $\xi_1 < \xi_2 < \dots < \xi_{n_t}$ の順に並べられる.
nxt	int	入力 出力	x 方向の初期節点の個数 $n_s (\geq 2)$. 求められた平滑化式の x 方向の節点の個数 n_t .
yt	double yt[nyl]	入力 出力	y 方向の初期節点 $\eta_j, j=1,2,\dots,\ell_s$. (“使用上の注意”c)参照) 求められた平滑化式の y 方向の節点 $\eta_j, j=1,2,\dots,\ell_t$. 結果は $\eta_1 < \eta_2 < \dots < \eta_{\ell_t}$ の順に並べられる.
nyt	int	入力 出力	y 方向の初期節点の個数 $\ell_s (\geq 2)$. 求められた平滑化式の y 方向の節点の個数 ℓ_t .
nxl	int	入力	x 方向の節点の個数の上限 ($\geq n_s$). (“使用上の注意”d)参照)
nyl	int	入力	y 方向の節点の個数の上限 ($\geq \ell_s$). (“使用上の注意”d)参照)
rnot	double	入力	残差二乗和の許容値 δ_t^2 . 標準値としては $\delta_t^2 = n_x \cdot n_y$ が適当.
c	double c[Clen][kc]	出力	平滑化係数 $c_{\alpha,\beta}$. $c_{\alpha,\beta} = c[\alpha + m - 1][\beta + m - 1]$, $\alpha = 1 - m, 2 - m, \dots, n_t - 1$, $\beta = 1 - m, 2 - m, \dots, \ell_t - 1$ のように格納される. $Clen = nxl + m - 1$.
kc	int	入力	配列 c の整合寸法 ($\geq nyl + m - 1$).
rnor	double rnor[Rlen][3]	出力	節点追加の各段階における (3), (4), (5) の値. $n_r + \ell_r = n_s + \ell_s, n_s + \ell_s + 1, \dots, n_t + \ell_t$, とするとき, $\delta_{n_r + \ell_r}^2$ は $rnor[P_r][0]$ に, $\bar{\sigma}_{n_r + \ell_r}^2$ は $rnor[P_r][1]$ に, AIC_r は $rnor[P_r][2]$ に格納される. ここで $P_r = (n_r - n_s) + (\ell_r - \ell_s)$. $Rlen = (nxl - n_s) + (nyl - \ell_s) + 1$. (“使用上の注意”e)参照)
vw	double vw[Vwlen]	作業領域	$Vwlen = \max(s_1, s_2)$. $s_1 = (n_x + n_y + 2m)(m + 1)$ $\quad + \max\{\max(n_x + m, n_y + m), 2 + \min(n_x + m, n_y + m)(m + 1)\}$ $s_2 = \{\min(n_x, n_y) + 3\}(m + 1) + n_x + n_y + nxl + nyl + 2$
ivw	int ivw[Ivwlen]	作業領域	$Ivwlen = n_x + n_y + \max(nxl, nyl) \cdot m$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	x 方向の節点の個数が, その上限に達しても収束判定 (1) が満たされなかった.	最後に得られた平滑化式を出力する.
11000	y 方向の節点の個数が, その上限に達しても収束判定 (1) が満たされなかった.	最後に得られた平滑化式を出力する.
30000	次のいずれかであった: $\sigma_{x_i} \leq 0$ $\sigma_{y_j} \leq 0$ $m < 1$ $xt[i] = xt[j]$ 又は $yt[i] = yt[j]$, ($i \neq j$) $n_s < 2$ 又は $\ell_s < 2$ $nxl < n_s$ 又は $nyl < \ell_s$ $\min_i(\xi_i) > \min_i(x_i)$ 又は $\max_i(\xi_i) < \max_i(x_i)$ $\min_j(\eta_j) > \min_j(y_j)$ 又は $\max_j(\eta_j) < \max_j(y_j)$	処理を打ち切る.

3. 使用上の注意

a) c_dbstd1 との関係

本関数に続けて, 関数 c_dbstd1 を呼び出すことにより, B-spline 平滑化式 (2) に基づく平滑値, 偏微分値あるいは二重積分値を求めることができる. そのとき, 引数 m, xt, nxt, yt, nyt, c の値は関数 c_dbstd1 の入力となる.

b) m

次数 m (奇数でも偶数でもよい) は 3 程度が良く, 5 を超えるのは良くない. これは, 平滑化係数 $c_{\alpha, \beta}$ を求める際の正規方程式が m が大きくなるにつれ, 悪条件になるからである.

c) xt 及び yt

x 方向, y 方向の初期節点列 ξ_i, η_j ; $i = 1, 2, \dots, n_s$, $j = 1, 2, \dots, \ell_s$ は, 通常,

$$n_s = \ell_s = 2, \quad \xi_1 = \min_i(x_i), \quad \xi_{n_s} = \max_i(x_i), \quad \eta_1 = \min_j(y_j), \quad \eta_{\ell_s} = \max_j(y_j)$$

と与えればよい.

d) nxl 及び nyl

x 方向, y 方向の節点の個数の上限 nxl, nyl はそれぞれ, $n_x / 2, n_y / 2$ 程度に与えるのがよい (節点の個数が増えると, 正規方程式が悪条件となる). なお, 本関数は, 節点の個数がその上限に達しても (1) が満たされないときは, x 方向, y 方向に応じて $icon = 10000$ 又は 11000 として処理を打ち切る.

e) rnor

rnor に出力される情報は, 節点を追加してゆく過程での各種評価基準値の履歴である. これらの履歴は, 得られた平滑化式の良悪の確認に使える. すなわち, これら評価基準値は, 通常, 節点の追加と共に減少するが, その変化は徐々に緩やかになる. 特に $\sigma_{n_r + \ell_r}^2$ 及び AIC_r が殆んど変化しなくなったときは, 平滑化式が良好になったことの現れである. したがって, 利用者は, rnor の内容を印刷することにより, 得られた平滑化式の良好さを確かめるのがよい. もし節点の個数が, x 方向, y 方向合わせて $n_t + \ell_r$ よりも少ない段階で, 良好な

平滑化式が得られていたことが分かれば、その段階での $\delta_{n_r+\ell_r}^2$ を新たに許容値 δ_t^2 として、再度、本関数を呼び出せば、その段階での平滑化式を求めることができる。

4. 使用例

区間 $[0.1, 0.9] \times [0.1, 0.9]$ で 100 個の補間点での関数 $f(x, y) = x^3 y^3$ の B-spline 補間式の補間係数を求め、ある点での補間値、微分値及び積分値を求める。

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define N 10
#define M 3
#define NTMAX 5
#define NT 3

MAIN__()
{
    int ierr, icon;
    int i, j, kf, kc, nx, ny, m, nxt, nyt, nxl, nyl, ivw[2*N+NTMAX*M];
    int iswx, iswy, ix, iy;
    double x[N], y[N], fxy[N][N], sx[N], sy[N], rnor[2*(NTMAX-NT)+1][3];
    double c[NTMAX+M-1][NTMAX+M-1], xt[NTMAX], yt[NTMAX];
    double vw[158];
    double p, h, vx, vy, f, fx, rnot;

    /* initialize data */
    nx = N;
    ny = N;
    m = M;
    nxt = NT;
    nyt = NT;
    nxl = NTMAX;
    nyl = NTMAX;
    rnot = nx*ny;
    kf = N;
    kc = NTMAX+M-1;
    p = 0.1;
    h = 0.8/(nx-1);
    for (i=0; i<nx; i++) {
        x[i] = p+i*h;
        y[i] = x[i];
        sx[i] = 1e-6; /* make up some error values */
        sy[i] = sx[i];
    }
    for (i=0; i<nx; i++) {
        fx = x[i]*x[i]*x[i];
        for (j=0; j<ny; j++) {
            fxy[i][j] = fx*y[j]*y[j]*y[j];
        }
    }
    p = 0;
    h = 1.0/(nxt-1);
    for (i=0; i<nxt; i++) {
        xt[i] = p+i*h;
        yt[i] = xt[i];
    }
    /* calculate B-spline smoothing coefficients */
    ierr = c_dbscd2(x, nx, y, ny, (double*)fxy, kf, sx, sy, m,
                    xt, &nxt, yt, &nyt, nxl, nyl, rnot,
                    (double*)c, kc, (double*)rnor, vw, ivw, &icon);
    if (icon >= 20000) {
        printf("ERROR: c_dbscd2 failed with icon = %d\n", icon);
        exit(1);
    }
    ix = 1;
    iy = ix;
    vx = 0.5;
    vy = vx;
    for (iswx=-1; iswx<=m; iswx++) {
        iswy = iswx;
```



```
/* calculate value at point */
ierr = c_dbstd1(m, xt, nxt, yt, nyt, (double*)c, kc,
               iswx, vx, &ix, iswy, vy, &iy, &f, vw, &icon);
if (icon >= 20000) {
    printf("ERROR: c_dbstd1 failed with icon = %d\n", icon);
    exit(1);
}
if (iswx == -1)
    printf("icon = %i    integral = %12.6e\n", icon, f);
else if (iswx == 0)
    printf("icon = %i    value = %12.6e\n", icon, f);
else
    printf("icon = %i    derivative %i = %12.6e\n", icon, iswx, f);
}
return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の BSCD2 の項目を参照のこと。

c_dbst1

実対称 3 重対角行列の固有値(バイセクション法)

```
ierr = c_dbst1 (d, sd, n, m, epst, e, vw,
               &icon);
```

1. 機能

n 次の実対称 3 重対角行列 $\mathbf{T}$ の固有値をバイセクション法により, 大きい方から, 若しくは小さい方から m 個求める. ただし $1 \leq m \leq n$ であること.

2. 引数

呼出し形式:

```
ierr = c_dbst1 (d, sd, n, m, epst, e, vw, &icon);
```

引数の説明:

d	double d[n]	入力	3 重対角行列 $\mathbf{T}$ の主対角要素.
sd	double sd[n]	入力	3 重対角行列 $\mathbf{T}$ の副対角要素. sd[i-1], i=2,...,n に格納する. sd[0]=0.
n	int	入力	3 重対角行列 $\mathbf{T}$ の次数 n .
m	int	入力	求めたい固有値の個数($m \neq 0$). $m=+m$ のときは大きい方から求める. $m=-m$ のときは小さい方から求める.
epst	double	入力	固有値の収束判定に使われる絶対誤差の上限. ただし, 負の値を入れると標準値が設定される. ("使用上の注意"b)参照)
e	double e[m]	出力	m 個の固有値. $m > 0$ の場合大きい順に, $m < 0$ の場合小さい順に並ぶ.
vw	double vw[n+2m]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	$n = 1$	$e[0] = d[0]$.
30000	次のいずれかであった. $n < m$ $m = 0$	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

$n/4$ 個以上の固有値を求める場合は, 関数 c_dtrql を用いた方が, 一般的に処理速度が速い.

通常, 実対称行列の固有値を求める場合は, 関数 c_dtrid1 により 3 重対角化し, 本関数又は関数 c_dtrql で固有値を求めること.

b) epst

零を固有値として持つことが考えられる場合の epst の与え方については“富士通 SSL II 使用手引書”の BSCT1 の項目を参照のこと。

4. 使用例

実対称バンド行列を、関数 c_dbtrid によって 3 重対角行列へ変換した後、 m 個の固有値を計算する。

```
#include <stdio.h>
#include <stdlib.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 15
#define NHMAX 2

MAIN__()
{
    int ierr, icon;
    int n, nh, m, i, k, ij;
    double a[NMAX*(NHMAX+1)-NHMAX*(NHMAX+1)/2], e[NMAX];
    double sd[NMAX], d[NMAX], vw[NMAX+2*NMAX], epst;

    /* initialize matrix */
    n = NMAX;
    k = NMAX;
    nh = NHMAX;
    a[0] = 10;
    a[1] = -3;
    a[2] = 10;
    ij = (nh+1)*nh/2;
    for (i=0; i<n-nh; i++) {
        a[ij] = -6;
        a[ij+1] = -3;
        a[ij+2] = 10;
        ij = ij+nh+1;
    }
    /* reduce to tridiagonal form */
    ierr = c_dbtrid(a, n, nh, d, sd, &icon);
    if (icon > 10000) {
        printf("ERROR: c_dbtrid failed with icon = %i\n", icon);
        exit (1);
    }
    /* find eigenvalues using c_dbst1 */
    m = n;
    epst = 1e-6;
    ierr = c_dbst1(d, sd, n, m, epst, e, vw, &icon);
    if (icon > 10000) {
        printf("ERROR: c_dbst1 failed with icon = %i\n", icon);
        exit (1);
    }
    printf("icon = %i\n", icon);
    /* print eigenvalues */
    printf("eigenvalues:\n");
    for (i=0; i<m; i++) {
        printf("%7.4f  ", e[i]);
    }
    printf("\n");
    /* find eigenvalues using c_dtrql */
    ierr = c_dtrql(d, sd, n, e, &m, &icon);
    if (icon >= 20000) {
        printf("ERROR: c_dtrql failed with icon = %i\n", icon);
        exit (1);
    }
    printf("icon = %i\n", icon);
    /* print eigenvalues */
    printf("eigenvalues:\n");
    for (i=0; i<m; i++) {
        printf("%7.4f  ", e[i]);
    }
    printf("\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の BSCT1 の項目及び[80], [118], [119]を参照のこと.

c_dbseg

実対称バンド行列の固有値及び固有ベクトル
(ルティスハウザー・シュワルツ法, バイセクション法, 逆反復法)

```
ierr = c_dbseg(a, n, nh, m, nv, epst, e, ev,
               k, vw, &icon);
```

1. 機能

次数 n , バンド幅 h の実対称バンド行列 $\mathbf{A}$ の, 大きい方から又は小さい方からの m 個の固有値を, ルティスハウザー・シュワルツ法及びバイセクション法により求め, これに対応する n_v 個の固有ベクトルを逆反復法により求める. 固有ベクトルは, $\|x\|_2 = 1$ となるように正規化される. $1 \leq m \leq n$, $0 \leq n_v \leq m$, $0 \leq h \ll n$ であること.

2. 引数

呼出し形式:

```
ierr = c_dbseg(a, n, nh, m, nv, epst, e, (double *)ev, k, vw, &icon);
```

引数の説明:

a	double a[Alen]	入力	実対称バンド行列 $\mathbf{A}$. 対称バンド行列用圧縮格納法. 格納法の詳細については“1.3.4 バンド行列の格納方法”を参照のこと. $Alen = n(h+1) - h(h+1)/2$.
		出力	$n_v \neq 0$ のとき, 演算後内容は保存される. $n_v = 0$ のとき, 演算後内容は保存されない.
n	int	入力	行列 $\mathbf{A}$ の次数 n .
nh	int	入力	バンド幅 h .
m	int	入力	求める固有値の個数 m . $m = +m$ のときは大きい方から求める. $m = -m$ のときは小さい方から求める.
nv	int	入力	求める固有ベクトルの個数 n_v . $nv = -n_v$ のときは $nv = +n_v$ とする. $n_v = 0$ のときは固有ベクトルを計算しない.
epst	double	入力	固有値の収束判定に使われる絶対誤差の上限. ただし, 負の値を入れると標準値が設定される.
e	double e[m]	出力	固有値.
ev	double ev[nv][k]	出力	固有ベクトル. 固有ベクトルは行方向に格納される.
k	int	入力	配列 ev の整合寸法. $n_v = 0$ のときは任意の変数でよい.
vw	double vw[Vwlen]	作業領域	$Vwlen = \max(3n + 2m, 2n(h+1))$. ただし, $n_v = 0$ のときは, $Vwlen = 3n + 2m$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	nh=0であった.	正常に実行される.
15000	固有ベクトルのすべてを求めることはできなかった.	求められなかった固有ベクトルを0ベクトルとする.
20000	固有ベクトルが一つも求められなかった.	すべての固有ベクトルを0ベクトルとする.
30000	次のいずれかであった. • nh < 0 • nh ≥ n • k < n • m = 0 • m < nv • m > n	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

本関数は実対称バンド行列用である. 行列の次数 n とバンド幅 h の比 h/n が, 1/6 以下程度の行列の固有値を, 大きい方からあるいは小さい方から少数個求める場合に適する.

b) 固有ベクトル

求めた固有値に対応する固有ベクトルも同時に計算できるが, 本関数で用いる逆反復法は, 実対称3重対角行列ではなく, 入力されたバンド行列に対して直接処理せざるをえないために比較的非効率である. したがって, 不要な固有ベクトルはなるべく計算しない方がよい.

4. 使用例

5×5の対称バンド行列の固有値・固有ベクトルを求める.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 15
#define HMAX 2

MAIN__()
{
    int ierr, icon;
    int n, nh, m, nv, i, j, k, ij;
    double a[NMAX*(HMAX+1)-HMAX*(HMAX+1)/2], e[NMAX], ev[NMAX][NMAX];
    double vw[2*NMAX*(HMAX+1)], epst;

    /* initialize matrix */
    n = NMAX;
    k = NMAX;
    nh = HMAX;
    a[0] = 10;
    a[1] = -3;
    a[2] = 10;
    ij = (nh+1)*nh/2;
    for (i=0; i<n-nh; i++) {
        a[ij] = -6;
        a[ij+1] = -3;
        a[ij+2] = 10;
        ij = ij+nh+1;
    }
    m = 1;
    nv = m;
```

```
    epst = -1;
    /* find eigenvalues and eigenvectors */
    ierr = c_dbseg(a, n, nh, m, nv, epst, e, (double*)ev, k, vw, &icon);
    printf("icon = %i\n", icon);
    /* print eigenvalues and eigenvectors */
    for (i=0;i<m;i++) {
        printf("eigenvalue: %7.4f\n", e[i]);
        printf("eigenvector: ");
        for (j=0;j<n;j++)
            printf("%7.4f ", ev[i][j]);
        printf("\n");
    }
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の BSEG の項目及び[118], [119]を参照のこと.

c_dbsegi

実対称バンド行列の固有値及び固有ベクトル (ジェニングス法)

<pre>ierr = c_dbsegi(a, n, nh, m, epst, lm, e, ev, k, &it, vw, &icon);</pre>

1. 機能

次数 n , バンド幅 h の実対称バンド行列 $\mathbf{A}$ の固有値を絶対値の大きい方 (又は小さい方) から m 個, 及びそれらに対応する固有ベクトルを, m 個の与えられた初期ベクトルをもとにして, ジェニングスの加速を伴うジェニングスの同時反復法により求める. ただし, 絶対値の小さい方から求めるときは, $\mathbf{A}$ は正値行列でなければならない. なお固有ベクトルは $\|\mathbf{x}\|_2 = 1$ のように正規化されている. $1 \leq m \ll n$, $0 \leq h \ll n$ であること.

2. 引数

呼出し形式:

```
ierr = c_dbsegi(a, n, nh, m, epst, lm, e, (double *)ev, k, &it, vw, &icon);
```

引数の説明:

a	double a[Alen]	入力	実対称バンド行列 $\mathbf{A}$. 対称バンド行列用圧縮格納法. 格納法の詳細については“1.3.4 バンド行列の格納方法”を参照のこと. $Alen = n(h+1) - h(h+1)/2$.
		出力	絶対値の小さい方から求めるときには, 演算後, 内容は保存されない.
n	int	入力	行列 $\mathbf{A}$ の次数 n .
nh	int	入力	行列 $\mathbf{A}$ のバンド幅 h .
m	int	入力	求める固有値及び固有ベクトルの個数 m . $m = +m$ のときは絶対値の大きい方から求める.. $m = -m$ のときは絶対値の小さい方から求める. (“使用上の注意”a)参照)
epst	double	入力	固有ベクトルの収束判定に用いられる定数 ε . 零又は負のときは標準値が設定される. (“使用上の注意”b)参照)
lm	int	入力	反復回数の上限. 反復回数がこれを超えると処理を中断する. (“使用上の注意”c)参照)
e	double e[m]	出力	固有値. 引数 m で指定された順序に格納される.
ev	double ev[m+2][k]	入力	行方向に格納された m 個の初期ベクトル. (“使用上の注意”d)参照)
		出力	固有ベクトル. 行方向に格納される.
k	int	入力	ev の整合寸法 ($\geq n$).
it	int	出力	固有値固有ベクトルが求まるまでの反復回数.
vw	double vw[Vwlen]	作業領域	$Vwlen = \max(n, 2m) + m(3m+1)/2$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	反復回数が上限 lm を超えた.	処理を終了する. e と ev にはそのときまでに得られた固有値と固有ベクトルの近似値が入っている.
28000	反復ごとの固有ベクトルの直交化処理が不可能となった.	処理を打ち切る.
29000	行列 A が正値行列でない. 又は非正則の可能性が強い.	処理を打ち切る.
30000	次のいずれかであった. $nh < 0$ 又は $nh \geq n$ $k < n$ $m = 0$ 又は $m > n$	処理を打ち切る.

3. 使用上の注意

a) m

固有値及び固有ベクトルの個数 m は, n に比べて小さく, 例えば $m/n < 1/10$ 程度にとるのがよい. 固有値を絶対値の大きい方から, あるいは小さい方から番号をつけて, $\lambda_1, \lambda_2, \dots, \lambda_m$ とするとき, もし可能ならば $|\lambda_{m+1} / \lambda_m| < 1$ 又は $|\lambda_{m+1} / \lambda_m| \gg 1$ となるように m を選ぶのが, 収束を速める上で望ましい.

b) $epst$

$epst$ は $\|x\|_2 = 1$ の意味で正規化された固有ベクトルの各成分の収束を判定するために用いられる. 固有ベクトルが判定定数 ε に対して収束したとき, 固有値は少なくとも $\|A\|_2 \varepsilon$ の精度で, そして多くの場合 $\|A\|_2 \varepsilon$ よりも高い精度で収束している. このことを考慮して幾分大きめに $epst$ の値を選ぶのがよい. 丸め誤差の単位を u とするとき, 標準値は $\varepsilon = 16\mu$ としている. しかし, 固有値が密集した問題では, 収束が得られないことがあり, この場合は $\varepsilon \geq 100\mu$ にとるのが安全である.

c) lm

反復回数の上限 lm は, 何らかの理由で収束が達成されないとき, 強制的に反復を中断するためのものである. 要求精度や固有値の密集の度合などを考慮して定められるべきものであるが, 500~1000 あたりが妥当な標準値である.

d) 初期固有ベクトル

初期固有ベクトルは, 求めようとする固有値に対応する固有ベクトルの良い近似であることが望ましい. しかしそのような近似ベクトルがわからないときには, 単位行列 I の初めの m 個の列ベクトルを初期ベクトルとして採用するのが標準的である.

e) 本関数の使用方法

C-SSL II には, 実対称バンド行列の固有値と固有ベクトルを直接法によって求めるための関数 `c_dbseg` が用意されている. 同じ問題を解く場合には, 概して直接法の方が速いが, 固有ベクトルの計算に余分の記憶容量を必要とする難点がある. 問題の規模, 要求精度, 記憶容量, 計算機の手数などを考慮して適当な関数を採用すべきである.

4. 使用例

実対称バンド行列 A の固有値及び固有ベクトルを求める.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */
```

```
#define NMAX 15
#define NHMAX 2

MAIN__()
{
    int ierr, icon;
    int n, nh, m, i, j, k, ij, it, lm;
    double a[NMAX*(NHMAX+1)-NHMAX*(NHMAX+1)/2], e[NMAX], ev[NMAX+2][NMAX];
    double vw[2*NMAX+NMAX*(3*NMAX+1)/2], epst;

    /* initialize matrix */
    n = NMAX;
    k = NMAX;
    nh = NHMAX;
    a[0] = 10;
    a[1] = -3;
    a[2] = 10;
    ij = (nh+1)*nh/2;
    for (i=0; i<n-nh; i++) {
        a[ij] = -6;
        a[ij+1] = -3;
        a[ij+2] = 10;
        ij = ij+nh+1;
    }
    m = 1;
    /* initialize m eigenvectors */
    for (i=0; i<m; i++)
        for (j=0; j<n; j++)
            if (i == j) ev[i][j] = 1;
            else ev[i][j] = 0;
    lm = 1000;
    epst = 1e-6;
    /* find eigenvalues and eigenvectors */
    ierr = c_dbsegj(a, n, nh, m, epst, lm, e, (double*)ev, k, &it, vw, &icon);
    printf("icon = %i\n", icon);
    /* print eigenvalues and eigenvectors */
    for (i=0; i<m; i++) {
        printf("eigenvalue: %7.4f\n", e[i]);
        printf("eigenvector: ");
        for (j=0; j<n; j++)
            printf("%7.4f ", ev[i][j]);
        printf("\n");
    }
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の BSEGJ の項目及び[61], [128]を参照のこと.

c_dbssf1

B-spline 平滑化式による 平滑化, 数値微分, 数値積分
ierr = c_dbssf1(m, xt, nt, c, isw, v, &i, &f, vw, &icon);

1. 機能

離散点 $x_1, x_2, \dots, x_n$ に対して観測値 $y_1, y_2, \dots, y_n$, 重み関数値 $w_i = w(x_i)$, $i=1, 2, \dots, n$ 及び Spline 関数の節点 $\xi_1, \xi_2, \dots, \xi_{n_t}$ ($\xi_1 < \xi_2 < \dots < \xi_{n_t}$) が与えられたとき, B-spline 平滑化式に基づいて $x = v \in [\xi_1, \xi_{n_t}]$ における平滑値あるいは微分値, あるいは ξ_1 から v までの積分値を求める.

ただし, 本関数を利用する前に, 関数 c_dbssf1 により, B-spline 平滑化式,

$$\bar{S}(x) = \sum_{j=1-m}^{n_t-1} c_j N_{j,m+1}(x) \quad (1)$$

における平滑化係数 c_j , $j=1-m, 2-m, \dots, n_t-1$ が計算されているとする.

ここで m は B-spline $N_{j,m+1}(x)$ の次数である. $m \geq 1$, $n_t \geq 3$, $\xi_1 \leq v \leq \xi_{n_t}$ であること.

2. 引数

呼出し形式:

```
ierr = c_dbssf1(m, xt, nt, c, isw, v, &i, &f, vw, &icon);
```

引数の説明:

m	int	入力	B-spline の次数 m .
xt	double xt[nt]	入力	節点 ξ_i .
nt	int	入力	節点の個数 n_t .
c	double c[nt+m-1]	入力	平滑化係数 c_j (c_dbssf1 の出力).
isw	int	入力	計算の種類を与える. 0 平滑値 $F = \bar{S}(v)$ を求める. l l 階微分値 $F = \bar{S}^{(l)}(v)$ を求める ($1 \leq l \leq m$). -1 積分値 $F = \int_{\xi_1}^v \bar{S}(x) dx$ を求める.
v	double	入力	平滑値などを求めたい点 v .
i	int	入力	$\xi_i \leq v < \xi_{i+1}$ のとき i には $i-1$ を入力する. i は $xt[i] \leq v < xt[i+1]$ を満たす. $v = \xi_{n_t}$ のときは $i = n_t - 2$ とする.
		出力	$\xi_i \leq v < \xi_{i+1}$ を満たす i . ("使用上の注意"b) 参照)
f	double	出力	平滑値あるいは l 階微分値あるいは積分値 ("isw" の項参照)
vw	double vw[m+1]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	$xt[i] \leq v < xt[i+1]$ が満たされていない.	本関数の中で左記の i をさがして処理を続ける.
30000	次のいずれかであった. $v < xt[0]$ 又は $v > xt[nt-1]$ $isw < -1$ 又は $isw > m$	処理を打ち切る.

3. 使用上の注意

a) c_dbf1 との関係

本関数は、関数 c_dbf1 により求められた B-spline 平滑化式(1)に基づいて平滑値あるいは微分値あるいは積分値を求めるものである。したがって、本関数を呼び出す前に、c_dbf1 を呼び出して平滑化式(1)を求め、続けて本関数を呼び出すことにより平滑値などを求めることができる。そのとき、引数 m , xt , nt 及び c は c_dbf1 のそれらと同一でなければならない。

b) i

引数 i は $xt[i] \leq v < xt[i+1]$ を満たしていることが好ましい。もし満足されていないときは、この条件式を満足するような i を探索して処理を続ける。

ここで、数学で一般に使用する添え字と C 言語の配列の添え字では、値が 1 ずれることに注意する必要がある。すなわち、C 言語の添え字は 0 から始まり、数学の添え字は 1 から始まるため i には $i-1$ を設定する。

4. 使用例

[0,1] 区間で等間隔な 10 個の補間点での関数 $f(x) = x^3$ の値が与えられたとき、ある点での補間値を求め結果を元の関数と比較する。

```
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define N 10
#define M 3
#define NT 5

MAIN__()
{
    int ierr, icon;
    int i, n, m, nt, isw, ivw[N];
    double x[N], y[N], w[N], c[NT+M-1], xt[NT], r[N], vw[(NT+M)*(M+1)];
    double p, h, v, f, rnor;

    /* initialize data */
    n = N;
    m = M;
    nt = NT;
    isw = 0;
    p = 0;
    h = 1.0/n;
    for (i=0; i<n; i++) {
        w[i] = 10;
        x[i] = p;
        y[i] = pow(p,3);
        p = p + h;
    }
    p = 0;
    h = 1.0/nt;
```

```
for (i=0;i<nt;i++) {
    xt[i] = p;
    p = p + h;
}
/* calculate B-spline smoothing coefficients */
ierr = c_dbf1(x, y, w, n, m, xt, nt, c, r, &rnor, vw, ivw, &icon);
i = nt/2;
v = xt[i] + (xt[i+1]-xt[i])/2;
for (isw=-1;isw<=m;isw++) {
    /* calculate value at point */
    ierr = c_dbsf1(m, xt, nt, c, isw, v, &i, &f, vw, &icon);
    if (isw == -1)
        printf("icon = %i    integral = %12.6e\n", icon, f);
    else if (isw == 0)
        printf("icon = %i    value = %12.6e\n", icon, f);
    else
        printf("icon = %i    derivative %i = %12.6e\n", icon, isw, f);
}
return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の BSF1 の項目を参照のこと。

c_dbsfd1

B-spline 2 次元平滑化式による平滑化, 数値微分, 数値積分

```
ierr = c_dbsfd1(m, xt, nxt, yt, nyt, c, kc,
               iswx, vx, &ix, iswy, vy, &iy, &f,
               vw, &icon);
```

1. 機能

x y 平面上の格子点 (x_i, y_j) ; $i=1,2,\dots,n_x$, $j=1,2,\dots,n_y$ において, 観測値 $f_{ij} = f(x_i, y_j)$, 観測誤差 $\sigma_{ij} = \sigma_{x_i} \cdot \sigma_{y_j}$ が与えられたとき, 双 m 次の B-Spline 平滑化式に基づいて点 $P(v_x, v_y)$ における平滑値, 偏微分値, あるいは領域 $[\xi_1 \leq x \leq v_x, \eta_1 \leq y \leq v_y]$ 上の二重積分値を求める.

ただし, 本関数を利用する前に, 関数 c_dbstd2 により, x 方向の節点列 $\xi_1, \xi_2, \dots, \xi_{n_t}$, y 方向の節点列 $\eta_1, \eta_2, \dots, \eta_{\ell_t}$, 更に, 双 m 次の B-spline 平滑化式

$$\bar{S}(x, y) = \sum_{\beta=1-m}^{\ell_t-1} \sum_{\alpha=1-m}^{n_t-1} c_{\alpha,\beta} N_{\alpha,m+1}(x) N_{\beta,m+1}(y). \quad (1)$$

における平滑化係数 $c_{\alpha,\beta}$ が計算されているとする. ここで, $m \geq 1$, $\xi_1 \leq v_x \leq \xi_{n_t}$, $\eta_1 \leq v_y \leq \eta_{\ell_t}$ であること.

2. 引数

呼出し形式:

```
ierr = c_dbsfd1(m, xt, nxt, yt, nyt, (double*)c, kc, iswx, vx, &ix, iswy, vy,
               &iy, &f, vw, &icon);
```

引数の説明:

m	int	入力	B-spline の次数 m .
xt	double xt[nxt]	入力	x 方向の節点 ξ_i .
nxt	int	入力	x 方向の節点 ξ_i の個数 n_t .
yt	double yt[nyt]	入力	y 方向の節点 η_j .
nyt	int	入力	y 方向の節点 η_j の個数 ℓ_t .
c	double c[nxt+m-1][kc]	入力	平滑化係数 $c_{\alpha,\beta}$.
kc	int	入力	配列 c の整合寸法 ($\geq \text{nyt} + m - 1$).
iswx	int	入力	x 方向に関する計算の種類を与える. $-1 \leq \text{iswx} \leq m$. (引数 “f” の項参照)
vx	double	入力	点 $P(v_x, v_y)$ の x 座標.
ix	int	入力	$\xi_i \leq v_x < \xi_{i+1}$ のとき ix には $i-1$ を入力する. ix は $\text{xt}[\text{ix}] \leq v_x < \text{xt}[\text{ix}+1]$ を満たす. $v_x = \xi_{n_t}$ のときは $\text{ix} = n_t - 2$ とする.
		出力	$\text{xt}[\text{ix}] \leq v_x < \text{xt}[\text{ix}+1]$ を満たす ix. (“使用上の注意”b) 参照)
iswy	int	入力	x 方向に関する計算の種類を与える. $-1 \leq \text{iswy} \leq m$. (引数 “f” の項参照)
vy	double	入力	点 $P(v_x, v_y)$ の y 座標.
iy	int	入力	$\eta_i \leq v_y < \eta_{i+1}$ のとき iy には $i-1$ を入力する. iy は $\text{yt}[\text{iy}] \leq v_y < \text{yt}[\text{iy}+1]$ を満たす. $v_y = \eta_{\ell_t}$ のときは $\text{iy} = \ell_t - 2$ とする.

		出力	yt[iy] ≤ vy < yt[iy+1]を満たす iy. (“使用上の注意”b)参照)
f	double	出力	平滑値あるいは偏微分値あるいは、積分値. いま, iswx = λ, iswy = μ, とすると, λ, μ の組合せごとに次の値を出力する. - λ, μ ≥ 0 のとき $f = \frac{\partial^{\lambda+\mu}}{\partial x^\lambda \partial y^\mu} \bar{S}(v_x, v_y)$ 平滑値は λ = μ = 0 とすることにより得られる. - λ = -1, μ ≥ 0 のとき $f = \int_{\xi_1}^{v_x} \frac{\partial^\mu}{\partial y^\mu} \bar{S}(x, v_y) dx$ - λ ≥ 0, μ = -1 のとき $f = \int_{\eta_1}^{v_y} \frac{\partial^\lambda}{\partial x^\lambda} \bar{S}(v_x, y) dy$ - λ = μ = -1 のとき $f = \int_{\eta_1}^{v_y} \int_{\xi_1}^{v_x} \bar{S}(x, y) dx dy$
vw	double	vw[Vwlen]	作業領域 Vwlen = 5(m+1) + max(n _t , ℓ _t).
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	xt[ix] ≤ vx < xt[ix+1] 又は yt[iy] ≤ vy < yt[iy+1] が満たされていない.	本関数内で左記の ix 又は iy をさがして処理を続ける.
30000	次のいずれかであった: - vx < xt[0] 又は vx > xt[nxt-1] - vy < yt[0] 又は vy > yt[nyt-1] - iswx < -1 又は iswx > m - iswy < -1 又は iswy > m	処理を打ち切る.

3. 使用上の注意

a) c_dbstd2 との関係

本関数は、関数 c_dbstd2 により求められた B-spline2 次元平滑化式 (1) に基づいて、平滑値、偏微分値、あるいは二重積分値を求めるものである。したがって、本関数を呼び出す前に、c_dbstd2 を呼び出して平滑化式 (1) を求め、続けて本関数を呼び出すことにより平滑値などを求めることができる。そのとき、引数 m, xt, nxt, yt, nyt, c, kc の値は c_dbstd2 のそれらと同一でなければならない。

b) ix 及び iy

引数 ix 及び iy はそれぞれ $x[ix] \leq vx < x[ix+1]$, $y[iy] \leq vy < y[iy+1]$ を満足していることが好ましい。もし満足されていないときは、これらを満足するような ix, iy を探索して処理を続ける。

ここで、数学で一般に使用する添え字と C 言語の配列の添え字では、値が 1 ずれることに注意する必要がある。すなわち、C 言語の添え字は 0 から始まり、数学の添え字は 1 から始まるため ix, iy は i-1, j-1 を設定する。

4. 使用例

区間 $[0.1, 0.9] \times [0.1, 0.9]$ で100個の補間点での関数 $f(x, y) = x^3 y^3$ のB-spline補間式の補間係数を求め、ある点での補間値、微分値及び積分値を求める。

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define N 10
#define M 3
#define NTMAX 5
#define NT 3

MAIN__()
{
    int ierr, icon;
    int i, j, kf, kc, nx, ny, m, nxt, nyt, nxl, nyl, ivw[2*N+NTMAX*M];
    int iswx, iswy, ix, iy;
    double x[N], y[N], fxy[N][N], sx[N], sy[N], rnor[2*(NTMAX-NT)+1][3];
    double c[NTMAX+M-1][NTMAX+M-1], xt[NTMAX], yt[NTMAX];
    double vw[158];
    double p, h, vx, vy, f, fx, rnot;

    /* initialize data */
    nx = N;
    ny = N;
    m = M;
    nxt = NT;
    nyt = NT;
    nxl = NTMAX;
    nyl = NTMAX;
    rnot = nx*ny;
    kf = N;
    kc = NTMAX+M-1;
    p = 0.1;
    h = 0.8/(nx-1);
    for (i=0; i<nx; i++) {
        x[i] = p+i*h;
        y[i] = x[i];
        sx[i] = 1e-6; /* make up some error values */
        sy[i] = sx[i];
    }
    for (i=0; i<nx; i++) {
        fx = x[i]*x[i]*x[i];
        for (j=0; j<ny; j++) {
            fxy[i][j] = fx*y[j]*y[j]*y[j];
        }
    }
    p = 0;
    h = 1.0/(nxt-1);
    for (i=0; i<nxt; i++) {
        xt[i] = p+i*h;
        yt[i] = xt[i];
    }
    /* calculate B-spline smoothing coefficients */
    ierr = c_dbfcd2(x, nx, y, ny, (double*)fxy, kf, sx, sy, m,
                   xt, &nxt, yt, &nyt, nxl, nyl, rnot,
                   (double*)c, kc, (double*)rnor, vw, ivw, &icon);
    if (icon >= 20000) {
        printf("ERROR: c_dbfcd2 failed with icon = %d\n", icon);
        exit(1);
    }
    ix = 1;
    iy = ix;
    vx = 0.5;
    vy = vx;
    for (iswx=-1; iswx<=m; iswx++) {
        iswy = iswx;
        /* calculate value at point */
        ierr = c_dbbfd1(m, xt, nxt, yt, nyt, (double*)c, kc,
                       iswx, vx, &ix, iswy, vy, &iy, &f, vw, &icon);
        if (icon >= 20000) {
            printf("ERROR: c_dbbfd1 failed with icon = %d\n", icon);
        }
    }
}
```



```
        exit(1);
    }
    if (iswx == -1)
        printf("icon = %i    integral = %12.6e\n", icon, f);
    else if (iswx == 0)
        printf("icon = %i    value = %12.6e\n", icon, f);
    else
        printf("icon = %i    derivative %i = %12.6e\n", icon, iswx, f);
    }
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の BSFD1 の項目を参照のこと。

c_dbsvec

実対称バンド行列の固有ベクトル (逆反復法)

<pre>ierr = c_dbsvec(a, n, nh, nv, e, ev, k, vw, &icon);</pre>

1. 機能

次数 n , バンド幅 h の実対称バンド行列 $\mathbf{A}$ において, n_v 個の固有値 $\lambda_1, \lambda_2, \dots, \lambda_{n_v}$ が与えられたとき, それらに対応する固有ベクトルを, 逆反復法によって求める.

2. 引数

呼出し形式:

```
ierr = c_dbsvec(a, n, nh, nv, e, (double *) ev, k, vw, &icon);
```

引数の説明:

a	double a[Alen]	入力	実対称バンド行列 $\mathbf{A}$. 対称バンド行列用圧縮格納法. 格納法の詳細については“1.3.4 バンド行列の格納方法”を参照のこと. $Alen = n(h+1) - h(h+1)/2$.
n	int	入力	行列 $\mathbf{A}$ の次数 n .
nh	int	入力	バンド幅 h .
nv	int	入力	求める固有ベクトルの数 n_v ($\neq 0$). $nv < 0$ のときは $ nv $ とする.
e	double e[nv]	入力	固有値. $ev[i-1] = \lambda_i, i=1, \dots, n_v$ のように格納する.
ev	double ev[nv][k]	出力	固有ベクトル. 固有ベクトルは行方向に格納される.
k	int	入力	配列 ev の整合寸法 ($\geq n$).
vw	double vw[2n(h+1)]	作業領域	
icon	int	入力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	$nh = 0$	正常終了.
15000	固有ベクトルのすべてを求めることはできなかった.	求められなかった固有ベクトルを 0 ベクトルとする.
20000	固有ベクトルが一つも求められなかった.	すべての固有ベクトルを 0 ベクトルとする.
30000	次のいずれかであった: $nh < 0$ 又は $nh \geq n$ $k < n$ $nv = 0$ 又は $nv > n$	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

本関数は実対称バンド行列用である. 実対称行列の固有値及び固有ベクトルを求める場合は, 関数 c_dseig1 又は c_dvseg2 を用いる. また, 実対称 3 重対角行列の固有値及び固有ベクトルを求める場合は, 関数 c_dteig1 又は c_dteig2 を用いること.

b) 密集した固有値に対応する固有ベクトル

固有値がある狭い範囲に密集しているときには、対応する固有ベクトルを求めるための逆反復法が収束しないことがある。このような場合には icon は 15000 又は 20000 となり、求められなかった固有ベクトルは 0 ベクトルになっている。

4. 使用例

実対称バンド行列 **A** の固有値を c_dbseg で求め、固有ベクトルを逆反復法で求める。

```
#include <stdio.h>
#include <stdlib.h>
#include "cssl.h" /* standard C-SSL II header file */
#define NMAX 15
#define NHMAX 2

MAIN__()
{
    int ierr, icon;
    int n, nh, m, nv, i, j, k, ij;
    double e[NMAX], ev[NMAX][NMAX];
    double a[NMAX*(NHMAX+1)-NHMAX*(NHMAX+1)/2];
    double b[NMAX*(NHMAX+1)-NHMAX*(NHMAX+1)/2];
    double vw[2*NMAX*(NHMAX+1)], epst;

    /* initialize matrix */
    n = NMAX;
    k = NMAX;
    nh = NHMAX;
    a[0] = 10;
    a[1] = -3;
    a[2] = 10;
    ij = (nh+1)*nh/2;
    for (i=0; i<n-nh; i++) {
        a[ij] = -6;
        a[ij+1] = -3;
        a[ij+2] = 10;
        ij = ij+nh+1;
    }
    /* save copy of a */
    for (i=0; i<n*(nh+1)-nh*(nh+1)/2; i++) b[i] = a[i];
    /* find eigenvalues and eigenvectors */
    m = n;
    nv = 0;
    epst = -1;
    ierr = c_dbseg(b, n, nh, m, nv, epst, e, (double*)ev, k, vw, &icon);
    if (icon > 20000) {
        printf("ERROR: c_dbseg failed with icon = %i\n", icon);
        exit (1);
    }
    /* find eigenvectors using dbsvec */
    nv = m;
    ierr = c_dbsvec(a, n, nh, nv, e, (double*)ev, k, vw, &icon);
    if (icon > 10000) {
        printf("ERROR: c_dbsvec failed with icon = %i\n", icon);
        exit (1);
    }
    printf("icon = %i\n", icon);
    /* print eigenvalues and eigenvectors */
    for (i=0; i<m; i++) {
        printf("eigenvalue: %7.4f\n", e[i]);
        printf("eigenvector: ");
        for (j=0; j<n; j++)
            printf("%7.4f ", ev[i][j]);
        printf("\n");
    }
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の BSVEC の項目を参照のこと。

c_dbtrid

実対称バンド行列の実対称 3 重対角行列への変換 (ルティスハウザー・シュワルツ法)

```
ierr = c_dbtrid(a, n, nh, d, sd, &icon);
```

1. 機能

次数 n , バンド幅 h の実対称バンド行列 $\mathbf{A}$ を, ルティスハウザー・シュワルツの直交相似変換により, 実対称 3 重対角行列 $\mathbf{T}$ に変換する.

$$\mathbf{T} = \mathbf{Q}_s^T \mathbf{A} \mathbf{Q}_s,$$

ただし $\mathbf{Q}_s$ は直交行列である. $0 \leq h \ll n$ であること.

2. 引数

呼出し形式:

```
ierr = c_dbtrid(a, n, nh, d, sd, &icon);
```

引数の説明:

a	double a[Alen]	入力	実対称バンド行列 $\mathbf{A}$. 対称バンド行列用圧縮格納法. 格納法の詳細については“1.3.4 バンド行列の格納方法”を参照のこと. $Alen = n(h+1) - h(h+1)/2$.
		出力	演算後, 内容は保存されない.
n	int	入力	行列 $\mathbf{A}$ の次数 n .
nh	int	入力	バンド幅 h .
d	double d[n]	出力	3 重対角行列 $\mathbf{T}$ の主対角要素.
sd	double sd[n]	出力	3 重対角行列 $\mathbf{T}$ の副対角要素. sd[i-1], $i=2, \dots, n$ のように格納され, sd[0] には 0 がセットされる.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	nh = 0 又は nh = 1	変換しない.
30000	nh < 0 又は nh ≥ n	処理を打ち切る.

3. 使用上の注意

a) ルティスハウザー・シュワルツ法について

実対称 3 重対角行列への変換法であるハウスホルダー法と比較すると, バンド幅と次数の比 $r = h/n$ が小さいときは, 記憶容量の点でも計算量の点でも本関数のルティスハウザー法がまさるが, r が 1/6 程度を超えると, 少なくとも計算量と精度に関してはハウスホルダー法がまさる.

4. 使用例

実対称バンド行列を 3 重対角行列に変換した後, 関数 c_dbst1 により固有値を計算する.

```
#include <stdio.h>
#include <stdlib.h>
#include "cssl.h" /* standard C-SSL II header file */
```

```

#define NMAX 15
#define NHMAX 2

MAIN__()
{
    int ierr, icon;
    int n, nh, m, i, k, ij;
    double a[NMAX*(NHMAX+1)-NHMAX*(NHMAX+1)/2], e[NMAX];
    double sd[NMAX], d[NMAX], vw[NMAX+2*NMAX], epst;

    /* initialize matrix */
    n = NMAX;
    k = NMAX;
    nh = NHMAX;
    a[0] = 10;
    a[1] = -3;
    a[2] = 10;
    ij = (nh+1)*nh/2;
    for (i=0;i<n-nh;i++) {
        a[ij] = -6;
        a[ij+1] = -3;
        a[ij+2] = 10;
        ij = ij+nh+1;
    }
    /* reduce to tridiagonal form */
    ierr = c_dbtrid(a, n, nh, d, sd, &icon);
    if (icon > 10000 ) {
        printf("ERROR: c_dbtrid failed with icon = %i\n", icon);
        exit (1);
    }
    /* find eigenvalues using c_dbsct1 */
    m = n;
    epst = 1e-6;
    ierr = c_dbsct1(d, sd, n, m, epst, e, vw, &icon);
    if (icon > 10000 ) {
        printf("ERROR: c_dbsct1 failed with icon = %i\n", icon);
        exit (1);
    }
    printf("icon = %i\n", icon);
    /* print eigenvalues */
    printf("eigenvalues:\n");
    for (i=0;i<m;i++) {
        printf("%7.4f  ", e[i]);
    }
    printf("\n");
    /* find eigenvalues using c_dtrql */
    ierr = c_dtrql(d, sd, n, e, &m, &icon);
    if (icon >= 20000 ) {
        printf("ERROR: c_dbtrql failed with icon = %i\n", icon);
        exit (1);
    }
    printf("icon = %i\n", icon);
    /* print eigenvalues */
    printf("eigenvalues:\n");
    for (i=0;i<m;i++) {
        printf("%7.4f  ", e[i]);
    }
    printf("\n");
    return(0);
}

```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の BTRID の項目を参照のこと。

c_dby0

第2種0次ベッセル関数 $Y_0(x)$
<code>ierr = c_dby0(x, &by, &icon);</code>

1. 機能

第2種0次ベッセル関数 $Y_0(x)$ の値,

$$Y_0(x) = \frac{2}{\pi} \left[J_0(x) \left\{ \log(x/2) + \gamma \right\} - \sum_{k=1}^{\infty} \frac{(-1)^k (x/2)^{2k}}{(k!)^2} \cdot \sum_{m=1}^k \frac{1}{m} \right]$$

を有理式及び漸近形式の近似式を用いて計算する. ただし, $x > 0$ であること.

($J_0(x)$: 第1種0次ベッセル関数, γ : オイラー定数)

2. 引数

呼出し形式:

```
ierr = c_dby0(x, &by, &icon);
```

引数の説明:

x	double	入力	独立変数 x .
by	double	出力	関数値 $Y_0(x)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
20000	$x \geq t_{\max}$ であった.	by = 0 とする.
30000	$x \leq 0$ であった.	by = 0 とする.

3. 使用上の注意

a) 引数 x

引数 x の範囲は,

$$0 < x < t_{\max}$$

であること.

x が上記範囲を超えると $\sin(x - \frac{\pi}{4})$, $\cos(x - \frac{\pi}{4})$ の値が不正確になるためである.

4. 使用例

$Y_0(x)$ の値を x が 0 から 100 まで 1 おきに計算し数表を作る.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
```

```
{
  int ierr, icon;
  double x, by;
  int i;

  for (i=1;i<=100;i++) {
    x = (double)i;
    /* calculate Bessel function */
    ierr = c_dby0(x, &by, &icon);
    if (icon == 0)
      printf("x = %4.2f    by = %f\n", x, by);
    else
      printf("ERROR: x = %4.2f    by = %f    icon = %i\n", x, by, icon);
  }
  return(0);
}
```

5. 手法概要

ベッセル関数 $Y_0(x)$ は, x の値によって近似式の形が異なる.

- $0 < x \leq 8$ のとき, べき級数展開の有理近似式を用いて計算する.
- $x > 8$ のとき, 漸近展開の近似式を用いて計算する.

計算方法の詳細については“富士通 SSL II 使用手引書”の BY0 の項目及び[48]を参照のこと.

c_dby1

第2種1次ベッセル関数 $Y_1(x)$
<code>ierr = c_dby1(x, &by, &icon);</code>

1. 機能

第2種1次ベッセル関数 $Y_1(x)$ の値,

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

を有理式及び漸近形式の近似式を用いて計算する. ただし, $x > 0$ であること.

($J_1(x)$: 第1種1次ベッセル関数, γ : オイラ一定数)

2. 引数

呼出し形式:

```
ierr = c_dby1(x, &by, &icon);
```

引数の説明:

x	double	入力	独立変数 x .
by	double	出力	関数値 $Y_1(x)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
20000	$x \geq t_{\max}$ であった.	by = 0 とする.
30000	$x \leq 0$ であった.	by = 0 とする.

3. 使用上の注意

a) 引数 x

引数 x の範囲は,

$$0 < x < t_{\max}$$

であること.

x が上記範囲を超えると $\sin(x - \frac{3\pi}{4})$, $\cos(x - \frac{3\pi}{4})$ の値が不正確になるためである.

4. 使用例

$Y_1(x)$ の値を x が 0 から 100 まで 1 おきに計算し数表を作る.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
```



```
{
  int ierr, icon;
  double x, by;
  int i;

  for (i=1;i<=100;i++) {
    x = (double)i;
    /* calculate Bessel function */
    ierr = c_dby1(x, &by, &icon);
    if (icon == 0)
      printf("x = %4.2f    by = %f\n", x, by);
    else
      printf("ERROR: x = %4.2f    by = %f    icon = %i\n", x, by, icon);
  }
  return(0);
}
```

5. 手法概要

ベッセル関数 $Y_1(x)$ は, x の値によって近似式の形が異なる.

- $0 < x \leq 8$ のとき, べき級数展開の有理近似式を用いて計算する.
- $x > 8$ のとき, 漸近展開の近似式を用いて計算する.

計算方法の詳細については“富士通 SSL II 使用手引書”の BY1 の項目及び[48]を参照のこと.

c_dbyn

第 2 種整数次ベッセル関数 $Y_n(x)$
<code>ierr = c_dbyn(x, n, &by, &icon);</code>

1. 機能

第 2 種整数次ベッセル関数 $Y_n(x)$ の値,

$$Y_n(x) = \frac{2}{\pi} [J_n(x) \{\log(x/2) + \gamma\}] - \frac{1}{\pi} \sum_{k=0}^{n-1} \frac{(n-k-1)!}{k!} (x/2)^{2k-n} \\ - \frac{1}{\pi} \sum_{k=0}^{\infty} \frac{(-1)^k}{k!(n+k)!} \cdot \{(x/2)^{2k+n} (\phi_k + \phi_{k+n})\}$$

を漸化式を用いて計算する. ただし, 上式において, $x > 0$ であり, γ はオイラー定数, $J_n(x)$ は第 1 種整数次ベッセル関数, また,

$$\phi_0 = 0 \\ \phi_L = \sum_{m=1}^L \frac{1}{m} \quad (L \geq 1)$$

である.

2. 引数

呼出し形式:

```
ierr = c_dbyn(x, n, &by, &icon);
```

引数の説明:

x	double	入力	独立変数 x .
n	int	入力	$Y_n(x)$ の次数 n .
by	double	出力	関数値 $Y_n(x)$.
icon	int	出力	コンディションコード. 下の表を参照. ただし, $n=0$ 又は, $n=1$ のときはそれぞれ関数 <code>c_dby0</code> , <code>c_dby1</code> の <code>icon</code> に準ずる.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
20000	$x \geq t_{\max}$ であった.	<code>by = 0</code> とする.
30000	$x \leq 0$ であった.	<code>by = 0</code> とする.

3. 使用上の注意

a) 引数 x

引数 x の範囲は,

$$0 < x < t_{\max}$$

であること.

x が上記範囲を超えると $\sin(x - \frac{\pi}{4})$, $\cos(x - \frac{\pi}{4})$ の値が不正確になるためである.

b) $Y_0(x)$ 及び $Y_1(x)$ の計算について

$Y_0(x)$ 及び $Y_1(x)$ を計算するには, 本関数よりもそれぞれ, 関数 c_dby0, 関数 c_dby1 を直接用いた方がよい.

4. 使用例

$Y_n(x)$ の値を x が 0 から 10 まで 1 おきに, n が 20 から 30 まで 10 おきに計算し数表を作る.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
    double x, by;
    int i, n;

    for (n=20;n<=30;n=n+10)
        for (i=1;i<=10;i++) {
            x = (double)i;
            /* calculate Bessel function */
            ierr = c_dbyn(x, n, &by, &icon);
            if (icon == 0)
                printf("x = %4.2f    n = %i    by = %e\n", x, n, by);
            else
                printf("ERROR: x = %4.2f    n = %i    by = %e    icon = %i\n",
                    x, n, by, icon);
        }
    return(0);
}
```

5. 手法概要

ベッセル関数 $Y_n(x)$ は漸化式を用いて計算される. ただし, $Y_0(x)$ 及び $Y_1(x)$ は, それぞれ Fortran ルーチンである DBY0, DBY1 を利用している.

計算方法の詳細については“富士通 SSL II 使用手引書”の BYN の項目を参照のこと.

c_dbyr

第2種実数次ベッセル関数 $Y_\nu(x)$
<code>ierr = c_dbyr(x, v, &by, &icon);</code>

1. 機能

第2種実数次ベッセル関数 $Y_\nu(x)$ の値,

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

の値を, 級数展開の変形による方法及び τ 法を用いて計算する. ただし, $x > 0$, $\nu \geq 0$ であること.

($J_\nu(x)$: 第1種ベッセル関数)

2. 引数

呼出し形式:

```
ierr = c_dbyr(x, v, &by, &icon);
```

引数の説明:

x	double	入力	独立変数 x .
v	double	入力	$Y_\nu(x)$ の次数 ν .
by	double	出力	関数値 $Y_\nu(x)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	次のいずれかであった. $x=0$ 又は, by がオーバーフローするほど大きな値となった. $x \geq t_{\max}$ であった.	- by には浮動小数点数の負の最大値を出力する. - by = 0 とする.
30000	$x < 0$ 又は $\nu < 0$ であった.	by = 0 とする.

3. 使用上の注意

a) $Y_0(x)$ 及び $Y_1(x)$ の計算について

$Y_0(x)$ 及び $Y_1(x)$ を計算するには, 本関数よりもそれぞれ, 関数 c_dby0, 関数 c_dby1 を直接用いた方がよい.

b) 評価順序

$Y_\nu(x), Y_{\nu+1}(x), Y_{\nu+2}(x), \dots, Y_{\nu+M}(x)$ の値をいっせいに必要とするときには, 本関数により $Y_\nu(x)$ と $Y_{\nu+1}(x)$ を求め, 次の漸化式を反復適用して $Y_{\nu+2}(x), Y_{\nu+3}(x), \dots, Y_{\nu+M}(x)$ と次数の高いものへと順次求めていくとよい.

$$Y_{v+1}(x) = \frac{2v}{x} Y_v(x) - Y_{v-1}(x)$$

c) 計算の効率化

x が大きいところでは, 続けて同じ v の値で, 本関数が呼ばれるときには, 計算の共通部分を通過し, 能率良く $Y_v(x)$ の値を計算できるようにしてある.

4. 使用例

$Y_v(x)$ の値を x が 0 から 10 まで 1 おきに, v が 0.5 から 0.8 まで 0.3 おきに計算し数表を作る.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
    double v, x, by;
    int nv, i;

    for (i=1; i<=10; i++) {
        x = (double)i;
        for (nv=50; nv<=80; nv=nv+30) {
            v = (double)nv/100;
            /* calculate Bessel function */
            ierr = c_dbyr(v, x, &by, &icon);
            if (icon == 0)
                printf("x = %5.2f    v = %5.2f    by = %e\n", x, v, by);
            else
                printf("ERROR: x = %5.2f    v = %5.2f    by = %e    icon = %i\n",
                    x, v, by, icon);
        }
    }
    return(0);
}
```

5. 手法概要

第 2 種実数次ベッセル関数 $Y_v(x)$ は, 級数展開の変形による方法及び τ 法を用いて計算する.

計算方法の詳細については“富士通 SSL II 使用手引書”の BYR の項目を参照のこと.

c_dcbn

複素変数第1種整数次変形ベッセル関数 $I_n(z)$
<code>ierr = c_dcbn(z, n, &zbi, &icon);</code>

1. 機能

複素変数第1種整数次変形ベッセル関数 $I_n(z)$ の値,

$$I_n(z) = \left(\frac{z}{2}\right)^n \sum_{k=0}^{\infty} \frac{(z^2/4)^k}{k!(n+k)!} \quad (1)$$

をべき級数展開式(1)及び漸化式を用いて計算する.

2. 引数

呼出し形式:

```
ierr = c_dcbn(z, n, &zbi, &icon);
```

引数の説明:

z	dcomplex	入力	独立変数 z .
n	int	入力	$I_n(z)$ の次数 n .
zbi	dcomplex	出力	関数値 $I_n(z)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	$ \operatorname{Re}(z) > \log(f_{\max})$ 又は, $ \operatorname{Im}(z) > \log(f_{\max})$ であった.	$\text{zbi.re} = 0, \text{zbi.im} = 0$ とする.

3. 使用上の注意

a) 引数 z

引数 z の範囲は,

$$|\operatorname{Re}(z)| \leq \log(f_{\max}) \quad \text{及び} \quad |\operatorname{Im}(z)| \leq \log(f_{\max})$$

であること.

b) 評価順序

$I_n(z), I_{n+1}(z), I_{n+2}(z), \dots, I_{n+M}(z)$ の値をいっせいに必要とするときには, 本関数により $I_{n+M}(z)$ と $I_{n+M-1}(z)$ を求め, 漸化式を反復適用して $I_{n+M-2}(z), I_{n+M-3}(z), \dots, I_n(z)$ と次数の低いものを求めていくとよい. 逆に, 本関数により $I_n(z)$ と $I_{n+1}(z)$ を求め, 漸化式により $I_{n+2}(z), I_{n+3}(z), \dots, I_{n+M}(z)$ と次数の高いものを求めていくことは不安定が生ずるので避けなければならない.

4. 使用例

$I_n(z)$ の値を $z=10+5i$ について, n を 1 及び 2 について計算する.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
    dcomplex z, zbi;
    int n;

    z.re = 10;
    z.im = 5;
    for (n=1;n<=2;n++) {
        /* calculate Bessel function */
        ierr = c_dcbin(z, n, &zbi, &icon);
        if (icon == 0)
            printf("z = {%.2f, %.2f}    n = %i    zbi = {%.2f, %.2f}\n",
                z.re, z.im, n, zbi.re, zbi.im);
        else
            printf("ERROR: z = {%.2f, %.2f}    n = %i\n",
                z.re, z.im, n, icon);
    }
    return(0);
}
```

5. 手法概要

ベッセル関数 $I_n(z)$ は, z の値によって計算方法が異なる.

- $|\operatorname{Re}(z)| + |\operatorname{Im}(z)| \leq 1$ のとき, ベキ級数展開式(1)を用いる.
- $|\operatorname{Re}(z)| + |\operatorname{Im}(z)| > 1$ のとき, 漸化式を用いる.
 m を適当に大きな整数 (z, n 及び要求精度によって決まる), δ を適当に小さい定数 (10^{-38}) とする.
 初期値を

$$G_{m+1}(z) = 0, \quad G_m(z) = \delta$$

として, $k = m, m-1, \dots, 1$ に対して漸化式

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

を反復適用する. このとき, $I_n(z)$ は,

$$I_n(z) \approx \frac{e^z G_n(z)}{\sum_{k=0}^m \varepsilon_k G_k(z)}$$

として求められる. ここで,

$$\varepsilon_k = \begin{cases} 1 & (k=0) \\ 2 & (k \geq 1) \end{cases}$$

である.

計算方法の詳細については“富士通 SSL II 使用手引書”のCBINの項目及び[133], [141]を参照のこと.

c_dcbjn

複素変数第1種整数次ベッセル関数 $J_n(z)$
<code>ierr = c_dcbjn(z, n, &zbj, &icon);</code>

1. 機能

複素変数第1種整数次ベッセル関数 $J_n(z)$ の値,

$$J_n(z) = \left(\frac{z}{2}\right)^n \sum_{k=0}^{\infty} \frac{(-z^2/4)^k}{k!(n+k)!} \quad (1)$$

をべき級数展開式(1)及び漸化式を用いて計算する.

2. 引数

呼出し形式:

```
ierr = c_dcbjn(z, n, &zbj, &icon);
```

引数の説明:

z	dcomplex	入力	独立変数 z .
n	int	入力	$J_n(z)$ の次数 n .
zbj	dcomplex	出力	関数値 $J_n(z)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	$ \operatorname{Re}(z) > \log(f_{\max})$ 又は, $ \operatorname{Im}(z) > \log(f_{\max})$ であった.	$\text{zbj.re} = 0, \text{zbj.im} = 0$ とする.

3. 使用上の注意

a) 引数 z

引数 z の範囲は,

$$|\operatorname{Re}(z)| \leq \log(f_{\max}) \quad \text{及び} \quad |\operatorname{Im}(z)| \leq \log(f_{\max})$$

であること.

b) 評価順序

$J_n(z), J_{n+1}(z), J_{n+2}(z), \dots, J_{n+M}(z)$ の値をいっせいに必要とするときには, 本関数により $J_{n+M}(z)$ と $J_{n+M-1}(z)$ を求め, 漸化式を反復適用して $J_{n+M-2}(z), J_{n+M-3}(z), \dots, J_n(z)$ と次数の低いものを求めていくとよい. 逆に, 本関数により $J_n(z)$ と $J_{n+1}(z)$ を求め, 漸化式により $J_{n+2}(z), J_{n+3}(z), \dots, J_{n+M}(z)$ と次数の高いものを求めていくことは不安定が生ずるので避けなければならない.

4. 使用例

$J_n(z)$ の値を $z = 10+5i$ について, n を 1 及び 2 について計算する.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
    dcomplex z, zbj;
    int n;

    z.re = 10;
    z.im = 5;
    for (n=1;n<=2;n++) {
        /* calculate Bessel function */
        ierr = c_dcbjn(z, n, &zbj, &icon);
        if (icon == 0)
            printf("z = {%4.2f, %4.2f}    n = %i    zbj = {%4.2f, %4.2f}\n",
                z.re, z.im, n, zbj.re, zbj.im);
        else
            printf("ERROR: z = {%4.2f, %4.2f}    n = %i\n",
                z.re, z.im, n, icon);
    }
    return(0);
}
```

5. 手法概要

ベッセル関数 $J_n(z)$ は, z の値によって計算方法が異なる.

- $|\operatorname{Re}(z)| + |\operatorname{Im}(z)| \leq 1$ のとき, ベキ級数展開式(1)を用いる.
- $|\operatorname{Re}(z)| + |\operatorname{Im}(z)| > 1$ のとき, 漸化式を用いる.
 m を適当に大きな整数 (z, n 及び要求精度によって決まる), δ を適当に小さい定数 (10^{-38}) とする.
 初期値を

$$F_{m+1}(z) = 0, \quad F_m(z) = \delta$$

として, $k = m, m-1, \dots, 1$ に対して漸化式,

$$F_{k-1}(z) = \frac{2k}{z} F_k(z) - F_{k+1}(z)$$

を反復適用する. このとき, $J_n(z)$ は,

$$J_n(z) \approx \frac{e^{-iz} F_n(z)}{\sum_{k=0}^m \varepsilon_k i^k F_k(z)}$$

として求められる. ここで,

$$\varepsilon_k = \begin{cases} 1 & (k=0) \\ 2 & (k \geq 1) \end{cases}$$

である.

計算方法の詳細については“富士通 SSL II 使用手引書”の CBJN の項目及び [133], [141] を参照のこと.

c_dcbjr

複素変数第 1 種実数次ベッセル関数 $J_\nu(z)$
<code>ierr = c_dcbjr(z, v, &zbj, &icon);</code>

1. 機能

複素変数第 1 種実数次ベッセル関数 $J_\nu(z)$ の値,

$$J_\nu(z) = \left(\frac{z}{2}\right)^\nu \sum_{k=0}^{\infty} \frac{(-z^2/4)^k}{k! \Gamma(\nu + k + 1)} \quad (1)$$

をべき級数展開式(1)及び漸化式を用いて計算する.

上式において, $(z/2)^\nu$ の値は, その主値を採用する. $(z/2)^\nu \equiv e^{\nu \log(z/2)}$ の主値は, $\log(z/2)$ の主値の選び方に依存するが, 本関数では Fortran 基本関数 CDLOG によって求められるものとなる. 通常は,

$$\log(z/2) \text{ の主値} = \log|z/2| + i \arg z, \quad -\pi < \arg z \leq \pi$$

である.

2. 引数

呼出し形式:

```
ierr = c_dcbjr(z, v, &zbj, &icon);
```

引数の説明:

z	dcomplex	入力	独立変数 z .
v	double	入力	$J_\nu(z)$ の次数 ν .
zbj	dcomplex	出力	関数値 $J_\nu(z)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	$ \operatorname{Re}(z) > \log(fl_{\max})$ 又は $ \operatorname{Im}(z) > \log(fl_{\max})$ であった.	<code>zbj.re = 0, zbj.im = 0</code> とする.
30000	$\nu < 0$ であった.	<code>zbj.re = 0, zbj.im = 0</code> とする.

3. 使用上の注意

a) 引数 **z** 及び, 引数 **v**

引数 z 及び, 引数 ν の範囲は,

$$|\operatorname{Re}(z)| \leq \log(fl_{\max}), \quad |\operatorname{Im}(z)| \leq \log(fl_{\max}) \quad \text{及び} \quad \nu \geq 0$$

であること.

b) 評価順序

$J_\nu(z), J_{\nu+1}(z), J_{\nu+2}(z), \dots, J_{\nu+M}(z)$ の値をいっせいに必要とするときには、本関数により $J_{\nu+M}(z)$ と $J_{\nu+M-1}(z)$ を求め、漸化式を反復適用して $J_{\nu+M-2}(z), J_{\nu+M-3}(z), \dots, J_\nu(z)$ と次数の低いものを求めていくとよい。逆に、本関数により $J_\nu(z)$ と $J_{\nu+1}(z)$ を求め、漸化式により $J_{\nu+2}(z), J_{\nu+3}(z), \dots, J_{\nu+M}(z)$ と次数の高いものを求めていくことは不安定が生ずるので避けなければならない。

4. 使用例

$J_\nu(z)$ の値を $z = 10+5i$ について、 ν を 0.1 から 10 まで 0.1 刻みで計算する。

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
    dcomplex z, zbj;
    int n;
    double v;

    z.re = 10;
    z.im = 5;
    for (n=1; n<=100; n++) {
        v = (double)n/10;
        /* calculate Bessel function */
        ierr = c_dcbjr(z, v, &zbj, &icon);
        if (icon == 0)
            printf("z = {%4.2f, %4.2f}    v = %5.2f    zbj = {%4.2f, %4.2f}\n",
                   z.re, z.im, v, zbj.re, zbj.im);
        else
            printf("ERROR: z = {%4.2f, %4.2f}    v = %5.2f\n",
                   z.re, z.im, v, icon);
    }
    return(0);
}
```

5. 手法概要

ベッセル関数 $J_\nu(z)$ は、 z の値によって計算方法が異なる。

- $|\operatorname{Re}(z)| + |\operatorname{Im}(z)| \leq 1$ のとき、べき級数展開式(1)を用いる。
- $|\operatorname{Re}(z)| + |\operatorname{Im}(z)| > 1$ のとき、漸化式を用いる。
 m を適当に大きな整数 (z, ν 及び要求精度によって決まる), δ を適当に小さい定数 (10^{-38}) とする。
 n 及び α を次式のように決める。

$$\nu = n + \alpha \quad (n: \text{整数}, 0 \leq \alpha < 1)$$

初期値を

$$F_{\alpha+m+1}(z) = 0, \quad F_{\alpha+m}(z) = \delta$$

として、 $k = m, m-1, \dots, 1$ に対して漸化式、

$$F_{\alpha+k-1}(z) = \frac{2(\alpha+k)}{z} F_{\alpha+k}(z) - F_{\alpha+k+1}(z)$$

を反復適用する。このとき、 $J_\nu(z)$ は、

$$J_\nu(z) \approx \frac{\frac{1}{2} \left(\frac{z}{2} \right)^\alpha \frac{\Gamma(2\alpha+1)}{\Gamma(\alpha+1)} e^{-iz} F_{\alpha+n}(z)}{\sum_{k=0}^m \frac{(\alpha+k)\Gamma(2\alpha+k) i^k}{k!} F_{\alpha+k}(z)}$$

として求められる.

計算方法の詳細については“富士通 SSL II 使用手引書”の CBJR の項目及び [133], [141]を参照のこと.

c_dcbkn

複素変数第 2 種整数次変形ベッセル関数 $K_n(z)$

ierr = c_dcbkn(z, n, &zbk, &icon);

1. 機能

複素変数第 2 種整数次変形ベッセル関数 $K_n(z)$ の値,

$$\begin{aligned}
 K_n(z) &= K_{-n}(z) \\
 &= (-1)^{n-1} \{ \gamma + \log(z/2) \} I_n(z) + \frac{(-1)^n}{2} \left(\frac{z}{2} \right)^n \sum_{k=0}^{\infty} \frac{(z^2/4)^k}{k!(n+k)!} (\phi_k + \phi_{k+n}) \\
 &\quad + \frac{1}{2} \left(\frac{z}{2} \right)^{-n} \sum_{k=0}^{n-1} \frac{(n-k-1)!}{k!} \left(-\frac{z^2}{4} \right)^k
 \end{aligned} \tag{1}$$

を漸化式による方法及び τ 法を用いて計算する. ただし, 上式において, $n=0$ のとき, 最後の項は 0 であり, γ はオイラー一定数, $I_n(z)$ は第 1 種変形ベッセル関数, また,

$$\begin{aligned}
 \phi_0 &= 0 \\
 \phi_L &= \sum_{m=1}^L \frac{1}{m} \quad (L \geq 1)
 \end{aligned}$$

である.

2. 引数

呼出し形式:

```
ierr = c_dcbkn(z, n, &zbk, &icon);
```

引数の説明:

z	dcomplex	入力	独立変数 z .
n	int	入力	$K_n(z)$ の次数 n .
zbk	dcomplex	出力	関数値 $K_n(z)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	次のいずれかであった. $\operatorname{Re}(z) > \log(f_{\max})$ $\operatorname{Re}(z) < 0$ かつ $\operatorname{Im}(z) > \log(f_{\max})$ $\operatorname{Re}(z) \geq 0$ かつ $\operatorname{Im}(z) \geq t_{\max}$	zbk.re = 0, zbk.im = 0 とする.
30000	$ z = 0$ であった.	zbk.re = 0, zbk.im = 0 とする.

3. 使用上の注意

a) 引数 z

計算途中のオーバーフロー、アンダフローを前もって防ぐため、コンディションコードで示した範囲の z はエラーとする。

b) 評価順序

$\operatorname{Re}(z) \geq 0$ の場合において、 $K_n(z), K_{n+1}(z), K_{n+2}(z), \dots, K_{n+M}(z)$ の値をいっせいに必要とするときには、本関数により $K_n(z)$ と $K_{n+1}(z)$ を求め、漸化式を反復適用して $K_{n+2}(z), K_{n+3}(z), \dots, K_{n+M}(z)$ と次数の高いものへと順次求めていくとよい。 $\operatorname{Re}(z) < 0$ の場合には、上記のことは安定ではないので、必要とする次数について本関数を呼ばなければならない。

4. 使用例

$K_n(z)$ の値を $z = 1+2i$ について、 n を 1 及び 2 について計算する。

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
    dcomplex z, zbk;
    int n;

    z.re = 1;
    z.im = 2;
    for (n=1; n<=2; n++) {
        /* calculate Bessel function */
        ierr = c_dcbkn(z, n, &zbk, &icon);
        if (icon == 0)
            printf("z = {%.2f, %.2f}    n = %i    zbk = {%.2f, %.2f}\n",
                z.re, z.im, n, zbk.re, zbk.im);
        else
            printf("ERROR: z = {%.2f, %.2f}    n = %i\n",
                z.re, z.im, n);
    }
    return(0);
}
```

5. 手法概要

ベッセル関数 $K_n(z)$ は、 z の値によって計算方法が異なる。

- $\operatorname{Re}(z) \geq 0$ の場合

$K_n(z)$ は、 $K_0(z)$ 及び $K_1(z)$ から、漸化式、

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

により計算できる。 $K_0(z)$ 及び $K_1(z)$ の計算は、領域によって計算方法が異なる。

- $|\operatorname{Im}(z)| \geq -4 \cdot \operatorname{Re}(z) + 8$ のとき、 τ 法を用いて計算する。
- $|\operatorname{Im}(z)| < -4 \cdot \operatorname{Re}(z) + 8$ のとき、漸化式を用いて計算する。

- $\operatorname{Re}(z) < 0$ の場合

- $|\operatorname{Im}(z)| \geq 0$ のとき、以下の関係式を用いて計算する。

$$K_n(z) = (-1)^n K_n(-z) - \pi i I_n(-z)$$

- $|\operatorname{Im}(z)| < 0$ のとき、以下の関係式を用いて計算する。

$$K_n(z) = (-1)^n K_n(-z) + \pi i I_n(-z)$$

計算方法の詳細については“富士通 SSL II 使用手引書”の CBKN の項目及び [142]を参照のこと.

c_dcblnc

複素行列の平衡化

ierr = c_dcblnc(za, k, n, dv, &icon);

1. 機能

n 次の複素行列 $\mathbf{A}$ を(1)の対角相似変換により, 平衡化する.

$$\tilde{\mathbf{A}} = \mathbf{D}^{-1} \mathbf{A} \mathbf{D} \quad (1)$$

ここで, 平衡化するとは, 変換後の複素行列 $\tilde{\mathbf{A}}$ について第 i 行 ($i = 1, \dots, n$) 要素のノルムの和と, 第 i 列要素のノルムの和をほぼ等しくすることである. また, 要素のノルムとは, 複素数 $z = x + iy$ に対して, $\|z\|_1 = |x| + |y|$ を考える. $\mathbf{D}$ は実対角行列である. $n \geq 1$ なること.

2. 引数

呼出し形式:

```
ierr = c_dcblnc((dcomplex *) za, k, n, dv, &icon);
```

引数の説明:

za	dcomplex	入力	複素行列 $\mathbf{A}$.
	za[n][k]	出力	平衡化後の複素行列 $\tilde{\mathbf{A}}$.
k	int	入力	配列 za の整合寸法 ($\geq n$).
n	int	入力	複素行列 $\mathbf{A}$ 及び $\tilde{\mathbf{A}}$ の次数 n .
dv	double dv[n]	出力	スケーリングファクタ ($\mathbf{D}$ の対角要素).
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
10000	$n = 1$	平衡化は行わない.
30000	次のいずれかであった. • $n < 1$ • $k < n$	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

行列の各要素のノルムが不揃いな場合, この行列の固有値及び固有ベクトルの計算値の精度が悪くなりやすい. 本関数は, これを防ぐためのものである.

行列の各要素の大きさがほぼ揃っているときには, 本関数により何等の変換も行われないので, 本関数の実行を省略した方がよい.

本関数は行列の平衡化にあたり, 行又は列の要素が, 対角要素を除いてすべて零となるような行 (又は列) がある場合, その行 (又は列) と対応する列 (又は行) については平衡化を省略する.

b) 固有ベクトル

行列 $\mathbf{A}$ の固有ベクトル $\mathbf{x}$ を求めるときは, 本関数により平衡化した行列 $\tilde{\mathbf{A}}$ の固有ベクトル $\tilde{\mathbf{x}}$ に対して, (2)の逆変換を行う必要がある.

$$\mathbf{x} = \mathbf{D}\tilde{\mathbf{x}}. \quad (2)$$

(2) の逆変換は関数 c_dchbk2 により実行できる.

4. 使用例

n 次の複素行列 $\mathbf{A}$ を平衡化した後, 関数 c_dches2 により複素ヘッセンベルグ行列に変換し, 固有値を関数 c_dchsqr で求める.

```
#include <stdio.h>
#include <stdlib.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 5

MAIN__()
{
    int ierr, icon;
    int n, i, j, k, m, mode, ip[NMAX], ind[NMAX];
    dcomplex za[NMAX][NMAX], ze[NMAX], zev[NMAX][NMAX], zaw[NMAX+1][NMAX];
    double dv[NMAX];

    /* initialize matrix */
    n = NMAX;
    k = NMAX;
    for (i=0;i<n;i++) {
        za[i][i].re = n-i;
        za[i][i].im = 0;
        for (j=0;j<i;j++) {
            za[i][j].re = n-i;
            za[j][i].re = n-i;
            za[i][j].im = 0;
            za[j][i].im = 0;
        }
    }
    /* balance matrix A */
    ierr = c_dcbinc((dcomplex*)za, k, n, dv, &icon);
    if (icon > 10000 ) {
        printf("ERROR: c_dcbinc failed with icon = %i\n", icon);
        exit (1);
    }
    /* reduce matrix to Hessenberg form */
    ierr = c_dches2((dcomplex*)za, k, n, ip, &icon);
    if (icon > 10000 ) {
        printf("ERROR: c_dches2 failed with icon = %i\n", icon);
        exit (1);
    }
    for (i=0;i<n;i++)
        for (j=0;j<n;j++) {
            zaw[i][j].re = za[i][j].re;
            zaw[i][j].im = za[i][j].im;
        }
    /* find eigenvalues */
    ierr = c_dchsqr((dcomplex*)zaw, k, n, ze, &m, &icon);
    if (icon >= 20000 ) {
        printf("ERROR: c_dchsqr failed with icon = %i\n", icon);
        exit (1);
    }
    for (i=0;i<m;i++) ind[i] = 1;
    /* find eigenvectors for given eigenvalues */
    ierr = c_dchvec((dcomplex*)za, k, n, ze,
                    ind, m, (dcomplex*)zev, (dcomplex*)zaw, &icon);
    if (icon >= 20000 ) {
        printf("ERROR: c_dchvec failed with icon = %i\n", icon);
        exit (1);
    }
    /* back transformation to find e-vectors of A */
    ierr = c_dchbk2((dcomplex*)zev, k, n, ind, m,
                    (dcomplex*)za, ip, dv, &icon);
    if (icon > 10000 ) {
        printf("ERROR: c_dchbk2 failed with icon = %i\n", icon);
        exit (1);
    }
    /* normalize e-vectors */
}
```

```
mode = 2;
ierr = c_dcnrml((dcomplex*)zev, k, n, ind, m, mode, &icon);
if (icon > 10000 ) {
    printf("ERROR: c_dcnrml failed with icon = %i\n", icon);
    exit (1);
}
printf("icon = %i\n", icon);
/* print eigenvalues and eigenvectors */
for (i=0;i<m;i++) {
    if (ind[i] != 0) {
        printf("eigenvalue:  %7.4f+i*%7.4f\n", ze[i].re, ze[i].im);
        printf("eigenvector:  ");
        for (j=0;j<n;j++)
            printf("%7.4f+i*%7.4f  ", zev[i][j].re, zev[i][j].im);
        printf("\n");
    }
}
return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の CBLNC の項目及び[119]を参照のこと.

c_dcbyn

複素変数第2種整数次ベッセル関数 $Y_n(z)$
<code>ierr = c_dcbyn(z, n, &zby, &icon);</code>

1. 機能

複素変数第2種整数次ベッセル関数 $Y_n(z)$ の値,

$$\begin{aligned}
 Y_n(z) &= (-1)^n Y_{-n}(z) \\
 &= \frac{2}{\pi} \left\{ \gamma + \log\left(\frac{z}{2}\right) \right\} J_n(z) - \frac{1}{\pi} \left(\frac{z}{2}\right)^n \sum_{k=0}^{\infty} \frac{(-z^2/4)^k}{k!(n+k)!} (\phi_k + \phi_{k+n}) \\
 &\quad - \frac{1}{\pi} \left(\frac{z}{2}\right)^{-n} \sum_{k=0}^{n-1} \frac{(n-k-1)!}{k!} \left(\frac{z^2}{4}\right)^k
 \end{aligned}$$

を漸化式による方法及び ϵ 法を用いて計算する。ただし、上式において、 $n=0$ のとき、最後の項は 0 であり、 γ はオイラー定数、 $J_n(z)$ は第1種ベッセル関数、また、

$$\begin{aligned}
 \phi_0 &= 0 \\
 \phi_L &= \sum_{m=1}^L \frac{1}{m} \quad (L \geq 1)
 \end{aligned}$$

である。

2. 引数

呼出し形式:

```
ierr = c_dcbyn(z, n, &zby, &icon);
```

引数の説明:

z	dcomplex	入力	独立変数 z .
n	int	入力	$Y_n(z)$ の次数 n .
zby	dcomplex	出力	関数値 $Y_n(z)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	$ \operatorname{Re}(z) > \log(f_{\max})$ 又は, $ \operatorname{Im}(z) > \log(f_{\max})$ であった.	<code>zby.re = 0</code> , <code>zby.im = 0</code> とする.
30000	$ z = 0$ であった.	<code>zby.re = 0</code> , <code>zby.im = 0</code> とする.

3. 使用上の注意

a) 引数 z

計算途中のアンダフローを前もって防ぐため、コンディションコードで示した z の範囲はエラーとする。

b) 評価順序

$Y_n(z), Y_{n+1}(z), Y_{n+2}(z), \dots, Y_{n+M}(z)$ の値をいっせいに必要とするときには、手法概要で述べる方法を用いるとよい。

4. 使用例

$Y_n(z)$ の値を $z = 1+2i$ について、 n を 1 及び 2 について計算する。

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
    dcomplex z, zby;
    int n;

    z.re = 1;
    z.im = 2;
    for (n=1;n<=2;n++) {
        /* calculate Bessel function */
        ierr = c_dcbyn(z, n, &zby, &icon);
        if (icon == 0)
            printf("z = {%4.2f, %4.2f}    n = %i    zby = {%4.2f, %4.2f}\n",
                z.re, z.im, n, zby.re, zby.im);
        else
            printf("ERROR: z = {%4.2f, %4.2f}    n = %i\n",
                z.re, z.im, n);
    }
    return(0);
}
```

5. 手法概要

複素変数第 2 種整数次ベッセル関数 $Y_n(z)$ は、関係式

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

を用いて計算する。ここで、 $I_n(-iz)$ の値は、漸化式で計算を行なっている Fortran ルーチン DCBIN (c_dcbn) を用いて計算し、 $K_n(-iz)$ の値は、漸化式及び τ 法で計算を行なっている Fortran ルーチン DCBKN (c_dcbkn) を用いて計算する。

なお、 $Y_n(z), Y_{n+1}(z), Y_{n+2}(z), \dots, Y_{n+M}(z)$ の値をいっせいに必要とするとき、これらの値を能率的に求めるには以下のようにすればよい。関数 c_dcbn により、 $I_{n+M}(-iz)$ 及び $I_{n+M-1}(-iz)$ の値を求め、漸化式を反復適用して次数の低いものへと、 $I_n(-iz)$ まで順次求める。また、関数 c_dcbkn により、 $K_n(-iz)$ 及び $K_{n+1}(-iz)$ の値を求め、漸化式を反復適用して次数の高いものへと、 $K_{n+M}(-iz)$ まで順次求めて、式(1)により、必要とするものを計算する。

計算方法の詳細については“富士通 SSL II 使用手引書”の CBYN の項目を参照のこと。

c_dceig2

複素行列の固有値及び固有ベクトル(QR 法)

```
ierr = c_dceig2(za, k, n, mode, ze, zev, vw,
                ivw, &icon);
```

1. 機能

n 次の複素行列 A の固有値及び対応する固有ベクトルを求める. 固有ベクトルは $\|x\|_2 = 1$ となるように正規化される. $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dceig2((dcomplex *)za, k, n, mode, ze, (dcomplex *)zev, vw, ivw,
                &icon);
```

引数の説明:

za	dcomplex	入力	複素行列 A .
	za[n][k]	出力	演算後, 内容は保存されない.
k	int	入力	配列 za 及び zev の整合寸法 ($k \geq n$).
n	int	入力	複素行列 A の次数 n .
mode	int	入力	平衡化の省略の指定. - mode = 1 のときは, 平衡化を省略する. - mode $\neq$ 1 のときは, 平衡化を行なう. (“使用上の注意”a)参照)
ze	dcomplex	出力	固有値.
	ze[n]		
zev	dcomplex	出力	固有ベクトル.
	zev[n][k]		固有ベクトルは, 固有値と対応する行に格納される.
vw	double	作業領域	
ivw	int	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	$n=1$ であった.	ze[0] = za[0][0] zev[0][0].re = 1 zev[0][0].im = 0 とする.
20000	三角行列に変換できず, 固有値・固有ベクトルは求められなかった.	処理を打ち切る.
30000	次のいずれかであった. $n < 1$ $k < n$	処理を打ち切る.

3. 使用上の注意

a) 平衡化について

複素行列の各要素の大きさが、大きく異なっているような場合には、一般に行列の平衡化によって結果の精度を上げることができる。平衡化を行なうには、`mode ≠ 1` と指定する。

他方、もし行列の各要素の大きさがほぼそろっているような場合には、行列の平衡化による効果を期待できない。このような場合には、`mode = 1` とすることにより、平衡化を省略できる。

4. 使用例

5×5の複素数行列の固有値及び固有ベクトルを求める。

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 5

MAIN__()
{
    int ierr, icon;
    int n, i, j, k, mode, ivw[NMAX];
    dcomplex za[NMAX][NMAX], ze[NMAX], zev[NMAX][NMAX];
    double vw[NMAX];

    /* initialize matrix */
    n = NMAX;
    k = NMAX;
    for (i=0; i<n; i++)
        for (j=0; j<=i; j++) {
            za[i][j].re = n-i;
            za[j][i].re = n-i;
            za[i][j].im = 0;
            za[j][i].im = 0;
        }
    mode = 0;
    /* find eigenvalues and eigenvectors */
    ierr = c_dceig2((dcomplex*)za, k, n, mode,
                   ze, (dcomplex*)zev, vw, ivw, &icon);
    printf("icon = %i\n", icon);
    /* print eigenvalues and eigenvectors */
    for (i=0; i<n; i++) {
        printf("eigenvalue:  {%7.4f, %7.4f}\n", ze[i].re, ze[i].im);
        printf("eigenvector:  ");
        for (j=0; j<n; j++)
            printf("{%7.4f, %7.4f}  ", zev[i][j].re, zev[i][j].im);
        printf("\n");
    }
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”のCEIG2の項目及び[118], [119]を参照のこと。

c_dceli1

第 1 種完全楕円積分 $K(x)$

ierr = c_dceli1(x, &celi, &icon);

1. 機能

第 1 種完全楕円積分 $K(x)$ の値,

$$K(x) = \int_0^{\pi/2} \frac{d\theta}{\sqrt{1-x\sin^2\theta}}$$

を近似式を用いて計算する. ただし, $0 \leq x < 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dceli1(x, &celi, &icon);
```

引数の説明:

x	double	入力	独立変数 x .
celi	double	出力	関数値 $K(x)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
30000	次のいずれかであった. • $x < 0$ • $x \geq 1$	celi = 0 とする.

3. 使用例

$K(x)$ の値を x が 0 から 0.99 まで 0.01 おきに計算し数表を作る.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
    double x, celi;
    int i;

    for (i=0;i<100;i++) {
        x = (double)i/100;
        /* calculate complete elliptic integral */
        ierr = c_dceli1(x, &celi, &icon);
        if (icon == 0)
            printf("x = %4.2f    celi = %f\n", x, celi);
        else
            printf("ERROR: x = %4.2f    celi = %f    icon = %i\n", x, celi, icon);
    }
    return(0);
}
```

4. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の CELI1 の項目及び[48]を参照のこと.

c_dceli2

第 2 種完全楕円積分 $E(x)$

<code>ierr = c_dceli2(x, &celi, &icon);</code>

1. 機能

第 2 種完全楕円積分 $E(x)$ の値

$$E(x) = \int_0^{\pi/2} \sqrt{1 - x \sin^2 \theta} d\theta$$

を近似式を用いて計算する。ただし、 $0 \leq x \leq 1$ であること。

2. 引数

呼出し形式:

```
ierr = c_dceli2(x, &celi, &icon);
```

引数の説明:

x	double	入力	独立変数 x .
celi	double	出力	関数値 $E(x)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	次のいずれかであった. • $x < 0$ • $x > 1$	celi = 0 とする.

3. 使用例

$E(x)$ の値を x が 0 から 0.99 まで 0.01 おきに計算し数表を作る.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
    double x, celi;
    int i;

    for (i=0; i<100; i++) {
        x = (double)i/100;
        /* calculate complete elliptic integral */
        ierr = c_dceli2(x, &celi, &icon);
        if (icon == 0)
            printf("x = %4.2f    celi = %f\n", x, celi);
        else
            printf("ERROR: x = %4.2f    celi = %f    icon = %i\n", x, celi, icon);
    }
    return(0);
}
```

4. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の CELI2 の項目及び[48]を参照のこと.

c_dcfri

余弦フレネル積分 $C(x)$
<code>ierr = c_dcfri(x, &cf, &icon);</code>

1. 機能

余弦フレネル積分 $C(x)$ の値,

$$C(x) = \frac{1}{\sqrt{2\pi}} \int_0^x \frac{\cos(t)}{\sqrt{t}} dt = \int_0^{\sqrt{\frac{2}{\pi}x}} \cos\left(\frac{\pi}{2}t^2\right) dt,$$

を多項式及び漸近形式の近似式を用いて計算する。ただし, $x \geq 0$ であること。

2. 引数

呼出し形式:

```
ierr = c_dcfri(x, &cf, &icon);
```

引数の説明:

x	double	入力	独立変数 x . (“使用上の注意”参照)
cf	double	出力	関数値 $C(x)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
20000	$x \geq t_{\max}$	cf = 0.5.
30000	$x < 0$	cf = 0.

3. 使用上の注意

a) x の範囲

引数 x の範囲は $0 \leq x < t_{\max}$ であること. x が大きくなると $\sin(x)$, $\cos(x)$ の値が正確に計算出来なくなることに對する処理である.

4. 使用例

$C(x)$ の値を x が 0 から 100 まで, 1 おきに計算し数表を作る.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
    double x, cf;
    int i;

    for (i=0;i<=100;i++) {
        x = i;
        /* calculate Cosine Fresnel integral */
        ierr = c_dcfri(x, &cf, &icon);
        if (icon == 0)
            printf("x = %5.2f    cf = %f\n", x, cf);
        else
            printf("ERROR: x = %5.2f    cf = %f    icon = %i\n", x, cf, icon);
    }
}
```

```
    }  
    return(0);  
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の CFRI の項目を参照のこと.

c_dcgsbm

行列格納方法の変換 (一般行列の格納方法→対称バンド行列用圧縮格納法)

ierr = c_dcgsbm(ag, k, n, asb, nh, &icon);

1. 機能

一般行列の格納方法で格納された $n \times n$, バンド幅 h の実対称バンド行列 **A** を, 対称バンド行列用圧縮格納法に変換する. $n > h \geq 0$ であること.

2. 引数

呼出し形式:

```
ierr = c_dcgsbm((double*)ag, k, n, asb, nh, &icon);
```

引数の説明:

ag	double ag[n][k]	入力	一般行列の格納方法で格納された対称バンド行列 A .
k	int	入力	配列 ag の整合寸法 ($\geq n$).
n	int	入力	行列の次数 n .
asb	double asb[Asblen]	出力	対称バンド行列用圧縮格納法で格納された対称バンド行列. $Asblen = n(h+1) - h(h+1) / 2$. 格納法の詳細については“1.3.4 バンド行列の格納方法”を参照のこと.
nh	int	入力	行列のバンド幅 h .
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
30000	次のいずれかであった. - nh < 0 - n ≤ nh - k < n	処理を打ち切る.

3. 使用上の注意

a) 行列の格納方法

配列 ag には, 上バンド部分及び対角部分だけを与えればよい. 下バンド部分は, 本関数内で上バンド部分が複写される.

b) 領域の節約について

配列 ag 上の内容を保存する必要のない場合は, ag と asb に同じ領域を指定すれば領域が節約できる.

4. 使用例

一般行列の格納法で与えられた対称バンド行列を, 対称バンド行列用圧縮格納法に変換する.

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

```
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define min(a,b) ((a) < (b) ? (a) : (b))
#define max(a,b) ((a) > (b) ? (a) : (b))

#define NMAX 5
#define NHMAX 2

/* print symmetric band matrix */
void prtsymbandmat(double a[], int n, int nh)
{
    int ij, i, j, jmin;
    printf("symmetric band matrix format\n");
    ij = 0;
    for (i=0; i<n; i++) {
        jmin = max(i-nh, 0);
        for (j=jmin; j<=i; j++)
            printf("%7.2f ", a[ij++]);
        printf("\n");
    }
}

/* print general matrix */
void prtgenmat(double *a, int k, int n, int m)
{
    int i, j;
    printf("general matrix format\n");
    for (i=0; i<n; i++) {
        for (j=0; j<m; j++)
            printf("%7.2f ", a[i*k+j]);
        printf("\n");
    }
}

MAIN__()
{
    int ierr, icon;
    int n, nh, i, j, k, jmax;
    double asb[NMAX*(NHMAX+1)-NHMAX*(NHMAX+1)/2], ag[NMAX][NMAX];

    /* zero matrix */
    n = NMAX;
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            ag[i][j] = 0;
    /* initialize symmetric band matrix
       in upper half of general matrix storage format */
    nh = NHMAX;
    for (i=0; i<n; i++) {
        jmax = min(i+nh, n-1);
        for (j=i; j<=jmax; j++)
            ag[i][j] = j-i+1;
    }
    k = NMAX;
    /* convert to symmetric band matrix storage format */
    ierr = c_dcgsbm((double*)ag, k, n, asb, nh, &icon);
    if (icon != 0) {
        printf("ERROR: c_dcgsbm failed with icon = %d\n", icon);
        exit(1);
    }
    /* print matrices */
    printf("ag: \n");
    prtgenmat((double*)ag, k, n, n);
    printf("asb: \n");
    prtsymbandmat(asb, n, nh);
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の CGSBM の項目を参照のこと。

c_dcgs

行列格納方法の変換 (一般行列の格納方法→対称行列用圧縮格納法)

```
ierr = c_dcgs(ag, k, n, as, &icon);
```

1. 機能

一般行列の格納方法で格納された $n \times n$ の実対称行列を, 対称行列用圧縮格納方法に変換する. $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dcgs((double*)ag, k, n, as, &icon);
```

引数の説明:

ag	double ag[n][k]	入力	一般行列の格納方法で格納された対称行列 A .
k	int	入力	配列 ag の整合寸法 ($\geq n$).
n	int	入力	行列 A の次数 n .
as	double as[Aslen]	出力	対称行列用圧縮格納法で格納された対称行列 A . $Aslen = n(n+1)/2$. 格納法の詳細については“1.3.2 対称行列の格納方法”を参照のこと.
icon	int	出力	コンディションコード.下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
30000	次のいずれかであった. • $n < 1$ • $k < n$	処理を打ち切る.

3. 使用上の注意

a) 一般行列の格納方法へ対称行列を格納する方法

配列 ag には, 上三角部分及び対角部分だけを与えればよい. 下三角部分は, 関数内において上三角部分が複写される.

b) 領域の節約について

配列 ag 上の内容を保存する必要のない場合は, 引数 as のかわりに同じ配列を指定することにより領域が節約できる. このとき 2 つの配列のサイズが異なるので十分なサイズをとっておくこと.

4. 使用例

一般行列の格納方法で格納された実対称行列を, 対称行列用圧縮格納方法に変換し, 元の行列と変換後の行列を出力する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 5

/* print symmetric matrix */
void prtsymmat(double a[], int n)
{
    int ij, i, j;
    printf("symmetric matrix format\n");
    ij = 0;
    for (i=0;i<n;i++) {
        for (j=0;j<=i;j++)
            printf("%7.2f ", a[ij++]);
        printf("\n");
    }
}

/* print general matrix */
void prtgenmat(double *a, int k, int n, int m)
{
    int i, j;
    printf("general matrix format\n");
    for (i=0;i<n;i++) {
        for (j=0;j<m;j++)
            printf("%7.2f ", a[i*k+j]);
        printf("\n");
    }
}

MAIN__()
{
    int ierr, icon;
    int n, i, j, ij, k;
    double as[NMAX*(NMAX+1)/2], ag[NMAX][NMAX];

    /* initialize general matrix storage format */
    n = NMAX;
    for (i=0;i<n;i++)
        for (j=i;j<n;j++) {
            ag[i][j] = n-i;
        }
    k = NMAX;
    /* convert to symmetric matrix storage format */
    ierr = c_dcgsm((double*)ag, k, n, as, &icon);
    if (icon != 0) {
        printf("ERROR: c_dcgsm failed with icon = %d\n", icon);
        exit(1);
    }
    /* print matrices */
    printf("ag: \n");
    prtgenmat((double*)ag, k, n, n);
    printf("as: \n");
    prtsymmat(as, n);
    return(0);
}
```

5. 手法概要

2次元配列 ag 上に一般行列の格納方法で格納された実対称行列を, 次に示す手順により, 1次元配列上に対称行列用圧縮格納法で格納する.

- 対角部分を対称軸として, 上三角部分を下三角部分に転送する.

$$ag[i][j] = ag[j][i], i < j$$

- ag の対角要素及び下三角要素 $ag[i-1][j-1]$ を, as の $i*(i-1)/2+j-1$ 番目に転送する.

ここで, $i \geq j$ である. 転送は ag の 1 列目から列ごとに行なう.

$NT = n(n+1)/2$ としたとき以下に対応関係を示す.

一般行列の格納方法の要素		行列の要素		圧縮格納法上の要素
$ag[0][0]$	→	a_{11}	→	$as[0]$
$ag[0][1]$	→	a_{12}	→	$as[1]$
$ag[1][1]$	→	a_{22}	→	$as[2]$
$\vdots$		$\vdots$		$\vdots$
$ag[i-1][j-1]$	→	a_{ij}	→	$as[i*(i-1)/2+j-1]$
$\vdots$		$\vdots$		$\vdots$
$ag[n-2][n-1]$	→	a_{n-1n}	→	$as[NT-2]$
$ag[n-1][n-1]$	→	a_{nn}	→	$as[NT-1]$

計算方法の詳細については“富士通 SSL II 使用手引書”の CGSM の項目を参照のこと.

c_dchbk2

複素行列の固有ベクトルへの逆変換

<pre>ierr = c_dchbk2(zev, k, n, ind, m, zp, ip, dv, &icon);</pre>

1. 機能

n 次の複素ヘッセンベルグ行列 $\mathbf{H}$ の m 個の固有ベクトルを, 複素行列 $\mathbf{A}$ の固有ベクトルへ逆変換する. ただし, $\mathbf{H}$ は $\mathbf{A}$ に安定化基本相似変換を適用して得られたものとする. 複素行列 $\mathbf{A}$ の固有ベクトルの正規化は行わない. $1 \leq m \leq n$ であること.

2. 引数

呼出し形式:

```
ierr = c_dchbk2((dcomplex *) zev, k, n, ind, m, (dcomplex *) zp, ip, dv,
                &icon);
```

引数の説明:

zev	dcomplex	入力	複素ヘッセンベルグ行列 $\mathbf{H}$ の m 個の固有ベクトル.
	zev[m][k]	出力	複素行列 $\mathbf{A}$ の m_l 個の固有ベクトル. ここで, m_l は, ind の要素のうち, その値が 1 であるものの個数である.
k	int	入力	配列 zev, zp の整合寸法 ($\geq n$).
n	int	入力	行列 $\mathbf{A}$ と $\mathbf{H}$ の次数 n .
ind	int ind[m]	入力	固有ベクトル逆変換要不要の指定. ind[j-1] = 0 なら, j 番目の固有値に対応する固有ベクトルの逆変換を行わない. ind[j-1] = 1 なら, j 番目の固有値に対応する固有ベクトルの逆変換を行なう.
m	int	入力	複素ヘッセンベルグ行列の全固有ベクトルに対応する固有値の総数 m .
zp	dcomplex zp[n][k]	入力	$\mathbf{A}$ から $\mathbf{H}$ への変換行列のための情報. (“使用上の注意”b)参照)
ip	int ip[n]	入力	$\mathbf{A}$ から $\mathbf{H}$ への変換行列のための情報. (“使用上の注意”b)参照)
dv	double dv[n]	入力	複素行列 $\mathbf{A}$ の平衡化に使われたスケーリングファクタ. 平衡化が行われなかったとき, dv は配列である必要はなく, このとき, dv[0] = 0 と指示できる. (“使用上の注意”c)参照)
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	$n = 1$	zev[0][0] = (1,0).
30000	次のいずれかであった. • $m < 1$ 又は $m > n$ • $k < n$	処理を打ち切る.

3. 使用上の注意

a) zev, ind 及び m

関数 c_dchvec 呼出し後の引数 zev, ind, m は、本関数でそのまま入力できるようになっている。

b) zp 及び ip

関数 c_dches2 の引数 za と ip、本関数の引数 zp と ip に対応しているので、そのまま使用できる。

c) dv

平衡化によるスケーリングファクタ dv の内容については、関数“c_dcbinc”を参照のこと。

4. 使用例

n 次の実行列 **A** を c_dblnc で平衡化した後、関数 c_dches2 により実ヘッセンベルグ行列に変換し、固有値を関数 c_dchsqr で求め、固有ベクトルを c_dchvec 及び c_dchbk2 で求め、c_dcnrm1 で固有ベクトルを正規化する。

```
#include <stdio.h>
#include <stdlib.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 5

MAIN__()
{
    int ierr, icon;
    int n, i, j, k, m, mode, ip[NMAX], ind[NMAX];
    dcomplex za[NMAX][NMAX], ze[NMAX], zev[NMAX][NMAX], zaw[NMAX+1][NMAX];
    double dv[NMAX];

    /* initialize matrix */
    n = NMAX;
    k = NMAX;
    for (i=0; i<n; i++) {
        za[i][i].re = n-i;
        za[i][i].im = 0;
        for (j=0; j<i; j++) {
            za[i][j].re = n-i;
            za[j][i].re = n-i;
            za[i][j].im = 0;
            za[j][i].im = 0;
        }
    }
    /* balance matrix A */
    ierr = c_dcbinc((dcomplex*)za, k, n, dv, &icon);
    if (icon > 10000 ) {
        printf("ERROR: c_dcbinc failed with icon = %i\n", icon);
        exit (1);
    }
    /* reduce matrix to Hessenberg form */
    ierr = c_dches2((dcomplex*)za, k, n, ip, &icon);
    if (icon > 10000 ) {
        printf("ERROR: c_dches2 failed with icon = %i\n", icon);
        exit (1);
    }
    for (i=0; i<n; i++)
        for (j=0; j<n; j++) {
            zaw[i][j].re = za[i][j].re;
            zaw[i][j].im = za[i][j].im;
        }
    /* find eigenvalues */
    ierr = c_dchsqr((dcomplex*)zaw, k, n, ze, &m, &icon);
    if (icon >= 20000 ) {
        printf("ERROR: c_dchsqr failed with icon = %i\n", icon);
        exit (1);
    }
    for (i=0; i<m; i++) ind[i] = 1;
    /* find eigenvectors for given eigenvalues */
    ierr = c_dchvec((dcomplex*)za, k, n, ze,
                    ind, m, (dcomplex*)zev, (dcomplex*)zaw, &icon);
    if (icon >= 20000 ) {
```

```
    printf("ERROR: c_dchvec failed with icon = %i\n", icon);
    exit (1);
}
/* back transformation to find e-vectors of A */
ierr = c_dchbk2((dcomplex*)zev, k, n, ind, m,
               (dcomplex*)za, ip, dv, &icon);
if (icon > 10000 ) {
    printf("ERROR: c_dchbk2 failed with icon = %i\n", icon);
    exit (1);
}
/* normalize e-vectors */
mode = 2;
ierr = c_dcnrml((dcomplex*)zev, k, n, ind, m, mode, &icon);
if (icon > 10000 ) {
    printf("ERROR: c_dcnrml failed with icon = %i\n", icon);
    exit (1);
}
printf("icon = %i\n", icon);
/* print eigenvalues and eigenvectors */
for (i=0;i<m;i++) {
    if (ind[i] != 0) {
        printf("eigenvalue:  %7.4f+i*%7.4f\n", ze[i].re, ze[i].im);
        printf("eigenvector:  ");
        for (j=0;j<n;j++)
            printf("%7.4f+i*%7.4f  ", zev[i][j].re, zev[i][j].im);
        printf("\n");
    }
}
return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の CHBK2 の項目及び[119]を参照のこと.

c_dches2

複素行列の複素ヘッセンベルグ行列への変換 (安定化基本相似変換)

```
ierr = c_dches2(za, k, n, ip, &icon);
```

1. 機能

n 次の複素行列 $\mathbf{A}$ を, 安定化基本相似変換(部分ピボットイングを伴うガウスの消去法)により, 複素ヘッセンベルグ行列 $\mathbf{H}$ へ変換する.

$$\mathbf{H} = \mathbf{S}^{-1} \mathbf{A} \mathbf{S}$$

ただし $\mathbf{S}$ は変換行列である. $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dches2((dcomplex *) za, k, n, ip, &icon);
```

引数の説明:

za	dcomplex	入力	複素行列 $\mathbf{A}$.
	za[n][k]	出力	複素ヘッセンベルグ行列 $\mathbf{H}$ 及び変換行列 $\mathbf{S}$. (“使用上の注意”a)参照)
k	int	入力	配列 za の整合寸法 ($\geq n$).
n	int	入力	行列 $\mathbf{A}$ の次数 n .
ip	int ip[n]	出力	変換行列 $\mathbf{S}$ のための情報. (“使用上の注意”a)参照)
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
10000	$n = 1$ 又は $n = 2$	変換しない.
30000	次のいずれかであった. • $n < 1$ • $k < n$	処理を打ち切る.

3. 使用上の注意

a) za 及び ip

出力された配列 za 及び ip は, 行列 $\mathbf{A}$ の固有ベクトルを求めるために必要となる.

b) 固有値の精度について

固有値の精度は, 複素ヘッセンベルグ行列に変換した時点である程度決定される. そのため, できるだけ精度よく変換を行うことが必要であり, 本関数ではそのための考慮が払われている. しかし, 固有値に非常に大きいものと小さいものがあるとき, 小さい方の固有値は変換の影響を受けやすく, したがって, 小さい固有値を精度よく求めることが困難な場合がある.

4. 使用例

n 次の複素行列 **A** を複素ヘッセンベルグ行列に変換し, 固有値を関数 `c_dchsqr` で求める.

```
#include <stdio.h>
#include <stdlib.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 5

MAIN__()
{
    int ierr, icon;
    int n, m, i, j, k, ip[NMAX];
    dcomplex za[NMAX][NMAX], ze[NMAX];

    /* initialize matrix */
    n = NMAX;
    k = NMAX;
    for (i=0; i<n; i++) {
        za[i][i].re = n-i;
        za[i][i].im = 0;
        for (j=0; j<i; j++) {
            za[i][j].re = n-i;
            za[j][i].re = n-i;
            za[i][j].im = 0;
            za[j][i].im = 0;
        }
    }
    /* reduce matrix to Hessenberg form */
    ierr = c_dches2((dcomplex*)za, k, n, ip, &icon);
    if (icon > 10000 ) {
        printf("ERROR: c_dches2 failed with icon = %i\n", icon);
        exit (1);
    }
    /* find eigenvalues */
    ierr = c_dchsqr((dcomplex*)za, k, n, ze, &m, &icon);
    if (icon >= 20000 ) {
        printf("ERROR: c_dchsqr failed with icon = %i\n", icon);
        exit (1);
    }
    printf("icon = %i\n", icon);
    /* print eigenvalues */
    printf("eigenvalues:\n");
    for (i=0; i<m; i++) {
        printf("%7.4f+i*%7.4f  ", ze[i].re, ze[i].im);
    }
    printf("\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の CHES2 の項目及び[119]を参照のこと.

c_dchsqr

複素ヘッセンベルグ行列の固有値(QR法)

ierr = c_dchsqr(za, k, n, ze, &m, &icon);

1. 機能

n 次の複素ヘッセンベルグ行列 **A** の固有値を, QR 法により求める. ただし $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dchsqr((dcomplex *) za, k, n, ze, &m, &icon);
```

引数の説明:

za	dcomplex	入力	複素ヘッセンベルグ行列 A .
	za[n][k]	出力	演算後, 内容は保存されない.
k	int	入力	配列 za の整合寸法 ($\geq n$).
n	int	入力	複素ヘッセンベルグ行列 A の次数 n .
ze	dcomplex	出力	行列 A の固有値.
	ze[n]		
m	int	出力	求まった固有値の個数.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
10000	$n = 1$.	$ze[0] = za[0][0]$.
15000	固有値のすべてを求めつくすことはできなかった.	m に求まった固有値の個数をセットする. $1 \leq m < n$.
20000	固有値が, 一つも求められなかった.	$m = 0$ とする.
30000	次のいずれかであった. • $n < 1$ • $k < n$	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

通常 c_dhes2 を実行した後, 本関数によって固有値を求める.

固有ベクトルをも必要とする場合は, 本関数を呼び出す前に, 配列 za の内容を他の領域へ移しておくこと.

4. 使用例

n 次の複素行列 **A** 関数 c_dches2 により複素ヘッセンベルグ行列に変換し, 固有値を求める.

```
#include <stdio.h>
#include <stdlib.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 5

MAIN__()
```

```
{
  int ierr, icon;
  int n, m, i, j, k, ip[NMAX];
  dcomplex za[NMAX][NMAX], ze[NMAX];

  /* initialize matrix */
  n = NMAX;
  k = NMAX;
  for (i=0;i<n;i++) {
    za[i][i].re = n-i;
    za[i][i].im = 0;
    for (j=0;j<i;j++) {
      za[i][j].re = n-i;
      za[j][i].re = n-i;
      za[i][j].im = 0;
      za[j][i].im = 0;
    }
  }
  /* reduce matrix to Hessenberg form */
  ierr = c_dches2((dcomplex*)za, k, n, ip, &icon);
  if (icon > 10000 ) {
    printf("ERROR: c_dches2 failed with icon = %i\n", icon);
    exit (1);
  }
  /* find eigenvalues */
  ierr = c_dchsqr((dcomplex*)za, k, n, ze, &m, &icon);
  if (icon >= 20000 ) {
    printf("ERROR: c_dchsqr failed with icon = %i\n", icon);
    exit (1);
  }
  printf("icon = %i\n", icon);
  /* print eigenvalues */
  printf("eigenvalues:\n");
  for (i=0;i<m;i++) {
    printf("%7.4f+i*%7.4f  ", ze[i].re, ze[i].im);
  }
  printf("\n");
  return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の CHSQR の項目及び[118], [119]を参照のこと.

c_dchvec

複素ヘッセンベルグ行列の固有ベクトル(逆反復法)

<pre>ierr = c_dchvec(za, k, n, ze, ind, m, zev, zaw, &icon);</pre>

1. 機能

n 次の複素ヘッセンベルグ行列 $\mathbf{A}$ の指定された固有値 λ_j に対応する固有ベクトル $\mathbf{x}_j$ を, 逆反復法により求める. 固有ベクトルの正規化は行わない. $n \geq 1$ なること.

2. 引数

呼出し形式:

```
ierr = c_dchvec((dcomplex *) za, k, n, ze, ind, m, (dcomplex *) zev,
                (dcomplex *) zaw, &icon);
```

引数の説明:

za	dcomplex za[n][k]	入力	複素ヘッセンベルグ行列 $\mathbf{A}$.
k	int	入力	za, ev, zaw の整合寸法 ($\geq n$).
n	int	入力	行列 $\mathbf{A}$ の次数 n .
ze	dcomplex ze[m]	入力	固有値. $ze[j-1] = \lambda_j, j=1, \dots, m$ のように格納する.
ind	int ind[m]	入力	固有ベクトル要不要の指定. ind[j-1] = 0 のとき, λ_j に対応する固有ベクトルを要求しない. ind[j-1] = 1 のとき, λ_j に対応する固有ベクトルを要求する.
		出力	求められなかった固有ベクトルの情報は消去される. (“使用上の注意”a)参照)
m	int	入力	配列 ze に格納されている固有値の総数 $m (\leq n)$.
zev	dcomplex zev[mk][k]	出力	固有ベクトル. ind の指示に従って求められた固有ベクトルは, zev の第 1 行から, 行ごとに格納される. mk は求める固有ベクトルの本数.
zaw	dcomplex zaw[n+1][k]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	$n = 1$	zev[0][0] = (1,0).
15000	ある固有値に対応する固有ベクトルが求められなかった.	求められなかった固有ベクトルの ind の情報は消去される.
20000	すべての固有ベクトルが求められなかった.	ind の情報はすべて消去される.

コード	意味	処理内容
30000	次のいずれかであった。 • $m < 1$ 又は $m > n$ • $k < n$	処理を打ち切る.

3. 使用上の注意

a) **ind** 及び **mk**

求められた固有ベクトルの本数 mk は、演算後における **ind** の要素のうち、その値が 1 であるものの個数である。ところで、本関数は、ある固有ベクトルが求められなかったとき、その固有ベクトルの **ind** の情報を消去しているので、 mk は演算前に指定した求めたい固有ベクトルの本数以下である。

b) 本関数の使用方法

本関数で用いる固有値は、関数 `c_dchsqr` を用いて求めることができる。関数 `c_dchsqr` で固有値を求めたときは、`c_dchsqr` の引数 ze, m が、そのまま本関数の引数として使用できる。

複素行列の固有ベクトルを部分的に求める場合は、

- まず、関数 `c_dches2` により複素行列を複素ヘッセンベルグ行列に変換し、
- 次に、関数 `c_dchsqr` により複素行列の固有値を求め、
- 本関数により複素ヘッセンベルグ行列の固有ベクトルを求め、
- 最後に関数 `c_dchbk2` により元の複素行列の固有ベクトルへ逆変換すること。

なお、複素行列の固有値固有ベクトルをすべて求めるときは、関数 `c_dceig2` を用いた方がよい。

本関数は、上記の手順で複素行列の固有ベクトルを部分的に求める場合にも用いることを前提としている。したがって、固有ベクトルの正規化は行わない。正規化が必要な場合は関数 `c_dcnrml` で行うことができる。

関数 `c_dchbk2` や `c_dcnrml` を使用するときは、本関数の引数 **ind**, m , **zev** はそのまま `c_dchbk2` や `c_dcnrml` の入力引数として使用できる。

4. 使用例

n 次の実行列 **A** を `c_dblnc` で平衡化した後、関数 `c_dches2` により実ヘッセンベルグ行列に変換し、固有値を関数 `c_dchsqr` で求め、固有ベクトルを `c_dchvec` 及び `c_dchbk2` で求め、`c_dcnrml` で固有ベクトルを正規化する。

```
#include <stdio.h>
#include <stdlib.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 5

MAIN__()
{
    int ierr, icon;
    int n, i, j, k, m, mode, ip[NMAX], ind[NMAX];
    dcomplex za[NMAX][NMAX], ze[NMAX], zev[NMAX][NMAX], zaw[NMAX+1][NMAX];
    double dv[NMAX];

    /* initialize matrix */
    n = NMAX;
    k = NMAX;
    for (i=0; i<n; i++) {
        za[i][i].re = n-i;
        za[i][i].im = 0;
        for (j=0; j<i; j++) {
            za[i][j].re = n-i;
            za[j][i].re = n-i;
            za[i][j].im = 0;
            za[j][i].im = 0;
        }
    }
}
```

```

/* balance matrix A */
ierr = c_dcbinc((dcomplex*)za, k, n, dv, &icon);
if (icon > 10000 ) {
    printf("ERROR: c_dcbinc failed with icon = %i\n", icon);
    exit (1);
}
/* reduce matrix to Hessenberg form */
ierr = c_dches2((dcomplex*)za, k, n, ip, &icon);
if (icon > 10000 ) {
    printf("ERROR: c_dches2 failed with icon = %i\n", icon);
    exit (1);
}
for (i=0;i<n;i++)
    for (j=0;j<n;j++) {
        zaw[i][j].re = za[i][j].re;
        zaw[i][j].im = za[i][j].im;
    }
/* find eigenvalues */
ierr = c_dchsqr((dcomplex*)zaw, k, n, ze, &m, &icon);
if (icon >= 20000 ) {
    printf("ERROR: c_dchsqr failed with icon = %i\n", icon);
    exit (1);
}
for (i=0;i<m;i++) ind[i] = 1;
/* find eigenvectors for given eigenvalues */
ierr = c_dchvec((dcomplex*)za, k, n, ze,
                ind, m, (dcomplex*)zev, (dcomplex*)zaw, &icon);
if (icon >= 20000 ) {
    printf("ERROR: c_dchvec failed with icon = %i\n", icon);
    exit (1);
}
/* back transformation to find e-vectors of A */
ierr = c_dchbk2((dcomplex*)zev, k, n, ind, m,
                (dcomplex*)za, ip, dv, &icon);
if (icon > 10000 ) {
    printf("ERROR: c_dchbk2 failed with icon = %i\n", icon);
    exit (1);
}
/* normalize e-vectors */
mode = 2;
ierr = c_dcnrml((dcomplex*)zev, k, n, ind, m, mode, &icon);
if (icon > 10000 ) {
    printf("ERROR: c_dcnrml failed with icon = %i\n", icon);
    exit (1);
}
printf("icon = %i\n", icon);
/* print eigenvalues and eigenvectors */
for (i=0;i<m;i++) {
    if (ind[i] != 0) {
        printf("eigenvalue:  %7.4f+i*%7.4f\n", ze[i].re, ze[i].im);
        printf("eigenvector:  ");
        for (j=0;j<n;j++)
            printf("%7.4f+i*%7.4f  ", zev[i][j].re, zev[i][j].im);
        printf("\n");
    }
}
return(0);
}

```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の CHVEC の項目及び[119]を参照のこと。

c_dcjart

複素係数高次代数方程式 (ヤラット法)

ierr = c_dcjart(za, &n, z, &icon);

1. 機能

複素係数高次代数方程式,

$$a_0 z^n + a_1 z^{n-1} + \dots + a_n = 0$$

の根をヤラット法により求める. a_i は複素数であり $|a_0| \neq 0, n \geq 1$.

2. 引数

呼出し形式:

```
ierr = c_dcjart(za, &n, z, &icon);
```

引数の説明:

za	dcomplex	入力	代数方程式の係数. $za[i]=a_i$ ($i=0, \dots, n$).
	za[n+1]		演算後内容は保存されない.
n	int	入力	代数方程式の次数 n .
		出力	求まった根の個数 (使用上の注意 b)参照).
z	dcomplex	出力	n 個の根. 根は求まった順に $z[0], \dots, z[n-1]$ に出力される.
	z[n]		
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	n 根を, すべては求めつくすことができなかった.	求まった根の個数を引数 n に, そして, 根を $z[0], \dots, z[n-1]$ に出力する.
30000	次のいずれかであった. $n < 1$ $a_0 = 0$	処理を打ち切る.

3. 使用上の注意

a) 計算手法について

n が 1 又は 2 の場合は, ヤラット法を使わず, 根の公式を使って計算している.

b) 求まった根について

n 次代数方程式は n 個の根を持つが, ごくまれに, 必ずしもそのすべてを求めることができないことがある. 利用者はこのことに注意され, 引数 icon あるいは n の演算後の値を確認すること.

4. 使用例

方程式 $z^3 - 6z^2 + 11z - 6 = 0$ の解を求める.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

#define N 3

MAIN__()
{
    int ierr, icon;
    dcomplex z[N];
    dcomplex za[] = {{1, 0},
                     {-6, 0},
                     {11, 0},
                     {-6, 0}};

    int n, i;

    /* initialize data */
    n = N;
    /* find roots of polynomial */
    ierr = c_dcjart(za, &n, z, &icon);
    printf("icon = %i    n = %i\n", icon, n);
    for (i=0; i<n; i++)
        printf("z[%i] = {%12.4e, %12.4e}\n", i, z[i].re, z[i].im);
    printf("exact roots are: {1, 0}, {2, 0} and {3, 0}\n");
    return(0);
}
```

5. 手法概要

本関数が適用している手法は、反復解法的一种である Garside-Jarratt-Mack の方法を、多少修正したものである。

計算方法の詳細については“富士通 SSL II 使用手引書”の CJART の項目及び[38], [132]を参照のこと。

c_dclu

複素行列の LU 分解 (クラウト法)

<pre>ierr = c_dclu(za, k, n, epsz, ip, &is, zvw, &icon);</pre>

1. 機能

$n \times n$ の正則な複素行列 **A** をクラウト法により LU 分解する.

$$\mathbf{PA} = \mathbf{LU}$$

ただし, **P** は部分ピボットによる行の入れ換えを行う置換行列, **L** は下三角行列, **U** は単位上三角行列である. $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dclu((dcomplex*)za, k, n, epsz, ip, &is, zvw, &icon);
```

引数の説明:

za	dcomplex za[n][k]	入力	行列 A .
		出力	行列 L と行列 U . (“使用上の注意” c)参照).
k	int	入力	配列 za の整合寸法 ($\geq n$).
n	int	入力	行列 A の次数 n .
epsz	double	入力	ピボットの相対零判定値 (≥ 0.0). 0.0 のときは標準値が採用される. (“使用上の注意” a)参照).
ip	int ip[n]	出力	部分ピボットによる行の入れ換えの履歴を示すトランス ポジションベクトル. (“使用上の注意” b)参照).
is	int	出力	行列 A の行列式を求めるための情報. 演算後の配列 za の n 個の対角要素と is の値を掛け合わせると行列式が得られる.
zvw	dcomplex zvw[n]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	行列 A のある行の要素がすべて零であ ったか又はピボットが相対的に零とな った. 行列 A は非正則の可能性が強い.	処理を打ち切る.
30000	次のいずれかであった. • $k < n$ • $n < 1$ • $epsz < 0$	処理を打ち切る.

3. 使用上の注意

a) epsz について

ピボットの相対零判定値 epsz に 10^{-6} を設定したとすると、この値は次の意味を持っている。すなわち、クラウト法による LU 分解の過程でピボットの実部と虚部の値が、同時に 10 進 s 桁以上の桁落ちを生じた場合にそのピボットを相対的に零とみなし、 $\text{icon} = 20000$ として処理を打ち切る。

epsz の標準値は丸め誤差の単位を μ としたとき、 $\text{epsz} = 16\mu$ である。

なお、ピボットが小さくなっても計算を続行させる場合には、 epsz へ極小の値を与えればよいが、その結果は保証されない。

b) ip について

トランスポジションベクトルとは、部分ピボットングを行う LU 分解

$$\mathbf{PA} = \mathbf{LU}$$

における置換行列に相当する。

本関数では、部分ピボットングに伴い配列 $\mathbf{za}$ の内容を実際に交換している。すなわち、分解の J 段階目 ($J = 1, \dots, n$) において第 I 行がピボット行として選択された場合には、配列 $\mathbf{za}$ の第 I 行と第 J 行の内容が交換される。そして、その履歴を示すために $\text{ip}[J-1]$ に I が格納される。

c) 逆行列

本関数に続けて、関数 `c_dcluiv` を呼び出すことにより、逆行列を計算することができる。

4. 使用例

行列 $\mathbf{A}$ の逆行列 $\mathbf{A}^{-1}$ を求める。結果の確認は行列 $\mathbf{A}$ とベクトル $\mathbf{x}$ の積 $\mathbf{b}$ から、 $\mathbf{A}^{-1}\mathbf{b}$ を計算し、その結果を $\mathbf{x}$ と比較している。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 100

MAIN__()
{
    int ierr, icon;
    int n, i, j, k, is;
    double epsz, eps;
    dcomplex za[NMAX][NMAX];
    dcomplex zb[NMAX], zx[NMAX], zy[NMAX], zvw[NMAX];
    int ip[NMAX];

    /* initialize matrix and vector */
    n = NMAX;
    k = NMAX;
    for (i=0; i<n; i++) {
        for (j=i; j<n; j++) {
            za[i][j].re = n-j;
            za[i][j].im = n-j;
            za[j][i].re = n-j;
            za[j][i].im = n-j;
        }
        zx[i].re = i+1;
        zx[i].im = i+1;
    }
    /* initialize constant vector zb = za*zx */
    ierr = c_dmcv((dcomplex*)za, k, n, n, zx, zb, &icon);
    epsz = 1e-6;
    /* perform LU decomposition */
}
```

```
ierr = c_dclu((dcomplex*)za, k, n, epsz, ip, &is, zvw, &icon);
if (icon != 0) {
    printf("ERROR: c_dvclu failed with icon = %d\n", icon);
    exit(1);
}
/* find matrix inverse from LU factors */
ierr = c_dcluiv((dcomplex*)za, k, n, ip, &icon);
if (icon != 0) {
    printf("ERROR: c_dcluiv failed with icon = %d\n", icon);
    exit(1);
}
/* calculate zy = za*zv */
ierr = c_dmcv((dcomplex*)za, k, n, n, zv, zy, &icon);
/* compare zx and zy */
eps = 1e-6;
for (i=0;i<n;i++)
    if (fabs((zy[i].re-zx[i].re)/zx[i].re) > eps ||
        fabs((zy[i].im-zx[i].im)/zx[i].im) > eps) {
        printf("WARNING: result inaccurate\n");
        exit(1);
    }
printf("Result OK\n");
return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の CLU の項目及び[7], [34], [83]を参照のこと.

c_dcluiv

LU 分解された複素行列の逆行列

ierr = c_dcluiv(zfa, k, n, ip, &icon);

1. 機能

LU 分解された $n \times n$ の複素行列 $\mathbf{A}$ の逆行列 $\mathbf{A}^{-1}$ を求める.

$$\mathbf{A}^{-1} = \mathbf{U}^{-1} \mathbf{L}^{-1} \mathbf{P} \quad (1)$$

ただし, $\mathbf{L}$, $\mathbf{U}$ はそれぞれ, $n \times n$ の下三角行列と単位上三角行列であり, $\mathbf{P}$ は LU 分解におけるピボットティングによる行の入換えを示す置換行列である. $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dcluiv((dcomplex*)zfa, k, n, ip, &icon);
```

引数の説明:

zfa	dcomplex zfa[n][k]	入力	行列 $\mathbf{L}$ と行列 $\mathbf{U}$. (“使用上の注意” a)参照). c_dclu の出力配列をそのまま使う.
		出力	逆行列 $\mathbf{A}^{-1}$.
k	int	入力	配列 zfa の整合寸法 ($\geq n$).
n	int	入力	行列 $\mathbf{L}$, $\mathbf{U}$ の次数 n .
ip	int ip[n]	入力	ピボットティングによる行の入換えの履歴を示すトランスポジションベクトル. (“使用上の注意” b)参照).
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
20000	複素行列が非正則であった.	処理を打ち切る.
30000	次のいずれかであった. • $k < n$ • $n < 1$ • ip に誤りがあった.	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

本関数は, LU 分解された複素行列を入力して, 分解された元の複素行列の逆行列を計算する. 分解は関数 c_dclu により行い, 続けて本関数を呼び出すことにより逆行列を求めることができる.

連立一次方程式を解く場合には, 関数 c_dlcx を用いること. 逆行列を求めて解くことは, c_dlcx を用いるときよりも演算回数が多くなるので避けた方がよい. 本関数は逆行列が必要なときだけ使用すること.

b) ip について

トランスポジションベクトルとは、部分ピボットを行う LU 分解

$$\mathbf{PA} = \mathbf{LU}$$

における置換行列 $\mathbf{P}$ に相当する。“c_dclu”の“使用上の注意”を参照すること。

4. 使用例

行列 $\mathbf{A}$ の逆行列 $\mathbf{A}^{-1}$ を求める。結果の確認は行列 $\mathbf{A}$ とベクトル $\mathbf{x}$ の積 $\mathbf{b}$ から、 $\mathbf{A}^{-1}\mathbf{b}$ を計算し、その結果を $\mathbf{x}$ と比較している。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 100

MAIN__()
{
    int ierr, icon;
    int n, i, j, k, is;
    double epsz, eps;
    dcomplex za[NMAX][NMAX];
    dcomplex zb[NMAX], zx[NMAX], zy[NMAX], zvw[NMAX];
    int ip[NMAX];

    /* initialize matrix and vector */
    n = NMAX;
    k = NMAX;
    for (i=0; i<n; i++) {
        for (j=i; j<n; j++) {
            za[i][j].re = n-j;
            za[i][j].im = n-j;
            za[j][i].re = n-j;
            za[j][i].im = n-j;
        }
        zx[i].re = i+1;
        zx[i].im = i+1;
    }
    /* initialize constant vector zb = za*zx */
    ierr = c_dmcv((dcomplex*)za, k, n, n, zx, zb, &icon);
    epsz = 1e-6;
    /* perform LU decomposition */
    ierr = c_dclu((dcomplex*)za, k, n, epsz, ip, &is, zvw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dvclu failed with icon = %d\n", icon);
        exit(1);
    }
    /* find matrix inverse from LU factors */
    ierr = c_dcluiv((dcomplex*)za, k, n, ip, &icon);
    if (icon != 0) {
        printf("ERROR: c_dcluiv failed with icon = %d\n", icon);
        exit(1);
    }
    /* calculate zy = za*zb */
    ierr = c_dmcv((dcomplex*)za, k, n, n, zb, zy, &icon);
    /* compare zx and zy */
    eps = 1e-6;
    for (i=0; i<n; i++)
        if (fabs((zy[i].re-zx[i].re)/zx[i].re) > eps ||
            fabs((zy[i].im-zx[i].im)/zx[i].im) > eps) {
            printf("WARNING: result inaccurate\n");
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の CLUIV の項目及び[34]を参照のこと.

c_dclux

LU 分解された複素行列の連立 1 次方程式

<code>ierr = c_dclux(zb, zfa, k, n, isw, ip, &icon);</code>

1. 機能

LU 分解された複素係数行列を持つ連立 1 次方程式

$$\mathbf{LUx} = \mathbf{Pb} \quad (1)$$

を解く. ただし, $\mathbf{L}$, $\mathbf{U}$ はそれぞれ $n \times n$ の下三角行列と単位上三角行列, $\mathbf{P}$ は置換行列 (係数行列を LU 分解したときの部分ピボットイングによる行の入換えを行う), $\mathbf{b}$ は n 次元の複素定数ベクトル, $\mathbf{x}$ は n 次元の解ベクトルである. なお, 方程式 (1) の代わりに

$$\mathbf{Ly} = \mathbf{Pb} \quad (2)$$

$$\mathbf{Uz} = \mathbf{b} \quad (3)$$

のいずれかを解くこともできる. $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dclux(zb, (dcomplex*)zfa, k, n, isw, ip, &icon);
```

引数の説明:

zb	dcomplex	入力	定数ベクトル $\mathbf{b}$.
	zb[n]	出力	解ベクトル $\mathbf{x}, \mathbf{y}, \mathbf{z}$ のうちいずれか.
zfa	dcomplex	入力	行列 $\mathbf{L} + (\mathbf{U} - \mathbf{I})$.
	zfa[n][k]		
k	int	入力	配列 zfa の整合寸法 ($\geq n$).
n	int	入力	行列 $\mathbf{L}, \mathbf{U}$ の次数 n .
isw	int	入力	制御情報.
			• isw=1 のとき, (1) の $\mathbf{x}$ を求める. • isw=2 のとき, (2) の $\mathbf{y}$ を求める. • isw=3 のとき, (3) の $\mathbf{z}$ を求める.
ip	int ip[n]	入力	部分ピボットイングによる行の入換えの履歴を示すトランスポジションベクトル.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	係数行列が非正則であった.	処理を打ち切る.
30000	次のいずれかであった. • $n < 1$ • $k < n$ • $isw \neq 1, 2, 3$ • ip に誤りがあった	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

連立 1 次方程式を解く場合、関数 `c_dclu` を呼び出して、係数行列を LU 分解させてから、本関数を呼び出せば方程式を解くことができる。しかし、通常は関数 `c_dlcx` を呼び出せば、一度に解が求められる。

4. 使用例

複素係数行列を LU 分解させてから、複素係数行列連立 1 次方程式を解く。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 100

MAIN__()
{
    int ierr, icon;
    int n, i, j, k, is, isw;
    double epsz, eps;
    dcomplex zfa[NMAX][NMAX];
    dcomplex zb[NMAX], zx[NMAX], zvw[NMAX];
    int ip[NMAX];

    /* initialize matrix and vector */
    n = NMAX;
    k = NMAX;
    for (i=0; i<n; i++) {
        for (j=i; j<n; j++) {
            zfa[i][j].re = n-j;
            zfa[i][j].im = n-j;
            zfa[j][i].re = n-j;
            zfa[j][i].im = n-j;
        }
        zx[i].re = i+1;
        zx[i].im = i+1;
    }
    /* initialize constant vector zb = za*zx */
    ierr = c_dmcv((dcomplex*)zfa, k, n, n, zx, zb, &icon);
    epsz = 1e-6;
    /* perform LU decomposition */
    ierr = c_dclu((dcomplex*)zfa, k, n, epsz, ip, &is, zvw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dclu failed with icon = %d\n", icon);
        exit(1);
    }
    isw = 1;
    /* solve system of equations using LU factors */
    ierr = c_dclux(zb, (dcomplex*)zfa, k, n, isw, ip, &icon);
    if (icon != 0) {
        printf("ERROR: c_dclux failed with icon = %d\n", icon);
        exit(1);
    }
    /* check result */
    eps = 1e-6;
    for (i=0; i<n; i++)
        if (fabs((zb[i].re-zx[i].re)/zx[i].re) > eps ||
            fabs((zb[i].im-zx[i].im)/zx[i].im) > eps) {
            printf("WARNING: result inaccurate\n");
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の CLUX の項目及び[7], [34], [83]を参照のこと。

c_dcnrml

複素行列の固有ベクトルの正規化

ierr = c_dcnrml(zev, k, n, ind, m, mode, &icon);

1. 機能

n 次の複素行列の m 個の固有ベクトル $\mathbf{x}_j$ ($j=1,2,\dots,m$) が与えられたとき, (1) 又は (2) の正規化を行いベクトル $\mathbf{y}_j$ を求める.

$$\mathbf{y}_j = \mathbf{x}_j / \|\mathbf{x}_j\|_{\infty} \quad (1)$$

$$\mathbf{y}_j = \mathbf{x}_j / \|\mathbf{x}_j\|_2 \quad (2)$$

ただし, $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dcnrml((dcomplex *) zev, k, n, ind, m, mode, &icon);
```

引数の説明:

zev	dcomplex zev[m][k]	入力	m 個の固有ベクトル $\mathbf{x}_j$ ($j=1,\dots,m$). ベクトル $\mathbf{x}_j$ は, zev の第 1 行から, 行ごとに格納すること. (“使用上の注意”参照).
		出力	正規化された m 個の固有ベクトル $\mathbf{y}_j$. ベクトル $\mathbf{y}_j$ は, 対応するベクトル $\mathbf{x}_j$ 上に格納される.
k	int	入力	配列 zev の整合寸法 ($\geq n$).
n	int	入力	複素行列の次数 n .
ind	int ind[m]	入力	正規化を行う固有ベクトルの指定. ind[j-1] = 0 のときは, j 番目の固有値に対応する固有ベクトルの正規化を行わない ind[j-1] = 1 のときは, j 番目の固有値に対応する固有ベクトルの正規化を行う. (“使用上の注意”参照)
m	int	入力	固有ベクトルの大きさ. (“使用上の注意”参照)
mode	int	入力	正規化の方法の指示: mode = 1 のときは, (1) の正規化を行う. mode = 2 のときは, (2) の正規化を行う.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	$n = 1$	zev[0][0] = (1,0).
30000	次のいずれかであった. $m < 1$ 又は $m > n$ $k < n$ - mode $\neq 1$ 又は 2 - ind に誤りがあった.	処理を打ち切る.

3. 使用上の注意

a) 引数 **zev**, **ind** 及び **m** について

関数 `c_dchvec` あるいは `c_dchbk2` を呼び出した後の引数 `zev`, `ind`, `m` は, 本関数でそのまま入力できるようになっている。

4. 使用例

関数 `c_dceig2` により求めた n 次の複素行列の固有ベクトルを, 本関数により $\|x\|_{\infty} = 1$ となるように正規化しなおす。

```
#include <stdio.h>
#include <stdlib.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 5

MAIN__()
{
    int ierr, icon;
    int n, i, j, k, mode, m, ind[NMAX], ivw[NMAX];
    dcomplex za[NMAX][NMAX], ze[NMAX], zev[NMAX][NMAX];
    double vw[NMAX];

    /* initialize matrix */
    n = NMAX;
    k = NMAX;
    for (i=0; i<n; i++) {
        za[i][i].re = n-i;
        za[i][i].im = 0;
        for (j=0; j<i; j++) {
            za[i][j].re = n-i;
            za[j][i].re = n-i;
            za[i][j].im = 0;
            za[j][i].im = 0;
        }
    }
    mode = 2;
    /* find eigenvalues and eigenvectors */
    ierr = c_dceig2((dcomplex*)za, k, n, mode,
                   ze, (dcomplex*)zev, vw, ivw, &icon);
    /* initialize ind array */
    m = n;
    for (i=0; i<m; i++) ind[i] = 1;
    mode = 1;
    /* normalize eigenvectors */
    ierr = c_dcnrml((dcomplex*)zev, k, n, ind, m, mode, &icon);
    if (icon > 10000 ) {
        printf("ERROR: c_dcnrml failed with icon = %i\n", icon);
        exit (1);
    }
    printf("icon = %i\n", icon);
    /* print eigenvalues and eigenvectors */
    for (i=0; i<n; i++) {
        printf("eigenvalue:  %7.4f+i*%7.4f\n", ze[i].re, ze[i].im);
        printf("eigenvector:  ");
        for (j=0; j<n; j++)
            printf("%7.4f+i*%7.4f  ", zev[i][j].re, zev[i][j].im);
        printf("\n");
    }
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の CNRML の項目を参照のこと。

c_dcosi

余弦積分 $C_i(x)$
<code>ierr = c_dcosi(x, &ci, &icon);</code>

1. 機能

余弦積分 $C_i(x)$ の値

$$C_i(x) = -\int_x^\infty \frac{\cos(t)}{t} dt$$

を近似多項式を用いて計算する. ただし $x \neq 0$ であること. $x < 0$ のときは, 上の積分は主値をとるものとする.

2. 引数

呼出し形式:

```
ierr = c_dcosi(x, &ci, &icon);
```

引数の説明:

x	double	入力	独立変数 x. (“使用上の注意”参照)
ci	double	出力	関数値 $C_i(x)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	$ x \geq t_{\max}$	ci = 0 とする.
30000	x = 0	ci = 0 とする.

3. 使用上の注意

a) 引数 x の範囲

引数 x の範囲は $|x| < t_{\max}$ であること. $|x|$ が大きくなると $\sin(x)$, $\cos(x)$ の値が正確に計算できなくなることに
対する処理である. $t_{\max}$ については “1.5 計算機定数” を参照のこと.

4. 使用例

$C_i(x)$ の値を x が 0.1 から 10.0 まで, 0.1 おきに計算し, 数表を作る.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
    double x, ci;
    int i;

    for (i=1; i<=100; i++) {
        x = (double)i/10;
        /* calculate integral */
        ierr = c_dcosi(x, &ci, &icon);
        if (icon == 0)
            printf("x = %5.2f    ci = %f\n", x, ci);
        else
            printf("ERROR: x = %5.2f    ci = %f    icon = %i\n", x, ci, icon);
    }
}
```



```
    return(0);  
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の COSI の項目を参照のこと。

c_dcqdr

複素係数 2 次方程式

ierr = c_dcqdr(z0, z1, z2, z, &icon);

1. 機能

複素係数 2 次方程式の根を求める。

$$a_0 z^2 + a_1 z + a_2 = 0 \quad (|a_0| \neq 0) \quad (1)$$

2. 引数

呼出し形式:

```
ierr = c_dcqdr(z0, z1, z2, z, &icon);
```

呼び出し形式:

z0	dcomplex	入力	2 次方程式の係数 a_0 .
z1	dcomplex	入力	2 次方程式の係数 a_1 .
z2	dcomplex	入力	2 次方程式の係数 a_2 .
z	dcomplex	出力	2 次方程式の根.
	z[2]		
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
10000	$ a_0 = 0$ であった.	$z[0]$ には $-a_2/a_1$ を入れる. $z[1]$ は保証しない.
30000	$ a_0 = 0$ かつ $ a_1 = 0$ であった.	処理を打ち切る.

3. 使用例

2 次方程式 $z^2 - 5z + 6 = 0$ の根を求める.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
    dcomplex z[2];
    dcomplex z0 = {1, 0};
    dcomplex z1 = {-5, 0};
    dcomplex z2 = {6, 0};

    /* find roots of quadratic */
    ierr = c_dcqdr(z0, z1, z2, z, &icon);
    printf("icon = %i    z[0] = {%12.4e, %12.4e}    z[1] = {%12.4e, %12.4e}\n",
           icon, z[0].re, z[0].im, z[1].re, z[1].im);
    printf("exact roots are: {3, 0} and {2, 0}\n");
    return(0);
}
```

4. 手法概要

2 次方程式 (1) の根は, 根の公式を用いて求める.

計算方法の詳細については “富士通 SSL II 使用手引書” の CQDR の項目を参照のこと.

c_dcsbgm

行列格納方法の変換 (対称バンド行列用圧縮格納法→一般行列の格納方法)

ierr = c_dcsbgm (asb, n, nh, ag, k, &icon);

1. 機能

対称バンド行列用圧縮格納法で格納された $n \times n$, バンド幅 h の実対称バンド行列を一般行列の格納方法に変換する. $n > h \geq 0$ であること.

2. 引数

呼出し形式:

```
ierr = c_dcsbgm(asb, n, nh, (double*)ag, k, &icon);
```

引数の説明:

asb	double asb[Asblen]	入力	対称バンド行列用圧縮格納法で格納された対称バンド行列 A . $Asblen = n(h+1) - h(h+1)/2$. 格納法の詳細については“1.3.4 バンド行列の格納方法”を参照のこと.
n	int	入力	行列 A の次数 n .
nh	int	入力	行列 A のバンド幅 h .
ag	double ag[n][k]	出力	一般行列の格納方法で格納された対称バンド行列 A .
k	int	入力	配列 ag の整合寸法 ($\geq n$).
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
30000	次のいずれかであった. - nh < 0 - n ≤ nh - k < n	処理を打ち切る.

3. 使用上の注意

a) 一般行列の格納方法で格納された対称バンド行列の格納方法

本関数により変換された一般行列の格納方法で格納された対称バンド行列とは, 下バンド部分, 対角部分だけではなく, 上バンド部分やその他 (零要素) も格納されている.

b) 領域の節約について

配列 asb 上の内容を保存する必要のない場合は, 引数 ag のかわりに asb を指定することにより領域が節約できる. このとき引数 ag のほうが大きなサイズを必要とするので, 十分なサイズをとっておくこと.

4. 使用例

実対称バンド行列が圧縮格納法で与えられたとき, これを一般行列の格納法に変換する.

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

```

#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define max(a,b) ((a) > (b) ? (a) : (b))

#define NMAX 5
#define NHMAX 2

/* print symmetric band matrix */
void prtsymbandmat(double a[], int n, int nh)
{
    int ij, i, j, jmin;
    printf("symmetric band matrix format\n");
    ij = 0;
    for (i=0;i<n;i++) {
        jmin = max(i-nh, 0);
        for (j=jmin;j<=i;j++)
            printf("%7.2f ",a[ij++]);
        printf("\n");
    }
}

/* print general matrix */
void prtgenmat(double *a, int k, int n, int m)
{
    int i, j;
    printf("general matrix format\n");
    for (i=0;i<n;i++) {
        for (j=0;j<m;j++)
            printf("%7.2f ",a[i*k+j]);
        printf("\n");
    }
}

MAIN__()
{
    int ierr, icon;
    int n, nh, i, j, ij, k, jmin;
    double asb[NMAX*(NHMAX+1)-NHMAX*(NHMAX+1)/2], ag[NMAX][NMAX];

    n = NMAX;
    /* initialize symmetric band matrix */
    nh = NHMAX;
    ij = 0;
    for (i=0;i<n;i++) {
        jmin = max(i-nh, 0);
        for (j=jmin;j<=i;j++)
            asb[ij++] = i-j+1;
    }
    k = NMAX;
    /* convert to general matrix storage format */
    ierr = c_dcsbgm(asb, n, nh, (double*)ag, k, &icon);
    if (icon != 0) {
        printf("ERROR: c_dcsbgm failed with icon = %d\n", icon);
        exit(1);
    }
    /* print matrices */
    printf("asb: \n");
    prtsymbandmat(asb, n, nh);
    printf("ag: \n");
    prtgenmat((double*)ag, k, n, n);
    return(0);
}

```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”のCSBGMの項目を参照のこと。

c_dcsbsm

行列格納方法の変換 (対称バンド行列用圧縮格納法→対称行列用圧縮格納法)

ierr = c_dcsbsm(asb, n, nh, as, &icon);

1. 機能

対称バンド行列用圧縮格納法で格納された $n \times n$ バンド幅 h の対称バンド行列を対称行列用圧縮格納法に変換する. $n > h \geq 0$ であること.

2. 引数

呼出し形式:

```
ierr = c_dcsbsm(asb, n, nh, as, &icon);
```

引数の説明:

asb	double asb[Asblen]	入力	対称バンド行列用圧縮格納法で格納された対称バンド行列 A . $Asblen = n(h+1) - h(h+1) / 2$. 格納法の詳細については“1.3.4 バンド行列の格納方法”を参照のこと.
n	int	入力	行列 A の次数 n .
nh	int	入力	行列 A のバンド幅 h .
as	double as[Aslen]	出力	対称行列用圧縮格納法で格納された対称バンド行列 A . $Aslen = n(n+1) / 2$. 格納法の詳細については“1.3.2 対称行列の格納方法”を参照のこと.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
30000	次のいずれかであった. • $nh < 0$ • $n \leq nh$	処理を打ち切る.

3. 使用上の注意

a) 領域の節約について

配列 asb 上の内容を保存する必要のない場合は, 引数 as のかわりに asb を指定することにより領域が節約できる. このとき引数 as のほうが大きなサイズを必要とするので, 十分なサイズをとっておくこと.

4. 使用例

正値対称バンド行列が対称バンド行列用圧縮格納法で与えられたとき, これを対称行列用圧縮格納法に変換する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */
```

```

#define max(a,b) ((a) > (b) ? (a) : (b))

#define NMAX 5
#define NHMAX 2

/* print symmetric matrix */
void prtsymmat(double a[], int n)
{
    int ij, i, j;
    printf("symmetric matrix format\n");
    ij = 0;
    for (i=0;i<n;i++) {
        for (j=0;j<=i;j++)
            printf("%7.2f ",a[ij++]);
        printf("\n");
    }
}

/* print symmetric band matrix */
void prtsymbandmat(double a[], int n, int nh)
{
    int ij, i, j, jmin;
    printf("symmetric band matrix format\n");
    ij = 0;
    for (i=0;i<n;i++) {
        jmin = max(i-nh, 0);
        for (j=jmin;j<=i;j++)
            printf("%7.2f ",a[ij++]);
        printf("\n");
    }
}

MAIN__()
{
    int ierr, icon;
    int n, nh, i, j, ij, k, jmin;
    double asb[NMAX*(NHMAX+1)-NHMAX*(NHMAX+1)/2], as[NMAX*(NMAX+1)/2];

    n = NMAX;
    /* initialize symmetric band matrix */
    nh = NHMAX;
    ij = 0;
    for (i=0;i<n;i++) {
        jmin = max(i-nh, 0);
        for (j=jmin;j<=i;j++)
            asb[ij++] = i-j+1;
    }
    k = NMAX;
    /* convert to symmetric matrix storage format */
    ierr = c_dcsbsm(asb, n, nh, as, &icon);
    if (icon != 0) {
        printf("ERROR: c_dcsbsm failed with icon = %d\n", icon);
        exit(1);
    }
    /* print matrices */
    printf("asb: \n");
    prtsymbandmat(asb, n, nh);
    printf("as: \n");
    prtsymmat(as, n);
    return(0);
}

```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”のCSBSMの項目を参照のこと。

c_dcsgm

行列格納方法の変換 (対称行列用圧縮格納法→一般行列の格納方法)

<code>ierr = c_dcsgm(as, n, ag, k, &icon);</code>

1. 機能

対称行列用圧縮格納法で格納された $n \times n$ の実対称行列を, 一般行列の格納方法に変換する. $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dcsgm(as, n, (double*)ag, k, &icon);
```

引数の説明:

as	double as[Aslen]	入力	対称行列用圧縮格納法で格納された対称行列 A . $Aslen = n(n+1)/2$. 格納法の詳細については“1.3.2 対称行列の格納方法”を参照のこと.
n	int	入力	行列 A の次数 n .
ag	double ag[n][k]	出力	一般行列の格納方法で格納された対称行列 A .
k	int	入力	配列 ag の整合寸法 ($\geq n$).
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
30000	次のいずれかであった. • $n < 1$ • $k < n$	処理を打ち切る.

3. 使用上の注意

a) 一般形式の格納方法への対称行列の格納方法

本関数により変換された一般行列の格納方法の対称行列とは, 上三角部分, 対角部分だけではなく, 下三角部分も格納されている.

b) 領域の節約について

配列 as 上の内容を保存する必要のない場合は, 引数 ag のかわりに as を指定することにより領域が節約できる. このとき引数 ag のほうが大きなサイズを必要とするので, 十分なサイズをとっておくこと.

4. 使用例

対称行列用圧縮格納法で格納された実対称行列を, 一般行列の格納方法に変換し両方の行列を出力する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 5

/* print symmetric matrix */
void prtsymmat(double a[], int n)
{
    int ij, i, j;
    printf("symmetric matrix format\n");
    ij = 0;
    for (i=0;i<n;i++) {
        for (j=0;j<=i;j++)
            printf("%7.2f ", a[ij++]);
        printf("\n");
    }
}

/* print general matrix */
void prtgenmat(double *a, int k, int n, int m)
{
    int i, j;
    printf("general matrix format\n");
    for (i=0;i<n;i++) {
        for (j=0;j<m;j++)
            printf("%7.2f ", a[i*k+j]);
        printf("\n");
    }
}

MAIN__()
{
    int ierr, icon;
    int n, i, j, ij, k;
    double as[NMAX*(NMAX+1)/2], ag[NMAX][NMAX];

    /* initialize symmetric matrix storage format */
    n = NMAX;
    ij = 0;
    for (i=0;i<n;i++)
        for (j=0;j<=i;j++) {
            as[ij++] = n-i;
        }
    k = NMAX;
    /* convert to general matrix storage format */
    ierr = c_dcsgm(as, n, (double*)ag, k, &icon);
    if (icon != 0) {
        printf("ERROR: c_dcsgm failed with icon = %d\n", icon);
        exit(1);
    }
    /* print matrices */
    printf("as: \n");
    prtsymmat(as, n);
    printf("ag: \n");
    prtgenmat((double*)ag, k, n, n);
    return(0);
}
```

5. 手法概要

1次元配列 `as` 上に対称行列用圧縮格納法で格納された実対称行列を, 次を示す手順により, 2次元配列上に一般行列の格納方法で格納する.

- `as` の後から順に, `ag` の対角部分と下三角部分とに転送する. 転送は n 列から列ごとに行なう.

$NT = n(n+1)/2$ としたとき以下に対応関係を示す.

圧縮モード上の要素		行列の要素		一般モード上の要素
as[NT-1]	→	a_{nn}	→	ag[n-1][n-1]
as[NT-2]	→	a_{nn-1}	→	ag[n-1][n-2]
⋮		⋮		⋮
as[i*(i-1)/2+j-1]	→	a_{ij}	→	ag[i-1][j-1]
⋮		⋮		⋮
as[1]	→	a_{21}	→	ag[0][1]
as[0]	→	a_{11}	→	ag[0][0]

- 対角部分を対称軸として, 下三角部分を上三角部分に, $ag[i][j] = ag[j][i]$ となるように転送する.

ただし, $j < i$ とする.

計算方法の詳細については“富士通 SSL II 使用手引書”のCSGMの項目を参照のこと.

c_dcssbm

行列格納方法の変換 (対称行列用圧縮格納法→対称バンド行列用圧縮格納法)

ierr = c_dcssbm(as, n, asb, nh, &icon);

1. 機能

対称行列用圧縮格納法で格納された $n \times n$ 、バンド幅 h の実対称バンド行列を、対称バンド行列用圧縮格納法に変換する。 $n > h \geq 0$ であること。

2. 引数

呼出し形式:

```
ierr = c_dcssbm(as, n, asb, nh, &icon);
```

引数の説明:

as	double as[Aslen]	入力	対称行列用圧縮格納法で格納された対称バンド行列 A . $Aslen = n(n+1)/2$. 格納法の詳細については“1.3.2 対称行列の格納方法”を参照のこと。
n	int	入力	行列 A の次数 n .
asb	double asb[Asblen]	出力	対称バンド行列用圧縮格納法で格納された対称バンド行列 A . $Asblen = n(h+1) - h(h+1)/2$. 格納法の詳細については“1.3.4 バンド行列の格納方法”を参照のこと。
nh	int	入力	行列 A のバンド幅 h .
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
30000	次のいずれかであった. • $nh < 0$ • $n \leq nh$	処理を打ち切る.

3. 使用上の注意

a) 領域の節約について

配列 as 上の内容を保存する必要のない場合は、引数 asb のかわりに as を指定することにより領域が節約できる。このとき引数 as のほうが大きなサイズを必要とするので、十分なサイズをとっておくこと。

4. 使用例

対称バンド行列が対称行列用圧縮格納法で与えられたとき、これを対称バンド行列用圧縮格納法に変換する。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */
```

```
#define max(a,b) ((a) > (b) ? (a) : (b))

#define NMAX 5
#define NHMAX 2

/* print symmetric matrix */
void prtsymmat(double a[], int n)
{
    int ij, i, j;
    printf("symmetric matrix format\n");
    ij = 0;
    for (i=0;i<n;i++) {
        for (j=0;j<=i;j++)
            printf("%7.2f ",a[ij++]);
        printf("\n");
    }
}

/* print symmetric band matrix */
void prtsymbandmat(double a[], int n, int nh)
{
    int ij, i, j, jmin;
    printf("symmetric band matrix format\n");
    ij = 0;
    for (i=0;i<n;i++) {
        jmin = max(i-nh, 0);
        for (j=jmin;j<=i;j++)
            printf("%7.2f ",a[ij++]);
        printf("\n");
    }
}

MAIN__()
{
    int ierr, icon;
    int n, nh, i, j, ij, k;
    double asb[NMAX*(NHMAX+1)-NHMAX*(NHMAX+1)/2], as[NMAX*(NMAX+1)/2];

    /* initialize band symmetric matrix in symmetric matrix storage format */
    n = NMAX;
    nh = NHMAX;
    ij = 0;
    for (i=0;i<n;i++)
        for (j=0;j<=i;j++) {
            if (abs(i-j) <= nh)
                as[ij++] = i-j+1;
            else
                as[ij++] = 0;
        }
    k = NMAX;
    /* convert to symmetric band matrix storage format */
    ierr = c_dcssbm(as, n, asb, nh, &icon);
    if (icon != 0) {
        printf("ERROR: c_dcssbm failed with icon = %d\n", icon);
        exit(1);
    }
    /* print matrices */
    printf("as: \n");
    prtsymmat(as, n);
    printf("asb: \n");
    prtsymbandmat(asb, n, nh);
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”のCSSBMの項目を参照のこと。

c_dctsdm

複素超越方程式 (マラー法)

<pre>ierr = c_dctsdm(&z, zfun, isw, eps, eta, &m, &icon);</pre>

1. 機能

与えられた初期値をもとに複素超越方程式,

$$f(z) = 0$$

の1根をマラー (Muller) 法により求める.

2. 引数

呼出し形式:

```
ierr = c_dctsdm(&z, zfun, isw, eps, eta, &m, &icon);
```

引数の説明:

z	dcomplex	入力	求めようとする根の初期値.
		出力	近似根.
zfun	function	入力	解こうとする方程式の関数 $f(z)$ を計算する複素数型ユーザ関数名.
			プロトタイプ宣言: dcomplex zfun(dcomplex z);
			引数の説明:
		z	dcomplex 入力 独立変数.
isw	int	入力	制御情報. 求めようとする根の収束判定方法を指定する. isw は, 以下のいずれかの値を設定する.
			1 $ f(z_i) \leq \text{eps}$: 収束判定法 I を満たす z_i を根とする.
			2 $ z_i - z_{i-1} \leq \text{eta} \cdot z_i $: 収束判定法 II を満たす z_i を根とする.
			3 上の二つの収束判定法を同時に採用し少なくとも一方が満たされたとき, z_i を根とする.
			(“使用上の注意”a)参照)
eps	double	入力	収束判定法 I (引数“isw”の項参照)で用いられる収束判定値 (≥ 0).
eta	double	入力	収束判定法 II (引数“isw”の項参照)で用いられる収束判定値 (≥ 0). (“使用上の注意”b)参照)
m	int	入力	根を求めるための反復回数の上限 (≥ 0). (“使用上の注意”c)参照)
		出力	実際に反復した回数.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
1	求まった近似根が収束判定法 I (引数“isw”の項参照)を満たした.	
2	求まった近似根が収束判定法 II (引数“isw”の項参照)を満たした.	
10	反復の回数は無条件に m 回 ($m = -m$) 繰り返した.	
11	反復回数は無条件に m 回繰り返すと指定されたが ($m = -m$), その回数の反復以前に $ f(z_i) = 0$ となったので反復を止めて, z_i を根とした.	
12	反復回数は無条件に m 回繰り返すと指定されたが ($m = -m$), その回数の反復以前に $ z_i - z_{i-1} \leq \mu \cdot z_i $ となったので反復を止めて, z_i を根とした.	z には最後の反復値 z_i が格納される.
10000	与えられた反復回数内で指定された収束条件を満たさなかった.	
20000	$f(z_{i-2}) = f(z_{i-1}) = f(z_i)$ の状態が起こったとき, z_{i-2} , z_{i-1} , z_i に摂動を与えて問題の回避を行なっているが, この摂動が 5 回以上行われた.	処理を打ち切る.
30000	次のいずれかであった. $m > 0$ のとき $\text{isw} = 1$ かつ $\text{eps} < 0$ $\text{isw} = 2$ かつ $\text{eta} < 0$ $\text{isw} = 3$ かつ, $\text{eps} < 0$ 又は $\text{eta} < 0$ $m = 0$ $\text{isw} \neq 1, 2, 3.$	処理を打ち切る.

3. 使用上の注意

a) 引数 isw

本関数は, $\text{isw} = 1$ と与えられたときでも, 反復の過程で

$$|z_i - z_{i-1}| \leq \mu \cdot |z_i| \quad (\mu \text{ は丸め誤差の単位})$$

が満たされたときは反復を止め, $\text{icon} = 2$ と出力する. 同様に, $\text{isw} = 2$ と与えられたときでも, 反復の過程で

$$|f(z_i)| = 0$$

となったときは反復を止め, $\text{icon} = 1$ と出力する.

b) 引数 eta

求めようとする根が多重根であつたり, 近接根である場合は大きめにとる必要がある. なお, $0 \leq \text{eta} < \mu$ (μ は丸め誤差の単位) であつたときは, 関数内部で $\text{eta} = \mu$ と置き換えている.

c) 引数 m

反復回数 m を $m = -m$ ($m > 0$) と与えると反復は無条件に m 回繰り返す.

ただし、引数“isw”と同様に、反復された近似根が

$$|f(z_i)| = 0 \quad \text{又は} \quad |z_i - z_{i-1}| \leq \mu \cdot |z_i|$$

のとき、反復を止め、icon = 11 又は icon = 12 と出力する。

4. 使用例

関数 $f(z) = e^z - i$ の 1 根を初期値 $z_0 = 0 + i0$ として求める。

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

dcomplex zfun(dcomplex z); /* user function prototype */

MAIN__()
{
    int ierr, icon;
    dcomplex z;
    double eps, eta;
    int isw, m;

    z.re = 0;
    z.im = 0;
    isw = 3;
    eps = 0;
    eta = 1.0e-6;
    m = 100;
    /* find zero of complex function */
    ierr = c_dctsdm(&z, zfun, isw, eps, eta, &m, &icon);
    printf("icon = %i   m = %i   z = {%12.4e, %12.4e}\n", icon, m, z.re, z.im);
    return(0);
}

/* complex user function: zfun(z) = z*z - zm */
dcomplex zfun(dcomplex z)
{
    const dcomplex zm = {0, 1};
    dcomplex zres;
    zres.re = z.re*z.re - z.im*z.im - zm.re;
    zres.im = 2*z.re*z.im - zm.im;
    return(zres);
}
```

5. 手法概要

本関数は、複素超越方程式の 1 根をマラー法を用いて求めている。

計算方法の詳細については“富士通 SSL II 使用手引書”の CTSDM の項目及び[111]を参照のこと。

c_decheb

チェビシェフ級数の求和

ierr = c_decheb(a, b, c, n, v, &f, &icon);

1. 機能

区間 $[a, b]$ で定義された n 項のチェビシェフ級数,

$$f(x) = \sum_{k=0}^{n-1} c_k T_k \left(\frac{2x - (b+a)}{b-a} \right)$$

が与えられたとき, 区間内の任意の点 v における級数の値 $f(v)$ を求める. ここで, $\sum$ は初項を $1/2$ 倍にして和をとることを意味する. ただし, $a \neq b$, $v \in [a, b]$, $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_decheb(a, b, c, n, v, &f, &icon);
```

引数の説明:

a	double	入力	チェビシェフ級数の定義域の下限 a .
b	double	入力	チェビシェフ級数の定義域の上限 b .
c	double c[n]	入力	係数 c_k . $c[k] = c_k$.
n	int	入力	級数の項数 n .
v	double	入力	区間 $[a, b]$ 内の点 v .
f	double	出力	点 v における級数の値 $f(v)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	次のいずれかであった. $n < 1$ $a = b$ $v \notin [a, b]$	処理を打ち切る.

3. 使用上の注意

本関数は, チェビシェフ級数の値 $f(v)$ を求める. 任意の滑らかな関数 $f(v)$ をチェビシェフ級数展開する場合は, 関数 c_dfcheb を用いればよい.

4. 使用例

関数 $f(x) = \sin x$ を c_dfcheb により級数展開し, 求めた展開係数を用いて級数の値を求める.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 257 /* default value */
```



```

double fun(double x); /* function prototype */

MAIN__()
{
    int ierr, icon;
    int i, n, nmin, nmax;
    double epsa, epsr, err, a, b, pi, v, f, h;
    double c[NMAX], tab[NMAX-2];

    /* initialize data */
    epsa = 5e-5;
    epsr = 0;
    nmin = 9; /* default value */
    nmax = NMAX;
    pi = 2*asin(1);
    a = 0;
    b = pi;
    /* expand function as Chebyshev series */
    ierr = c_dfcheb(a, b, fun, epsa, epsr, nmin, nmax, c, &n, &err, tab, &icon);
    if (icon >= 20000) {
        printf("ERROR: icon = %4i\n", icon);
        exit(1);
    }
    /* now evaluate Chebyshev series at 32 points */
    h = pi/(2*32);
    printf(" v          f          error   \n");
    for (i=0; i<32; i++) {
        v = b*pow(cos(i*h), 2);
        ierr = c_decheb(a, b, c, n, v, &f, &icon);
        if (icon != 0) {
            printf("ERROR: icon = %4i\n", icon);
            exit(1);
        }
        err = fun(v) - f;
        printf("%6.3f %12.5e %12.5e\n", v, f, err);
    }
    return(0);
}

/* function to expand */
double fun(double x)
{
    double sum, xn, xp, p, term, eps;
    int n;
    eps = 1e-7; /* approx. amach */
    sum = x;
    p = x*x;
    xp = x*p;
    xn = -6;
    n = 3;
    while (1) {
        term = xp/xn;
        sum = sum+term;
        if (fabs(term) <= eps) break;
        n = n+2;
        xp = xp*p;
        xn = -xn*n*(n-1);
    }
    return (sum);
}

```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の ECHEB の項目を参照のこと。

c_decosp

cosine 級数の求和

ierr = c_decosp(th, a, n, v, &f, &icon);

1. 機能

周期 $2T$ をもつ n 項の cosine 級数

$$f(t) = \frac{1}{2}a_0 + \sum_{k=1}^{n-1} a_k \cos \frac{\pi}{T} kt$$

が与えられたとき、任意の点 v における級数の値 $f(v)$ を求める。ただし、 $T > 0$, $n \geq 1$ とする。

2. 引数

呼出し形式:

```
ierr = c_decosp(th, a, n, v, &f, &icon);
```

引数の説明:

th	double	入力	cosine 級数の半周期 T .
a	double a[n]	入力	係数 a_k . $a[k] = a_k$.
n	int	入力	級数の項数 n .
v	double	入力	点 v .
f	double	出力	点 v における級数の値 $f(v)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
30000	次のいずれかであった: • $n < 1$ • $th \leq 0$	処理を打ち切る.

3. 使用上の注意

本関数は、cosine 級数の値 $f(v)$ を求める。任意の周期 $2T$ をもつ滑らかな偶関数 $f(t)$ を cosine 級数展開する場合は、関数 c_dfcosf を用いればよい。

4. 使用例

助変数 ω を含む奇関数の積分(1)の値を求める。ただし $\omega = \pi/4$ 。

$$F(x) = \int_0^x \frac{\sin \omega t}{\sqrt{1 + \sin^2 \omega t}} dt \quad (1)$$

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 257 /* default value */

double truefun(double t); /* prototype for check function */
double fun(double t); /* integral function prototype */
double w; /* auxiliary variable for function fun */
```

```

MAIN__()
{
    int ierr, icon;
    int i, n, nmin, nmax;
    double epsa, epsr, err, th, pi, v, f, q, h;
    double a[NMAX], tab[(NMAX-3)/2];

    /* initialize data */
    epsa = 0.5e-4;
    epsr = epsa;
    nmin = 0; /* default value */
    nmax = NMAX;
    pi = 2*asin(1);
    w = pi/4;
    th = pi/w;
    /* expand integral function as sine series */
    ierr = c_dfssinf(th, fun, epsa, epsr, nmin, nmax, a, &n, &err, tab, &icon);
    if (icon >= 20000) {
        printf("ERROR: icon = %4i\n", icon);
        exit(1);
    }
    /* integrate termwise */
    for (i=1;i<n;i++)
        a[i] = -a[i]/(i*w);
    /* evaluate cosine series at v=0 to find a0 value */
    v = 0;
    ierr = c_decosp(th, a, n, v, &f, &icon);
    if (icon != 0) {
        printf("ERROR: icon = %4i\n", icon);
        exit(1);
    }
    a[0] = -f*2; /* notice factor of 2 */
    /* now evaluate cosine series to give integral */
    h = th/10;
    printf(" v          f          exact   \n");
    for (i=1;i<=10;i++) {
        v = i*h;
        ierr = c_decosp(th, a, n, v, &f, &icon);
        if (icon != 0) {
            printf("ERROR: icon = %4i\n", icon);
            exit(1);
        }
        q = truefun(v); /* exact integral */
        printf("%4.2f %12.6e %12.6e\n", v, f, q);
    }
    return(0);
}

/* function to integrate */
double fun(double t)
{
    double p;
    p = sin(w*t);
    return p/sqrt(1+p*p);
}

/* exact integral function */
double truefun(double t)
{
    double pi;
    pi = 2*asin(1);
    return (pi/4-asin(cos(w*t)*sqrt(0.5)))/w;
}

```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の ECOSP の項目を参照のこと。

c_deig1

実行列の固有値及び固有ベクトル(2段 QR 法)
<pre>ierr = c_deig1(a, k, n, mode, er, ei, ev, vw, &icon);</pre>

1. 機能

n 次の実行列 $\mathbf{A}$ の固有値と対応する固有ベクトルを求める. 固有ベクトルは $\|x\|_2 = 1$ となるように正規化される. $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_deig1((double *)a, k, n, mode, er, ei, (double *)ev, vw, &icon);
```

引数の説明:

a	double	入力	実行列 $\mathbf{A}$.
	a[n][k]	出力	演算後内容は保存されない.
k	int	入力	配列 a 及び ev の整合寸法 ($k \geq n$).
n	int	入力	実行列 $\mathbf{A}$ の次数 n .
mode	int	入力	平衡化の省略の指定. • mode = 1 のときは, 平衡化を省略する. • mode $\neq$ 1 のときは, 平衡化を行なう. (“使用上の注意”b)参照)
er	double er[n]	出力	固有値の実部.
ei	double ei[n]	出力	固有値の虚部. 第 j 番目の固有値が複素固有値であれば, 第 $j+1$ 番目の固有値は, 第 j 番目の固有値と共役な固有値である.
ev	double ev[n][k]	出力	固有ベクトル. 固有ベクトルは, 固有値と対応する行に格納される. ただし, 第 j 番目の固有値が複素固有値であれば, 固有ベクトルも複素ベクトルとなる. 実部は, 第 j 行に格納され, 虚部は, 第 $j+1$ 行に格納される.(“使用上の注意”a)参照)
vw	double vw[n]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
10000	$n=1$ であった.	er[0] = a[0][0] ev[0][0] = 1 とする.
20000	三角行列に変換できず, 固有値・固有ベクトルは求められなかった.	処理を打ち切る.

コード	意味	処理内容
30000	次のいずれかであった. • $n < 1$ • $k < n$	処理を打ち切る.

3. 使用上の注意

a) 複素固有値と対応する固有ベクトルについて

一般に実行列の固有値には, 実固有値と複素固有値が存在し, 複素固有値は共役な複素固有値と対になっている.

本関数では, ある j 番目の固有値 (λ_j) が複素固有値のとき, λ_j 及び λ_j^* を `er` 及び `ei` に, 次のように格納する.

$$\lambda_j = \text{er}[j-1] + i \cdot \text{ei}[j-1]$$

$$\begin{aligned}\lambda_j^* &= \text{er}[j] + i \cdot \text{ei}[j] \\ &= \text{er}[j-1] - i \cdot \text{ei}[j-1]\end{aligned}$$

一方, λ_j に対応する固有ベクトル $\mathbf{x}_j$ は,

$$\mathbf{x}_j = \mathbf{u}_j + i \cdot \mathbf{v}_j$$

なる複素ベクトルとなり, λ_j^* に対応する固有ベクトル $\mathbf{x}_j^*$ は,

$$\mathbf{x}_j^* = \mathbf{u}_j - i \cdot \mathbf{v}_j$$

となる. したがって, $\mathbf{x}_j$ の実部 ($\mathbf{u}_j$) と虚部 ($\mathbf{v}_j$) をそれぞれ保存しておけば, $\mathbf{x}_j^*$ は簡単に求まる. したがって, 本関数では, λ_j に対応する固有ベクトル $\mathbf{x}_j$ だけを求め, λ_j^* に対応する固有ベクトル $\mathbf{x}_j^*$ は計算していない.

なお, $\mathbf{x}_j$ の実部と虚部は, 次のように格納する.

$$\mathbf{u}_j = \text{ev}[j-1][m]$$

$$\mathbf{v}_j = \text{ev}[j][m]$$

$$: m = 0, 1, 2, \dots, n-1$$

b) 平衡化について

複素行列の各要素の大きさが, オーダの意味で大きく異なっているような場合には, 一般に行列の平衡化によって結果の精度を上げることができる. 平衡化を行なうには, `mode` $\neq 1$ と指定する.

他方, もし行列の各要素の大きさがほぼそろっているような場合には, 行列の平衡化による効果を期待できない. このような場合には, `mode` = 1 とすることにより, 平衡化を省略できる.

4. 使用例

5×5 の行列の固有値及び固有ベクトルを計算する.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 5

MAIN__()
{
    int ierr, icon;
    int n, i, j, k, mode;
```

```
double a[NMAX][NMAX], er[NMAX], ei[NMAX], ev[NMAX][NMAX], vw[NMAX];

/* initialize matrix */
n = NMAX;
k = NMAX;
for (i=0;i<n;i++)
    for (j=0;j<=i;j++) {
        a[i][j] = i-n;
        a[j][i] = n-i;
    }
mode = 0;
/* find eigenvalues and eigenvectors */
ierr = c_deig1((double*)a, k, n, mode, er, ei, (double*)ev, vw, &icon);
printf("icon = %i\n", icon);
/* print eigenvalues and eigenvectors */
i = 0;
while (i<n) {
    if (ei[i] == 0) {
        /* real eigenvector */
        printf("eigenvalue: %12.4f\n", er[i]);
        printf("eigenvector:");
        for (j=0;j<n;j++)
            printf("%7.4f ", ev[i][j]);
        printf("\n");
        i++;
    }
    else {
        /* complex eigenvector pair */
        printf("eigenvalue:  {%7.4f, %7.4f}\n", er[i], ei[i]);
        printf("eigenvector:  ");
        for (j=0;j<n;j++)
            printf("{%7.4f, %7.4f} ", ev[i][j], ev[i+1][j]);
        printf("\n");
        printf("eigenvalue:  {%7.4f, %7.4f}\n", er[i+1], ei[i+1]);
        printf("eigenvector:  ");
        for (j=0;j<n;j++)
            printf("{%7.4f, %7.4f} ", ev[i][j], -ev[i+1][j]);
        printf("\n");
        i = i+2;
    }
}
return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”のEIG1の項目及び[118], [119]を参照のこと.

c_desinp

sine 級数の求和

ierr = c_desinp(th, b, n, v, &f, &icon);

1. 機能

周期 $2T$ をもつ n 項の sine 級数,

$$f(t) = \sum_{k=0}^{n-1} b_k \sin \frac{\pi}{T} kt$$

が与えられたとき, 任意の点 v における級数の値 $f(v)$ を求める. ただし, $T > 0$, $n \geq 1$ とする.

2. 引数

呼出し形式:

```
ierr = c_desinp(th, b, n, v, &f, &icon);
```

引数の説明:

th	double	入力	sine 級数の半周期 T .
b	double b[n]	入力	係数 b_k . $b[0] = 0$, $b[1] = b_1, \dots, b[n-1] = b_{n-1}$ のように格納する.
n	int	入力	級数の項数 n .
v	double	入力	点 v .
f	double	出力	点 v における級数の値 $f(v)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
30000	次のいずれかであった. • $n < 1$ • $th \leq 0$	処理を打ち切る.

3. 使用上の注意

本関数は, sine 級数の値 $f(v)$ を求める. 任意の周期 $2T$ をもつ滑らかな奇関数 $f(t)$ を sine 級数展開する場合は, 関数 `c_dfsinf` を用いればよい.

4. 使用例

助変数 ω を含む偶関数の積分(1) の値を求める. ただし $\omega = \pi/4$.

$$F(x) = \int_0^x \frac{\cos \omega t}{\sqrt{1 + \cos^2 \omega t}} dt \quad (2)$$

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 257 /* default value */
```

```
double truefun(double t); /* prototyptpe for check function */
double fun(double t); /* integral function prototype */
double w; /* auxiliary variable for function fun */

MAIN__()
{
    int ierr, icon;
    int i, n, nmin, nmax;
    double epsa, epsr, err, th, pi, v, f, q, h;
    double b[NMAX], tab[(NMAX-3)/2];

    /* initialize data */
    epsa = 0.5e-4;
    epsr = epsa;
    nmin = 0; /* default value */
    nmax = NMAX;
    pi = 2*asin(1);
    w = pi/4;
    th = pi/w;
    /* expand integral function as cosine series */
    ierr = c_dfcof(th, fun, epsa, epsr, nmin, nmax, b, &n, &err, tab, &icon);
    if (icon >= 20000) {
        printf("ERROR: icon = %4i\n", icon);
        exit(1);
    }
    /* integrate termwise */
    for (i=1;i<n;i++)
        b[i] = b[i]/(i*w);
    /* now evaluate cosine series to give integral */
    h = th/10;
    printf(" v      f      exact  \n");
    for (i=1;i<=10;i++) {
        v = i*h;
        ierr = c_desinp(th, b, n, v, &f, &icon);
        if (icon != 0) {
            printf("ERROR: icon = %4i\n", icon);
            exit(1);
        }
        q = truefun(v); /* exact integral */
        printf("%4.2f %12.6e %12.6e\n", v, f, q);
    }
    return(0);
}

/* function to integrate */
double fun(double t)
{
    double p;
    p = cos(w*t);
    return p/sqrt(1+p*p);
}

/* exact integral function */
double truefun(double t)
{
    return asin(sin(w*t)*sqrt(0.5))/w;
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”のESINPの項目を参照のこと。

c_dexpi

指数積分 $E_i(x)$, $\bar{E}_i(x)$

ierr = c_dexpi(x, &ei, &icon);

1. 機能

次のように定義される指数積分 $E_i(x)$, $\bar{E}_i(x)$ の値を, 近似式を用いて計算する.

$x < 0$ の場合:

$$E_i(x) = -\int_{-x}^{\infty} \frac{e^{-t}}{t} dt = \int_{-\infty}^x \frac{e^t}{t} dt$$

$x > 0$ の場合:

$$\bar{E}_i(x) = -\text{P.V.} \int_{-x}^{\infty} \frac{e^{-t}}{t} dt = \text{P.V.} \int_{-\infty}^x \frac{e^t}{t} dt$$

ここに, P.V. は $t=0$ で主値をとって積分することを意味する. ただし, $x \neq 0$ であること.

2. 引数

呼出し形式:

```
ierr = c_dexpi(x, &ei, &icon);
```

引数の説明:

x	double	入力	独立変数 x . (“使用上の注意”参照)
ei	double	出力	関数値 $E_i(x)$ 又は $\bar{E}_i(x)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
20000	$x > \log(fl_{\max})$ であった. $x < -\log(fl_{\max})$ であった.	ei = $fl_{\max}$ とする. ei = 0 とする.
30000	$x = 0$	ei = 0 とする.

3. 使用上の注意

a) 引数 x の範囲

- $x \neq 0$ であること. $E_i(x)$, $\bar{E}_i(x)$ は $x=0$ では定義されていない.
- $|x| \leq \log(fl_{\max})$ であること. $|x|$ の値が上記の範囲を超えると, $E_i(x)$, $\bar{E}_i(x)$ はそれぞれ, e^x の計算においてアンダフロー, オーバフローをおこす. これを本関数内で前もって防ぐためである. $fl_{\max}$ については, “1.5 計算機定数”を参照のこと.

4. 使用例

$\bar{E}_i(x)$ の値を x が 0.01 から 1.0 まで, 0.01 おきに計算して数表を作成する.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
```

```
{
  int ierr, icon;
  double x, ei;
  int i;

  for (i=1;i<=100;i++) {
    x = (double)i/100;
    /* calculate integral */
    ierr = c_dexpi(x, &ei, &icon);
    if (icon == 0)
      printf("x = %5.2f    ei = %f\n", x, ei);
    else
      printf("ERROR: x = %5.2f    ei = %f    icon = %i\n", x, ei, icon);
  }
  return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の EXPI の項目及び[22], [23]を参照のこと。

c_dfcheb

実関数のチェビシェフ級数展開 (関数入力, 高速 cosine 変換)
ierr = c_dfcheb(a, b, fun, epsa, epsr, nmin, nmax, c, &n, &err, tab, &icon);

1. 機能

区間 $[a, b]$ で滑らかな関数 $f(x)$ と, 要求精度 $\varepsilon_a, \varepsilon_r$ が与えられたとき, $f(x)$ をチェビシェフ級数展開し,

$$\left| f(x) - \sum_{k=0}^{n-1} c_k T_k \left(\frac{2x - (b+a)}{b-a} \right) \right| \leq \max\{\varepsilon_a, \varepsilon_r \|f\|\} \quad (1)$$

を満たす n 個の係数 $c_0, c_1, \dots, c_{n-1}$ を求める. ここで, $\sum$ は初項を $1/2$ 倍して和をとることを意味する. また, $\|f\|$ は, 区間 $[a, b]$ での関数の標本点,

$$x_j = \frac{a+b}{2} + \frac{b-a}{2} \cos\left(\frac{\pi}{n-1} j\right), \quad j = 0, 1, \dots, n-1$$

における関数値を用いて,

$$\|f\| = \max_{0 \leq j \leq n-1} |f(x_j)|$$

と定義する. ただし, $a \neq b$, $\varepsilon_a \geq 0$, $\varepsilon_r \geq 0$ であること.

2. 引数

呼出し形式:

```
ierr = c_dfcheb(a, b, fun, epsa, epsr, nmin, nmax, c, &n, &err, tab, &icon);
```

引数の説明:

a	double	入力	関数 $f(x)$ の定義域の下限 a .
b	double	入力	関数 $f(x)$ の定義域の上限 b .
fun	function	入力	展開しようとする関数 $f(x)$ を計算するユーザ関数名. (“使用例”参照) プロトタイプ宣言: double fun(double x); 引数の説明: x double 入力 独立変数.
epsa	double	入力	級数展開の絶対誤差の上限 ε_a . (“使用上の注意” c)参照)
epsr	double	入力	級数展開の相対誤差の上限 ε_r . (“使用上の注意” c)参照)
nmin	int	入力	級数展開の項数 n の下限 (≥ 0). $2^k + 1$ の値で与える ($k \geq 0$). 標準値は 9. (“使用上の注意” d)参照)
nmax	int	入力	級数の項数 n の上限 ($\geq nmin$). $2^k + 1$ の値で与える ($k \geq 0$). 標準値は 257. (“使用上の注意” d)参照)
c	double c[nmax]	出力	係数 c_k . $c[k] = c_k$, $k = 0, 1, \dots, n-1$ のように格納される.
n	int	出力	級数展開の項数 n (≥ 5). $n = 2^k + 1$ ($k \geq 2$)
err	double	出力	級数展開の絶対誤差の推定値. (“使用上の注意” c)参照)

tab	double tab[Tablen]	出力	級数展開で使用された三角関数表が格納される。 $a \neq 0$ のとき, $Tablen = \max\{3, (nmax-3)/2\}$ $a = 0$ のとき, $Tablen = \max\{3, nmax-2\}$ (“使用上の注意” a)参照)
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	丸め誤差のために, 要求精度が得られなかった. 要求精度が厳しすぎる.	c には, 得られた係数を出力する. 級数展開の誤差は達成可能な限度まで達している.
20000	級数展開の項数が, その上限に達しても要求精度が得られなかった.	処理を打ち切る. c には, そのときまでに得られた係数を出力する. また, err には, この時点での絶対誤差の推定値を出力する.
30000	次のいずれかであった. • $a = b$ • $epsa < 0$ • $epsr < 0$ • $nmin < 0$ • $nmax < nmin$	処理を打ち切る.

3. 使用上の注意

a) 区間 $[a, b]$ 及び 引数 **tab**

本関数では通常, 区間 $[a, b]$ を $[-1, 1]$ に変数変換し, $f(x)$ をチェビシェフ多項式系により展開する. しかし, 区間の端点 a が零の場合は, 変数変換での計算誤差 (桁落ち) の発生を防止するために, $f(x)$ をずらしチェビシェフ多項式系により展開している. ただし, 求める係数 $\{c_k\}$ は, 同一である. その為, 内部で使用する三角関数表は, 前者の場合 $(nmax-3)/2$, 後者の場合 $nmax-2$ となっている.

本関数を繰り返し呼ぶ場合, 必要な三角関数表は, 1 回だけしか計算されない. すなわち, 三角関数表に不足が生じた場合, その都度不足分だけ計算し追加される. したがって, 引数 **tab** の内容は, 常に保存したまま呼び出すこと.

b) 精度

展開の項数 n の増大とともに, 誤差が小さくなる度合は, 関数 $f(x)$ の滑らかさと区間 $[a, b]$ の中の大きさに著しく影響される. $f(x)$ が解析関数ならば, 誤差は指数関数的オーダー $O(r^n)$, $0 < r < 1$ で減少し, $f(x)$ が k 回

連続微分可能ならば, 有理式のオーダー $O\left(\left|\frac{a-b}{n}\right|^k\right)$ で減少する. $k=0, 1$ の場合, 展開項数が多くなり, 誤差の推

定も正確さを期しがたい. したがって, 本関数を適用する関数は, 少なくとも 2 回連続微分可能であることが望ましい.

c) 引数 **epsa** 及び **epsr**

本関数は, 与えられた $\varepsilon_a, \varepsilon_r$ に対して (1) を満たす級数を求める. したがって, $\varepsilon_r = 0$ とおけば絶対誤差 ε_a 以内の, また $\varepsilon_a = 0$ とおけば相対誤差 ε_r 以内の精度で級数展開をすることになる. しかし, 関数の性質と $\varepsilon_a, \varepsilon_r$ の値によってはこの目的が達成されないこともある. 例えば, 関数の計算精度に比べて, $\varepsilon_a, \varepsilon_r$ が小さすぎる場合には展開の項数がその上限に達しない場合でも, 丸め誤差の影響の方が大きくなり, これ以上計算を続行しても無意味である.

このような状態を見出したときは icon を 10000 にして処理を打ち切る. このときの級数の精度は, 使用計算機で達成可能な限度まで達している. また, 展開の項数が nmax 以内では収束しないこともあり, この場合 icon を 20000 にして処理を打ち切る. このときの係数の値は, それまでに得られている近似値であり, 級数の精度は保証されない.

なお, 本関数ではいずれの場合にも, 級数の絶対誤差の推定値を引数 err に出力するので精度の目安にすることができる.

d) 引数 nmin 及び nmax

nmin, nmax が $2^k + 1 (k \geq 0)$ の値でない場合, それらを超えない最大の $2^k + 1$ の値とみなす. ただし, $nmax < 5$ の場合, $nmax = 5$ とみなす.

4. 使用例

関数 $f(x) = \sin x$ を級数展開し, 求めた展開係数を用いて c_decheb により級数の値を求める.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 257 /* default value */

double fun(double x); /* function prototype */

MAIN__()
{
    int ierr, icon;
    int i, n, nmin, nmax;
    double epsa, epsr, err, a, b, pi, v, f, h;
    double c[NMAX], tab[NMAX-2];

    /* initialize data */
    epsa = 5e-5;
    epsr = 0;
    nmin = 9; /* default value */
    nmax = NMAX;
    pi = 2*asin(1);
    a = 0;
    b = pi;
    /* expand function as Chebyshev series */
    ierr = c_dfcheb(a, b, fun, epsa, epsr, nmin, nmax, c, &n, &err, tab, &icon);
    if (icon >= 20000) {
        printf("ERROR: icon = %4i\n", icon);
        exit(1);
    }
    /* now evaluate Chebyshev series at 32 points */
    h = pi/(2*32);
    printf(" v          f          error   \n");
    for (i=0; i<32; i++) {
        v = b*pow(cos(i*h), 2);
        ierr = c_decheb(a, b, c, n, v, &f, &icon);
        if (icon != 0) {
            printf("ERROR: icon = %4i\n", icon);
            exit(1);
        }
        err = fun(v) - f;
        printf("%6.3f %12.5e %12.5e\n", v, f, err);
    }
    return(0);
}

/* function to expand */
double fun(double x)
{
    double sum, xn, xp, p, term, eps;
    int n;
    eps = 1e-7; /* approx. amach */
    sum = x;
    p = x*x;

```

```
    xp = x*p;
    xn = -6;
    n = 3;
    while (1) {
        term = xp/xn;
        sum = sum+term;
        if (fabs(term) <= eps) break;
        n = n+2;
        xp = xp*p;
        xn = -xn*n*(n-1);
    }
    return (sum);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の FCHEB の項目及び[129]を参照のこと.

c_dfccosf

偶関数の cosine 級数展開 (関数入力, 高速 cosine 変換)

```
ierr = c_dfccosf(th, fun, epsa, epsr, nmin,
                 nmax, a, &n, &err, tab, &icon);
```

1. 機能

任意の周期 $2T$ をもつ滑らかな偶関数 $f(t)$ を要求精度 ε_a , ε_r に基づき cosine 級数展開する. すなわち

$$\left| f(t) - \sum_{k=0}^{n-1} a_k \cos \frac{\pi}{T} kt \right| \leq \max\{\varepsilon_a, \varepsilon_r \|f\|\} \quad (1)$$

を満たす n 個の係数 $\{a_k\}$ を求める.

ここで $\sum'$ は, 初項を $1/2$ 倍して和をとることを意味する. また, 関数のノルム $\|f\|$ は, 半周期 $[0, T]$ での関数の標本点,

$$t_j = \frac{T}{n-1} j, \quad j = 0, 1, \dots, n-1$$

における関数値を用いて

$$\|f\| = \max_{0 \leq j \leq n-1} |f(t_j)|$$

と定義する. ただし, $T > 0$, $\varepsilon_a \geq 0$, $\varepsilon_r \geq 0$ であること.

2. 引数

呼出し形式:

```
ierr = c_dfccosf(th, fun, epsa, epsr, nmin, nmax, a, &n, &err, tab, &icon);
```

引数の説明:

th	double	入力	関数 $f(t)$ の半周期 T .
fun	function	入力	展開しようとする関数 $f(t)$ を計算するユーザ関数名. (“使用例”参照) プロトタイプ宣言: double fun(double t); 引数の説明: t double 入力 独立変数
epsa	double	入力	級数展開の絶対誤差の上限 ε_a . (“使用上の注意” b)参照)
epsr	double	入力	級数展開の相対誤差の上限 ε_r . (“使用上の注意” b)参照)
nmin	int	入力	級数展開の項数 n の下限 (≥ 0). $2^k + 1$ の値で与える ($k \geq 0$). 標準値は 9. (“使用上の注意” c)参照)
nmax	int	入力	級数の項数 n の上限 ($\geq nmin$). $2^k + 1$ の値で与える ($k \geq 0$). 標準値は 257. (“使用上の注意” c)参照)
a	double a[nmax]	出力	係数 a_k . $a[k] = a_k$, $k = 0, 1, \dots, n-1$ のように格納される.
n	int	出力	級数展開の項数 n (≥ 5). $n = 2^k + 1$ ($k \geq 2$)
err	double	出力	級数展開の絶対誤差の推定値. (“使用上の注意” b)参照)
tab	double tab[Tablen]	出力	級数展開で使用された三角関数表が格納される. $Tablen = \max\{3, (nmax-3)/2\}$. (“使用上の注意” d)参照)

icon int 出力 コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
10000	丸め誤差のために, 要求精度が得られなかった. 要求精度が厳しすぎる.	a には, 得られた係数を出力する. 級数展開の誤差は達成可能な限度まで達している.
20000	級数展開の項数が, その上限に達しても要求精度が得られなかった.	処理を打ち切る. a には, そのときまでに得られた係数を出力する. また, err には, この時点での絶対誤差の推定値を出力する.
30000	次のいずれかであった. • $th \leq 0$ • $epsa < 0$ • $epsr < 0$ • $nmin < 0$ • $nmax < nmin$	処理を打ち切る.

3. 使用上の注意

a) 精度

展開の項数 n の増大とともに, 誤差が小さくなる度合は, 全区間 $(-\infty, \infty)$ における関数 $f(t)$ の滑らかさに著しく影響される. $f(t)$ が解析的周期関数ならば, 誤差は指数関数的オーダー $O(r^n)$, $0 < r < 1$ で減少し, $f(t)$ が k 回連続微分可能ならば, 有理式のオーダー $O(n^{-k})$ で減少する. $k = 0, 1$ の場合, 展開項数が多くなり, 誤差の推定も正確さを期しがたい. したがって, 本ルーチンを適用する関数は, 少なくとも 2 回連続微分可能であることが望ましい.

b) 引数 epsa 及び epsr

本関数は, 与えられた $\varepsilon_a, \varepsilon_r$ に対して(1)を満たす級数を求める. したがって, $\varepsilon_r = 0$ とおけば絶対誤差 ε_a 以内の, また $\varepsilon_a = 0$ とおけば相対誤差 ε_r 以内の精度で級数展開をすることになる. しかし, 関数の性質と $\varepsilon_a, \varepsilon_r$ の値によってはこの目的が達成されないこともある. 例えば, 関数の計算精度に比べて, $\varepsilon_a, \varepsilon_r$ が小さすぎる場合には展開の項数がその上限に達しない場合でも, 丸め誤差の影響の方が大きくなり, これ以上計算を続行しても無意味である.

このような状態を見出したときは icon を 10000 にして処理を打ち切る. このときの級数の精度は, 使用計算機で達成可能な限度まで達している. また, 関数の性質によって展開の項数が nmax 以内では収束しないこともあり, この場合 icon を 20000 にして処理を打ち切る. このときの係数の値は, それまでに得られている近似値であり, 級数の精度は保証されない.

なお, 本関数ではいずれの場合にも, 級数の絶対誤差の推定値を引数 err に出力するので精度の目安にすることができる.

c) 引数 nmin 及び nmax

nmin, nmax が $2^k + 1$ ($k \geq 0$) の値でない場合, それらを超えない最大の $2^k + 1$ の値とみなす. ただし, $nmax < 5$ の場合, $nmax = 5$ とみなす.

d) 引数 tab

本関数を繰り返し呼ぶ場合, 必要な三角関数表は, 1 回だけしか計算されない. すなわち, 三角関数表に不足が生じた場合, その都度不足分だけ計算し追加される. したがって, 引数 tab の内容は, 常に保存したまま呼び出すこと.

e) 本関数の使用方法

関数 $f(t)$ が単に周期関数であって、その cosine 変換を計算する場合、偶関数 $(f(t) + f(-t))/2$ に対して本関数を適用すればよい。

$f(t)$ が周期をもたず絶対積分可能なときの cosine 変換については“富士通 SSL II 使用手引書”の FCOSF の項目を参照のこと。

4. 使用例

助変数 ω を含む偶関数の積分

$$F(x) = \int_0^x \frac{\cos \omega t}{\sqrt{1 + \cos^2 \omega t}} dt$$

の値を求める。ただし $\omega = \pi/4$ 。

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 257 /* default value */

double truefun(double t); /* prototyptpe for check function */
double fun(double t); /* integral function prototype */
double w; /* auxiliary variable for function fun */

MAIN__()
{
    int ierr, icon;
    int i, n, nmin, nmax;
    double epsa, epsr, err, th, pi, v, f, q, h;
    double b[NMAX], tab[(NMAX-3)/2];

    /* initialize data */
    epsa = 0.5e-4;
    epsr = epsa;
    nmin = 0; /* default value */
    nmax = NMAX;
    pi = 2*asin(1);
    w = pi/4;
    th = pi/w;
    /* expand integral function as cosine series */
    ierr = c_dfcosf(th, fun, epsa, epsr, nmin, nmax, b, &n, &err, tab, &icon);
    if (icon >= 20000) {
        printf("ERROR: icon = %4i\n", icon);
        exit(1);
    }
    /* integrate termwise */
    for (i=1; i<n; i++)
        b[i] = b[i]/(i*w);
    /* now evaluate cosine series to give integral */
    h = th/10;
    printf(" v      f      exact  \n");
    for (i=1; i<=10; i++) {
        v = i*h;
        ierr = c_desinp(th, b, n, v, &f, &icon);
        if (icon != 0) {
            printf("ERROR: icon = %4i\n", icon);
            exit(1);
        }
        q = truefun(v); /* exact integral */
        printf("%4.2f %12.6e %12.6e\n", v, f, q);
    }
    return(0);
}

/* function to integrate */
double fun(double t)
{
    double p;
```

```
    p = cos(w*t);
    return p/sqrt(1+p*p);
}

/* exact integral function */
double truefun(double t)
{
    return asin(sin(w*t)*sqrt(0.5))/w;
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”のFCOSFの項目及び[129]を参照のこと。

c_dfcosm

離散型 cosine 変換 (中点公式, 2 基底 FFT)

ierr = c_dfcosm(a, n, isn, tab, &icon);

1. 機能

周期 2π の偶関数 $x(t)$ の半周期を等分割した n 個の標本 $\{x_{j+1/2}\}$

$$x_{j+1/2} = x\left(\frac{\pi}{n}\left(j + \frac{1}{2}\right)\right), j = 0, 1, \dots, n-1$$

が与えられたとき, 中点公式による離散型 cosine 変換, 又はその逆変換を高速変換手法 (FFT)により行う. ただし, $n = 2^\ell$ ($\ell: 0$ 又は, 正整数)であること.

a) cosine 変換

$\{x_{j+1/2}\}$ を入力し, (1)で定義する変換を行い, フーリエ係数 $\{\frac{n}{2}a_k\}$ を求める.

$$\frac{n}{2}a_k = \sum_{j=0}^{n-1} x_{j+1/2} \cos \frac{\pi}{n} k \left(j + \frac{1}{2}\right), k = 0, 1, \dots, n-1 \quad (1)$$

b) cosine 逆変換

$\{a_k\}$ を入力し, (2)で定義する変換を行い, フーリエ級数の値 $\{x_{j+1/2}\}$ を求める.

$$x_{j+1/2} = \sum_{k=0}^{n-1} ' a_k \cos \frac{\pi}{n} k \left(j + \frac{1}{2}\right), j = 0, 1, \dots, n-1 \quad (2)$$

ここで $\sum '$ は, 初項だけを $1/2$ 倍して和をとることを意味する.

2. 引数

呼出し形式:

```
ierr = c_dfcosm(a, n, isn, tab, &icon);
```

引数の説明:

a	double a[n]	入力	$\{x_{j+1/2}\}$ 又は $\{a_k\}$.
		出力	$\{\frac{n}{2}a_k\}$ 又は $\{x_{j+1/2}\}$.
n	int	入力	標本数 n .
isn	int	入力	変換か逆変換かを指定する. isn = 1: 変換. isn = -1: 逆変換.
tab	double tab[n-1]	出力	変換で使用された三角関数表が格納される. ("使用上の注意" b)参照)
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	次のいずれかであった. • isn ≠ 1 又は -1 • n ≠ 2^ℓ (ℓ:0 又は 正整数)	処理を打ち切る.

3. 使用上の注意

a) 一般的なフーリエ変換の定義

中点公式による離散型 cosine 変換及び, 逆変換は, 一般的に (3), (4) で定義される.

$$a_k = \frac{2}{n} \sum_{j=0}^{n-1} x_{j+1/2} \cos \frac{\pi}{n} k \left(j + \frac{1}{2} \right), k = 0, 1, \dots, n-1 \quad (3)$$

$$x_{j+1/2} = \sum_{k=0}^{n-1} a_k \cos \frac{\pi}{n} k \left(j + \frac{1}{2} \right), j = 0, 1, \dots, n-1 \quad (4)$$

本関数では, (3), (4) の左辺に対応して $\{\frac{n}{2} a_k\}$, $\{x_{j+1/2}\}$ を求めるので, 結果の正規化は必要に応じて行うこと.

b) 引数 **tab**

同一項数の下で本関数を繰り返し呼ぶ場合, 三角関数表の計算は最初の 1 回だけ行われる. したがって 2 回目以降, 引数 **tab** の内容は保存したまま呼び出すこと. 一方, 変換の項数が異なる場合でも, 三角関数表に不足が生じた場合だけ, その都度不足分を計算し表に追加するので効率がよい.

4. 使用例

n 個の標本を本関数により変換したのち正規化し, 離散型フーリエ係数を求める. 引続き逆変換し元のデータと比較することにより結果を確認する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 512

MAIN__()
{
    int ierr, icon;
    double phai, ran, eps, cn;
    double a[NMAX], b[NMAX], tab[NMAX-1];
    int i, n, isn;

    /* generate initial data */
    n = NMAX;
    phai = (sqrt(5.0)-1.0)/2;
    for (i=0;i<n;i++) {
        ran = (i+1)*phai;
        a[i] = ran - (int)ran;
    }
    for (i=0;i<n;i++)
        b[i] = a[i];
    /* perform normal transform */
    isn = 1;
    ierr = c_dfcosm(a, n, isn, tab, &icon);
    if (icon != 0) {
        printf("ERROR: c_dfcosm failed with icon = %d\n", icon);
        exit(1);
    }
    /* normalize */
    cn = 2.0/n;
    for (i=0;i<n;i++)
```

```
    a[i] = cn*a[i];
/* perform inverse transform */
    isn = -1;
    ierr = c_dfcosm(a, n, isn, tab, &icon);
    if (icon != 0) {
        printf("ERROR: c_dfcosm failed with icon = %d\n", icon);
        exit(1);
    }
/* check results */
    eps = 1e-6;
    for (i=0;i<n;i++)
        if (fabs((a[i] - b[i])/b[i]) > eps) {
            printf("WARNING: result inaccurate\n");
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の FCOSM の項目及び[131]を参照のこと.

c_dfnsinf

奇関数の sine 級数展開 (関数入力, 高速 sine 変換)

```
ierr = c_dfnsinf(th, fun, epsa, epsr, nmin,
                 nmax, b, &n, &err, tab, &icon);
```

1. 機能

任意の周期 $2T$ をもつ滑らかな奇関数 $f(t)$ を要求精度 $\varepsilon_a, \varepsilon_r$ に基づき sine 級数展開する. すなわち,

$$\left| f(t) - \sum_{k=0}^{n-1} b_k \sin \frac{\pi}{T} kt \right| \leq \max\{\varepsilon_a, \varepsilon_r \|f\|\} \quad (1)$$

を満たす n 個の係数 b_k を求める. b_k は自明の係数 $b_0 = 0$ を含むので, 実質的な係数の個数は $n-1$ である. 関数のノルム $\|f\|$ は, 半周期 $[0, T]$ での関数の標本点

$$t_j = \frac{T}{n-1} j, \quad j = 0, 1, \dots, n-1$$

における関数値を用いて

$$\|f\| = \max_{0 \leq j \leq n-1} |f(t_j)|$$

と定義する. ただし, $T > 0, \varepsilon_a \geq 0, \varepsilon_r \geq 0$ であること.

2. 引数

呼出し形式:

```
ierr = c_dfnsinf(th, fun, epsa, epsr, nmin, nmax, b, &n, &err, tab, &icon);
```

引数の説明:

th	double	入力	関数 $f(t)$ の半周期 T .
fun	function	入力	展開しようとする関数 $f(t)$ を計算するユーザ関数名.(“使用例”参照) プロトタイプ宣言: double fun(double t);
epsa	double	入力	級数展開の絶対誤差の上限 ε_a . (“使用上の注意” b)参照)
epsr	double	入力	級数展開の相対誤差の上限 ε_r . (“使用上の注意” b)参照)
nmin	int	入力	級数展開の項数 n の下限 (≥ 0). 2^k の値で与える ($k \geq 0$). 標準値は 8. (“使用上の注意” c)参照)
nmax	int	入力	級数の項数 n の上限 ($\geq nmin$). 2^k の値で与える ($k \geq 0$). 標準値は 256. (“使用上の注意” c)参照)
b	double b[nmax]	出力	係数 b_k . $b[k] = b_k, k = 0, 1, \dots, n-1$ のように格納される.
n	int	出力	級数展開の項数 $n (\geq 4)$. $n = 2^k (k \geq 2)$

err	double	出力	級数展開の絶対誤差の推定値. (“使用上の注意” b)参照)
tab	double	出力	級数展開で使用された三角関数表が格納される.
	tab[<i>Tablen</i>]		<i>Tablen</i> = max{3, <i>nmax</i> /2-1}. (“使用上の注意” d)参照)
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	丸め誤差のために要求精度が得られなかった. 要求精度が厳しすぎる.	b には得られた係数を出力する. 級数展開の誤差は達成可能な限度まで達している.
20000	級数展開の項数が, その上限に達しても要求精度が得られなかった.	処理を打ち切る. b には, そのときまでに得られた係数を出力する. また, err には, この時点での絶対誤差の推定値を出力する.
30000	次のいずれかであった. • $th \leq 0$ • $epsa < 0$ • $epsr < 0$ • $nmin < 0$ • $nmax < nmin$	処理を打ち切る.

3. 使用上の注意

a) 精度

展開の項数 n の増大とともに, 誤差が小さくなる度合は, 全区間 $(-\infty, \infty)$ における関数 $f(t)$ の滑らかさに著しく影響される. $f(t)$ が解析的周期関数ならば, 誤差は指数関数的オーダー $O(r^n)$, $0 < r < 1$ で減少し, $f(t)$ が k 回連続微分可能ならば, 有理式のオーダー $O(n^{-k})$ で減少する. $k = 0, 1$ の場合, 展開項数が多くなり, 誤差の推定も正確さを期しがたい. したがって, 本ルーチンを適用する関数は, 少なくとも 2 回連続微分可能であることが望ましい.

b) 引数 $epsa$ 及び $epsr$

本関数は, 与えられた $\varepsilon_a, \varepsilon_r$ に対して(1)を満たす級数を求める. したがって, $\varepsilon_r = 0$ とおけば絶対誤差 ε_a 以内の, また $\varepsilon_a = 0$ とおけば相対誤差 ε_r 以内の精度で級数展開をすることになる. しかし, $\varepsilon_a, \varepsilon_r$ の値によってはこの目的が達成されないこともある. 例えば, 関数の計算精度に比べて, $\varepsilon_a, \varepsilon_r$ が小さすぎる場合には展開の項数がその上限に達しない場合でも, 丸め誤差の影響の方が大きくなり, これ以上計算を続行しても無意味である.

このような状態を見出したときは icon を 10000 にして処理を打ち切る. このときの級数の精度は, 使用計算機で達成可能な限度まで達している. また, 関数の性質によって展開の項数が $nmax$ 以内では収束しないこともあり, この場合 icon を 20000 にして処理を打ち切る. このときの係数の値は, それまでに得られている近似値であり, 級数の精度は保証されない.

なお, 本関数ではいずれの場合にも, 級数の絶対誤差の推定値を引数 err に出力するので精度の目安にすることができる.

c) 引数 $nmin$ 及び $nmax$

$nmin, nmax$ が 2^k ($k \geq 0$) の値でない場合, それらを超えない最大の 2^k の値とみなす. ただし, $nmax < 4$ の場合, $nmax = 4$ とみなす.

d) 引数 tab

本関数を繰り返し呼ぶ場合、必要な三角関数表は、1 回だけしか計算されない。すなわち、三角関数表に不足が生じた場合、その都度不足分だけ計算し追加される。したがって、引数 tab の内容は、常に保存したまま呼び出すこと。

e) 本関数の使用方法

関数 $f(t)$ が単に周期関数であって、その sine 変換を計算する場合、奇関数 $(f(t) - f(-t))/2$ に対して本関数を適用すればよい。

$f(t)$ が周期をもたず絶対積分可能なときの sine 変換については“富士通 SSL II 使用手引書”の FSINF の項目を参照のこと。

4. 使用例

助変数 ω を含む奇関数の積分

$$F(x) = \int_0^x \frac{\sin \omega t}{\sqrt{1 + \sin^2 \omega t}} dt$$

の値を求める。ただし $\omega = \pi/4$ 。

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 257 /* default value */

double truefun(double t); /* prototype for check function */
double fun(double t); /* integral function prototype */
double w; /* auxiliary variable for function fun */

MAIN__()
{
    int ierr, icon;
    int i, n, nmin, nmax;
    double epsa, epsr, err, th, pi, v, f, q, h;
    double a[NMAX], tab[(NMAX-3)/2];

    /* initialize data */
    epsa = 0.5e-4;
    epsr = epsa;
    nmin = 0; /* default value */
    nmax = NMAX;
    pi = 2*asin(1);
    w = pi/4;
    th = pi/w;
    /* expand integral function as sine series */
    ierr = c_dfssinf(th, fun, epsa, epsr, nmin, nmax, a, &n, &err, tab, &icon);
    if (icon >= 20000) {
        printf("ERROR: icon = %4i\n", icon);
        exit(1);
    }
    /* integrate termwise */
    for (i=1; i<n; i++)
        a[i] = -a[i]/(i*w);
    /* evaluate cosine series at v=0 to find a0 value */
    v = 0;
    ierr = c_decosp(th, a, n, v, &f, &icon);
    if (icon != 0) {
        printf("ERROR: icon = %4i\n", icon);
        exit(1);
    }
    a[0] = -f*2; /* notice factor of 2 */
    /* now evaluate cosine series to give integral */
    h = th/10;
    printf(" v      f      exact  \n");
```



```
for (i=1;i<=10;i++) {
    v = i*h;
    ierr = c_decosp(th, a, n, v, &f, &icon);
    if (icon != 0) {
        printf("ERROR: icon = %4i\n", icon);
        exit(1);
    }
    q = truefun(v); /* exact integral */
    printf("%4.2f %12.6e %12.6e\n", v, f, q);
}
return(0);
}

/* function to integrate */
double fun(double t)
{
    double p;
    p = sin(w*t);
    return p/sqrt(1+p*p);
}

/* exact integral function */
double truefun(double t)
{
    double pi;
    pi = 2*asin(1);
    return (pi/4-asin(cos(w*t)*sqrt(0.5)))/w;
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の FSINF の項目及び[129]を参照のこと.

c_dfsm

離散型 sine 変換 (中点公式, 2 基底 FFT)

```
ierr = c_dfsm(a, n, isn, tab, &icon);
```

1. 機能

周期 2π の偶関数 $x(t)$ の半周期を等分割した n 個の標本 $\{x_{j+1/2}\}$,

$$x_{j+1/2} = x\left(\frac{\pi}{2}\left(j + \frac{1}{2}\right)\right), j = 0, 1, \dots, n-1$$

が与えられたとき, 中点公式による離散型 sine 変換, 又はその逆変換を高速変換手法 (FFT)により行う. ただし, $n = 2^\ell$ ($\ell:0$ 又は, 正整数)であること.

a) sine 変換

$\{x_{j+1/2}\}$ を入力し, (1)で定義する変換を行い, フーリエ係数 $\{\frac{n}{2}b_k\}$ を求める.

$$\frac{n}{2}b_k = \sum_{j=0}^{n-1} x_{j+1/2} \sin \frac{\pi}{n} k \left(j + \frac{1}{2}\right), k = 1, 2, \dots, n \quad (1)$$

b) sine 逆変換

$\{b_k\}$ を入力し, (2)で定義する変換を行い, フーリエ級数の値 $\{x_{j+1/2}\}$ を求める.

$$x_{j+1/2} = \sum_{k=1}^{n-1} b_k \sin \frac{\pi}{n} k \left(j + \frac{1}{2}\right) + \frac{1}{2} b_n \sin \pi \left(j + \frac{1}{2}\right), j = 0, 1, \dots, n-1 \quad (2)$$

2. 引数

呼出し形式:

```
ierr = c_dfsm(a, n, isn, tab, &icon);
```

引数の説明:

a	double a[n]	入力	$\{x_{j+1/2}\}$ 又は $\{b_k\}$.
		出力	$\{\frac{n}{2}b_k\}$ 又は $\{x_{j+1/2}\}$.
n	int	入力	標本数 n .
isn	int	入力	変換か逆変換かを指定する. isn = 1: 変換. isn = -1: 逆変換.
tab	double tab[n-1]	出力	変換で使用された三角関数表が格納される. (“使用上の注意” c)参照)
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	次のいずれかであった. • isn ≠ 1 又は -1 • n ≠ 2 ^ℓ (ℓ:0 又は 正整数)	処理を打ち切る.

3. 使用上の注意

a) 一般的なフーリエ変換の定義

中点公式による離散型 sine 変換及び, 逆変換は, 一般的に (3), (4) で定義される.

$$b_k = \frac{2}{n} \sum_{j=0}^{n-1} x_{j+1/2} \sin \frac{\pi}{n} k \left(j + \frac{1}{2} \right), k = 1, 2, \dots, n \quad (3)$$

$$x_{j+1/2} = \sum_{k=1}^{n-1} b_k \sin \frac{\pi}{n} k \left(j + \frac{1}{2} \right) + \frac{1}{2} b_n \sin \pi \left(j + \frac{1}{2} \right), j = 0, 1, \dots, n-1 \quad (4)$$

本関数では, (3), (4) の左辺に対応して $\{\frac{n}{2} b_k\}, \{x_{j+1/2}\}$ を求めるので, 結果の正規化は必要に応じて行うこと.

b) 三角多項式の計算

逆変換により, n 次の三角多項式

$$x(t) = b_1 \sin t + b_2 \sin 2t + \dots + b_n \sin nt$$

の値 $x\left(\frac{\pi}{n}\left(j+\frac{1}{2}\right)\right)$ を求める場合は, 最高次の係数 b_n はあらかじめ 2 倍する必要がある.

c) 引数 tab

同一項数の下で本関数を繰り返し呼ぶ場合, 三角関数表の計算は最初の 1 回だけ行われる. したがって 2 回目以降, 引数 tab の内容は保存したまま呼び出すこと. 一方, 変換の項数が異なる場合でも, 三角関数表に不足が生じた場合だけ, その都度不足分を計算し表に追加するので効率がよい.

4. 使用例

n 個の標本を本関数により変換したのち正規化し, 離散型フーリエ係数を求める. 引続き逆変換し元のデータと比較することにより結果を確認する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 512

MAIN__()
{
    int ierr, icon;
    double phai, ran, eps, cn;
    double a[NMAX], b[NMAX], tab[NMAX-1];
    int i, n, isn;

    /* generate initial data */
    n = NMAX;
    phai = (sqrt(5.0)-1.0)/2;
    for (i=0; i<n; i++) {
        ran = (i+1)*phai;
        a[i] = ran - (int)ran;
    }
    for (i=0; i<n; i++)
        b[i] = a[i];
    /* perform normal transform */
    isn = 1;
    ierr = c_dfssnm(a, n, isn, tab, &icon);
    if (icon != 0) {
        printf("ERROR: c_dfssnm failed with icon = %d\n", icon);
        exit(1);
    }
    /* normalize */
    cn = 2.0/n;
    for (i=0; i<n; i++)
        a[i] = cn*a[i];
    /* perform inverse transform */
```

```
    isn = -1;
    ierr = c_dfssnm(a, n, isn, tab, &icon);
    if (icon != 0) {
        printf("ERROR: c_dfssnm failed with icon = %d\n", icon);
        exit(1);
    }
    /* check results */
    eps = 1e-6;
    for (i=0; i<n; i++)
        if (fabs((a[i] - b[i])/b[i]) > eps) {
            printf("WARNING: result inaccurate\n");
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”のFSINMの項目及び[131]を参照のこと。

c_dgbseg

実対称バンド行列の一般固有値及び固有ベクトル(ジェニングス法)

ierr = c_dgbseg(a, b, n, nh, m, epsz, epst, lm, e, ev, k, &it, vw, &icon);

1. 機能

次数 n , バンド幅 h の実対称バンド行列 $\mathbf{A}$ と $\mathbf{B}$ とに対する一般固有値問題,

$$\mathbf{Ax} = \lambda \mathbf{Bx}$$

の, 絶対値の大きい方から又は小さい方からの m 個の固有値及びそれらに対応する固有ベクトルを, m 個の与えられた初期ベクトルをもとにして, ジェニングスのベクトル加速法を伴う, ジェニングスの同時反復法によって求める. ただし, 絶対値の大きい方から求めるときは $\mathbf{B}$ が, 小さい方から求める時は $\mathbf{A}$ が正値行列でなければならない. なお固有ベクトルは, それぞれの場合に応じて,

$$\mathbf{X}^T \mathbf{BX} = \mathbf{I}$$

又は,

$$\mathbf{X}^T \mathbf{AX} = \mathbf{I}$$

の意味で正規化されている. $1 \leq m \ll n$, $0 \leq h \ll n$ であること.

2. 引数

呼出し形式:

```
ierr = c_dgbseg(a, b, n, nh, m, epsz, epst, lm, e, (double *)ev, k, &it, vw,
                &icon);
```

引数の説明:

a	double a[Alen]	入力	実対称バンド行列 $\mathbf{A}$. 対称バンド行列用圧縮格納法. 格納法の詳細については“1.3.4 バンド行列の格納方法”を参照のこと. $Alen = n(h+1) - h(h+1)/2$
		出力	絶対値の小さい方から求めるときには, 演算後, その内容は保存されない. (“使用上の注意” a)参照)
b	double b[Blen]	入力	実対称バンド行列 $\mathbf{B}$. 対称バンド行列用圧縮格納法. 格納法の詳細については“1.3.4 バンド行列の格納方法”を参照のこと. $Blen = n(h+1) - h(h+1)/2$
		出力	演算後, 内容は保存されない. (“使用上の注意” a)参照)
n	int	入力	行列 $\mathbf{A}, \mathbf{B}$ の次数 n .
nh	int	入力	行列 $\mathbf{A}, \mathbf{B}$ のバンド幅 h . (“使用上の注意” b)参照)
m	int	入力	求める固有値及び固有ベクトルの個数 m . $m > 0$ のときは絶対値の大きい方から求める. $m < 0$ のときは絶対値の小さい方から求める. (“使用上の注意” c)参照)
epsz	double	入力	$\mathbf{A}$ 又は $\mathbf{B}$ の LL^T 分解におけるピボットの相対零判定値. 零又は負のときは標準値が設定される. (“使用上の注意” d)参照)
epst	double	入力	固有ベクトルの収束判定に用いられる定数 ε . 零又は負のときは標準値が設定される. (“使用上の注意” e)参照)
lm	int	入力	反復回数の上限. 反復回数がこれを超えると処理を中断する. (“使用上の注意” f)参照)
e	double e[m]	出力	固有値. 指定された順序に格納される.

ev	double ev[m+2][k]	入力	始めの m 行に, 行方向に格納された m 個の初期ベクトル. (“使用上の注意” g)参照)
		出力	固有ベクトル. 始めの m 行に行方向に格納される.
k	int	入力	ev の整合寸法 ($\geq n$).
it	int	出力	固有値固有ベクトルが求まるまでの反復回数.
vw	double vw[Vwlen]	作業領域	$Vwlen = 2n + m(3m+1)/2$
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
20000	反復回数が上限 lm を超えた.	処理を終了する. e と ev には, そのときまでに得られた固有値と固有ベクトルの近似値が入っている.
25000	反復ごとの固有ベクトルの直交化処理が不可能となった.	処理を打ち切る.
28000	行列 A 又は B が正値行列でない.	処理を打ち切る.
29000	行列 A 又は B が正則行列でない.	処理を打ち切る.
30000	次のいずれかであった. • $nh < 0$ 又は $nh \geq n$ • $k < n$ • $m = 0$ 又は $m > n$	処理を打ち切る.

3. 使用上の注意

a) 引数 a 及び b

絶対値の小さい方から求めた場合, 行列 **B** の内容は配列 a に保存される. よって, 行列 **B** が同一な複数组の一般固有値問題を続けて扱う場合には, これを利用できる.

b) バンド幅

A と **B** のバンド幅は同一であること. もし, **A** と **B** のバンド幅が異なる場合は大きいバンド幅に合わせるために, 引数 a 若しくは b へ拡大された分だけ零を追加すること.

c) 引数 m

固有値及び固有ベクトルの個数 m は, n に比べて小さく, 例えば $m/n < 1/10$ 程度にとるのがよい. 固有値を絶対値の大きい方から, あるいは小さい方から番号をつけて, $\lambda_1, \lambda_2, \dots, \lambda_n$ とするとき, もし選んだ m が $|\lambda_{m+1} / \lambda_m| \ll 1$ あるいは $|\lambda_{m+1} / \lambda_m| \gg 1$ を満足するならば, 収束が速い.

d) 引数 epsz

引数 epsz の標準値は, 丸め誤差の単位 μ としたとき, 16μ である.

本関数において epsz に 10^8 を設定したとすると, 行列 **A** 又は **B** の LL^T 分解の過程でピボットの値が 10 進 s 桁以上の桁落ちを生じた場合に, そのピボットを零と見なしコンディションコードを設定し(icon = 29000)処理を打ち切る. なお, ピボットが小さくなくても計算を続行させる場合には, epsz に非常に小さな値(例えば 10^{-70})を与えればよい.

行列 **A** 又は **B** の LL^T 分解の過程でピボットが負となった場合, 行列を非正則であると見なしコンディションコードを設定し(icon = 28000)処理を打ち切る.

e) 引数 epst

引数 epst は $\|x\|_2 = 1$ の意味で正規化された固有ベクトルの各成分の収束を判定するために用いられる。固有ベクトルが判定定数 ε に対して収束したとき、固有値は少なくとも $\|A\| \cdot \varepsilon$ の精度で、そして多くの場合 $\|A\| \cdot \varepsilon$ よりも高い精度で収束している。このことを考慮して幾分大きめに epst を選ぶのがよい。丸め誤差の単位を μ とするとき、標準値は 16μ である。しかし、固有値が密集した問題では、収束が得られないことがあるので、そのような場合には $\varepsilon \geq 100\mu$ ぐらいにとるのが安全である。

f) 引数 lm

反復回数の上限 lm は、何らかの理由で収束が達成されないとき、強制的に反復を中断するためのものである。要求精度や固有値の密集の度合などを考慮して定められるべきものであるが、500～1000 あたりが妥当な標準値である。

g) 初期固有ベクトル

初期固有ベクトルは、求めようとする固有値に対応する固有ベクトルの良い近似であることが望ましい。しかしそのような近似ベクトルが得られないときには、単位行列 **I** の始めの m 個の行ベクトルを初期ベクトルとして採用するのが標準的である。

4. 使用例

実対称バンド行列 **A** と **B** とに対する、一般固有値問題 $Ax = \lambda Bx$ の固有値と固有ベクトルを求める。

```
#include <stdlib.h>
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

#define max(a,b) ((a) > (b) ? (a) : (b))

#define NMAX 5
#define NHMAX 2

MAIN__()
{
    int ierr, icon;
    int n, m, nh, i, j, k, ij, lm, it, jmin;
    double a[NMAX*(NHMAX+1)-NHMAX*(NHMAX+1)/2];
    double b[NMAX*(NHMAX+1)-NHMAX*(NHMAX+1)/2];
    double e[NMAX], ev[NMAX+2][NMAX], vw[2*NMAX+NMAX*(3*NMAX+1)/2], epsz, epst;

    /* initialize matrix */
    n = NMAX;
    nh = NHMAX;
    ij = 0;
    for (i=0; i<n; i++) {
        jmin = max(i-nh, 0);
        for (j=jmin; j<i; j++) {
            a[ij] = n-i;
            b[ij++] = 0;
        }
        a[ij] = n-i;
        b[ij++] = 1;
    }
    k = NMAX;
    m = n;
    /* initialize m eigenvectors */
    for (i=0; i<m; i++)
        for (j=0; j<n; j++)
            if (i == j) ev[i][j] = 1;
            else ev[i][j] = 0;
    lm = 1000;
    epsz = 0;
    epst = 0;
    /* find eigenvalues and eigenvectors */
    ierr = c_dgbseg(a, b, n, nh, m, epsz, epst, lm,
                   e, (double*)ev, k, &it, vw, &icon);
    if (icon >= 20000) {
```

```
        printf("ERROR: c_dgbseg failed with icon = %d\n", icon);
        exit(1);
    }
    /* print eigenvalues and eigenvectors */
    for (i=0;i<m;i++) {
        printf("e-value %d: %10.4f\n",i+1,e[i]);
        printf("e-vector:");
        for (j=0;j<n;j++)
            printf("%7.4f ",ev[i][j]);
        printf("\n");
    }
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の GBSEG の項目及び[61], [128]を参照のこと.

c_dgcheb

チェビシェフ級数の導関数

ierr = c_dgcheb(a, b, c, &n, &icon);

1. 機能

区間 $[a, b]$ で定義された n 項のチェビシェフ級数,

$$f(x) = \sum_{k=0}^{n-1} c_k T_k \left(\frac{2x - (b+a)}{b-a} \right)$$

が与えられたとき, その導関数をチェビシェフ級数

$$f'(x) = \sum_{k=0}^{n-2} c'_k T_k \left(\frac{2x - (b+a)}{b-a} \right)$$

に展開し, その係数 c'_k を求める ($k=0,1,\dots,n-2$). ここで, $\sum'$ は初項を $1/2$ 倍して和をとることを意味する. ただし, $a \neq b$, $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dgcheb(a, b, c, &n, &icon);
```

引数の説明:

a	double	入力	チェビシェフ級数の定義域の下限 a .
b	double	入力	チェビシェフ級数の定義域の上限 b .
c	double c[n]	入力	係数 c_k . $c[k] = c_k$, $k=0,1,\dots,n-1$ のように格納する.
		出力	導関数の係数 c'_k . $c[k] = c'_k$, $k=0,1,\dots,n-2$ のように格納される.
n	int	入力	項数 n .
		出力	導関数の項数 $n-1$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	次のいずれかであった. • $n < 1$ • $a = b$	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

任意の関数の導関数を求める場合, チェビシェフ級数展開する関数 c_dfcheb と, 本関数を順次呼び出せばよい. 更に, 任意の点における微係数を求める場合は, チェビシェフ級数の求和の関数 c_decheb を続けて呼び出せばよい. (“使用例” 参照)

b) 高階の導関数

高階の導関数を必要とする場合, 本関数を繰り返し呼び出せばよい.

導関数の誤差は、末尾 2 項の係数の絶対値和で推定できるが、高階になる程誤差が大きくなることを考慮する必要がある。

4. 使用例

指数関数 $f(x)=e^x$ をチェビシェフ級数展開し、その導関数を求める。

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 257 /* default value */

double fun(double x); /* function prototype */

MAIN__()
{
    int ierr, icon;
    int i, n, nmin, nmax;
    double epsa, epsr, err, a, b, pi, v, f, h;
    double c[NMAX], tab[(NMAX-3)/2];

    /* initialize data */
    epsr = 5e-5;
    epsa = 0;
    nmin = 9; /* default value */
    nmax = NMAX;
    pi = 2*asin(1);
    a = -2;
    b = 2;
    /* expand function as Chebyshev series */
    ierr = c_dfcheb(a, b, fun, epsa, epsr, nmin, nmax, c, &n, &err, tab, &icon);
    if (icon >= 20000) {
        printf("ERROR: icon = %4i\n", icon);
        exit(1);
    }
    /* now calculate derivative */
    ierr = c_dgcheb(a, b, c, &n, &icon);
    if (icon != 0) {
        printf("ERROR: icon = %4i\n", icon);
        exit(1);
    }
    /* now evaluate Chebyshev series at points */
    h = 0.05;
    printf(" v          differential          error   \n");
    for (i=0; i<=80; i++) {
        v = a+i*h;
        ierr = c_decheb(a, b, c, n, v, &f, &icon);
        if (icon != 0) {
            printf("ERROR: icon = %4i\n", icon);
            exit(1);
        }
        err = fun(v) - f;
        printf("%5.2f %12.5e %12.5e\n", v, f, err);
    }
    return(0);
}

/* function to expand */
double fun(double x)
{
    double sum, xn, xp, term, eps;
    int n;
    eps = 1e-7; /* approx. amach */
    sum = 1;
    xp = x;
    xn = 1;
    n = 1;
    while (1) {
        term = xp/xn;
        sum = sum+term;
        if (fabs(term) <= fabs(sum)*eps) break;
        n = n+1;
    }
}
```

```
        xp = xp*x;  
        xn = xn*n;  
    }  
    return (sum);  
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の GCHEB の項目を参照のこと.

c_dginv

実行列の一般逆行列 (特異値分解法)

```
ierr = c_dginv(a, ka, m, n, sig, v, kv, eps,
               vw, &icon);
```

1. 機能

$m \times n$ の実行列 **A** の一般逆行列 $\mathbf{A}^+$ を特異値分解法によって求める. ただし, $m \geq 1, n \geq 1$ なること.

2. 引数

呼出し形式:

```
ierr = c_dginv((double*)a, ka, m, n, sig, (double*)v, kv, eps, vw, &icon);
```

引数の説明:

a	double a[m][ka]	入力	行列 A .
		出力	一般逆行列 $\mathbf{A}^+$ の転置行列. (“使用上の注意”a)参照).
ka	int	入力	配列 a の整合寸法 ($\geq n$).
m	int	入力	行列 A の行数 m .
n	int	入力	行列 A の列数, 行列 V の行数 n .
sig	double sig[n]	出力	行列 A の特異値. (“使用上の注意”b)参照).
v	double v[n][kv]	出力	特異値分解による直交変換行列.
kv	int	入力	配列 v の整合寸法 ($\geq \min(m+1, n)$).
eps	double	入力	特異値の相対零判定値 (≥ 0.0). 0.0 のときは標準値が採用される. (“使用上の注意”c)参照).
vw	double vw[n]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
15000	ある特異値が求められなかった.	処理を打ち切る.
29000	作業用配列を確保できなかった.	処理を打ち切る.
30000	次のいずれかであった. • ka < n • m < 1 • n < 1 • kv < min(m+1,n) • eps < 0	処理を打ち切る.

3. 使用上の注意

a) **a** について

配列 **a** に出力されるのは **A** の一般逆行列の転置行列 $(\mathbf{A}^+)^T$ であるので注意すること。

b) **sig** について

特異値はすべて非負で、大きさの減少する順に格納される。icon = 15000 の場合には、求められなかった特異値は-1 とし、大小順に整列しない。

c) **eps** について

入力引数 **eps** は **A** の階数を決定するための重要な役目を果たすので、その選択は慎重に行わなければならない。

この判定基準 **eps** より小さい特異値は 0 とみなされる。eps として 0.0 が入力されたときは、 $\text{eps} = 16\mu$ を標準値として用いる。ただし、 μ は丸め誤差の単位である。0 より小さい値が入力されたときは icon = 30000 となる。

d) 最小二乗解

連立一次方程式 $\mathbf{Ax} = \mathbf{b}$ の最小二乗最小ノルム解は、一般化逆行列 $\mathbf{A}^+$ によって、 $\mathbf{x} = \mathbf{A}^+\mathbf{b}$ として表されるが、この目的のためには、専用関数 `c_dlaxlm` を用いる方がはるかに有利である。結局、本関数は、一般逆行列 $\mathbf{A}^+$ 自体が必要な場合以外は用いないほうがよい。

4. 使用例

行列 **A** の一般逆行列を求め、結果を出力する。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define MMAX 7
#define NMAX 5

MAIN__()
{
    int ierr, icon;
    int m, n, i, j, ka, kv;
    double a[MMAX][NMAX], sig[NMAX], v[NMAX][NMAX], vw[NMAX], eps;

    /* initialize system */
    m = MMAX;
    n = NMAX;
    for (i=0; i<n; i++)
        for (j=i; j<n; j++) {
            a[i][j] = n-j;
            a[j][i] = n-j;
        }
    for (i=n; i<m; i++)
        for (j=0; j<n; j++) {
            a[i][j] = 0;
            if (i%n == j) a[i][j] = 1;
        }
    ka = NMAX;
    kv = NMAX;
    eps = 0;
    /* generalized inverse */
    ierr = c_dginv((double*)a, ka, m, n, sig, (double*)v, kv, eps, vw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dginv failed with icon = %d\n", icon);
        exit(1);
    }
    /* print transposed generalized inverse */
}
```

```
    for (i=0;i<m;i++) {  
        for (j=0;j<n;j++)  
            printf("%7.4f ",a[i][j]);  
        printf("\n");  
    }  
    return(0);  
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の GINV の項目及び[41]を参照のこと。

c_dgsbk

一般形の固有ベクトルへの逆変換 (実対称行列の一般固有値問題)

ierr = c_dgsbk(ev, k, n, m, b, &icon);

1. 機能

n 次の実対称行列 $\mathbf{S}$ の m 個の固有ベクトル $\mathbf{y}_j$ ($j = 1, 2, \dots, m$) を, 一般固有値問題 $\mathbf{Ax} = \lambda \mathbf{Bx}$ の固有ベクトル $\mathbf{x}_j$ ($j = 1, 2, \dots, m$) へ変更する. ただし, $\mathbf{S}$ は $\mathbf{B} = \mathbf{LL}^T$ として,

$$\mathbf{S} = \mathbf{L}^{-1} \mathbf{A} \mathbf{L}^{-T}$$

により得られたものであること. ここで $\mathbf{L}$ は下三角行列である. $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dgsbk((double *)ev, k, n, m, b, &icon);
```

引数の説明:

ev	double ev[m][k]	入力 出力	実対称行列 $\mathbf{S}$ の m 個の固有ベクトル. 一般固有値問題 $\mathbf{Ax} = \lambda \mathbf{Bx}$ の固有ベクトル. (“使用上の注意” a) 参照)
k	int	入力	入力 ev の整合寸法 ($\geq n$).
n	int	入力	実対称行列 $\mathbf{S}$, $\mathbf{A}$ 及び $\mathbf{B}$ の次数 n .
m	int	入力	固有ベクトルの個数 m . $m < 0$ のときはその絶対値をとって使う.
b	double b[n(n+1)/2]	入力	下三角行列 $\mathbf{L}$. 対称行列用圧縮格納法. 格納法の詳細については “1.3.2 対称行列の格納方法” を参照のこと. (“使用上の注意” b) 参照)
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	$n = 1$	$\text{ev}[0][0] = 1/b[0]$.
30000	次のいずれかであった. • $m = 0$ 又は $ m > n$ • $k < n$	処理を打ち切る.

3. 使用上の注意

a) 固有ベクトル

固有ベクトル $\mathbf{x}_j$ ($j = 1, 2, \dots, m$) は, 入力のベクトル $\mathbf{y}_j$ ($j = 1, 2, \dots, m$) が $\mathbf{Y}^T \mathbf{Y} = \mathbf{I}$ を満足するように正規化されていれば, $\mathbf{X}^T \mathbf{B} \mathbf{X} = \mathbf{I}$ を満足するように出力される. ただし, $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_m]$, $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m]$ である.

b) 引数 b

関数 c_dgschl の出力引数 b は, 本関数の入力引数 b と対応しているのでそのまま使用できる.

4. 使用例

n 次の実対称行列 **A** 及び n 次の正値対称行列 **B** で与えられる一般固有値問題を、一連の関数 `c_dgschl`, `c_dseig1`, `c_dgsbk` により、固有値固有ベクトルを求める。

```
#include <stdio.h>
#include <stdlib.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 5

MAIN__()
{
    int ierr, icon;
    int n, i, j, k, ij, m;
    double a[NMAX*(NMAX+1)/2], b[NMAX*(NMAX+1)/2], vw[2*NMAX];
    double e[NMAX], ev[NMAX][NMAX], epsz;

    /* initialize matrix */
    n = NMAX;
    ij = 0;
    for (i=0;i<n;i++) {
        for (j=0;j<i;j++) {
            a[ij] = n-i;
            b[ij++] = 0;
        }
        a[ij] = n-i;
        b[ij++] = 1;
    }
    /* reduce to standard form */
    epsz = 0;
    ierr = c_dgschl(a, b, n, epsz, &icon);
    if (icon > 10000 ) {
        printf("ERROR: c_dgschl failed with icon = %i\n", icon);
        exit (1);
    }
    /* find eigenvalues and eigenvectors */
    k = NMAX;
    ierr = c_dseig1(a, n, e, (double*)ev, k, &m, vw, &icon);
    if (icon >= 20000 ) {
        printf("ERROR: c_dseig1 failed with icon = %i\n", icon);
        exit (1);
    }
    /* back transformation */
    ierr = c_dgsbk((double*)ev, k, n, m, b, &icon);
    if (icon > 10000 ) {
        printf("ERROR: c_dgsbk failed with icon = %i\n", icon);
        exit (1);
    }
    printf("icon = %i\n", icon);
    /* print eigenvalues and eigenvectors */
    for (i=0;i<m;i++) {
        printf("eigenvalue: %7.4f\n", e[i]);
        printf("eigenvector: ");
        for (j=0;j<n;j++)
            printf("%7.4f ", ev[i][j]);
        printf("\n");
    }
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”のGSBKの項目及び[119]を参照のこと。

c_dgschl

一般形から標準形への変換 (実対称行列の一般固有値問題)

ierr = c_dgschl(a, b, n, epsz, &icon);

1. 機能

n 次の実対称行列 **A**, 及び n 次の正値対称行列 **B** に対して, 一般固有値問題

$$\mathbf{Ax} = \lambda \mathbf{Bx}$$

を標準固有値問題

$$\mathbf{Sy} = \lambda \mathbf{y}$$

に変換する. ここで **S** は実対称行列である. $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dgschl(a, b, n, epsz, &icon);
```

引数の説明:

a	double a[n(n+1)/2]	入力	実対称行列 A . 対称行列用圧縮格納法. 格納法の詳細については“1.3.2 対称行列の格納方法”を参照のこと.
		出力	実対称行列 S . 対称行列用圧縮格納法. 格納法の詳細については“1.3.2 対称行列の格納方法”を参照のこと.
b	double b[n(n+1)/2]	入力	正値対称行列 B . 対称行列用圧縮格納法. 格納法の詳細については“1.3.2 対称行列の格納方法”を参照のこと.
		出力	下三角行列 L ($\mathbf{B} = \mathbf{LL}^T$). 対称行列用圧縮格納法. 格納法の詳細については“1.3.2 対称行列の格納方法”を参照のこと.
n	int	入力	行列の次数 n .
epsz	double	入力	B の $\mathbf{LL}^T$ 分解におけるピボットの相対零判定値. 0.0 又は負を指定すると, 標準値が採用される. (“使用上の注意” 参照)
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	$n = 1$	$a[0] = a[0]/b[0]$ $b[0] = \sqrt{b[0]}$
28000	行列 B の $\mathbf{LL}^T$ 分解において, ピボットが負となった. 入力行列 B は正値ではない.	処理を打ち切る.
29000	行列 B の $\mathbf{LL}^T$ 分解において, ピボットが相対的に零となった. 入力行列 B は非正則の可能性が強い.	処理を打ち切る.
30000	$n < 1$	処理を打ち切る.

3. 使用上の注意

a) 引数 epsz

引数 epsz の標準値は丸め誤差の単位を μ としたとき, $\text{epsz} = 16\mu$ である.

本関数において epsz に 10^{-5} を設定したとすると、正値対称行列 $\mathbf{B}$ の LL^T 分解の過程でピボットの値が 10 進 s 桁以上の桁落ちを生じた場合に、そのピボットを零と見なしコンディションコードを設定し ($\text{ICON}=29000$) 処理を打ち切る。なおピボットが小さくなくても計算を続行させる場合には、 epsz に極小の値(例えば 10^{-70})を与えればよいが、その結果は保証されない。

行列の $\mathbf{B}$ の LL^T 分解の過程でピボットが負となった場合、 $\mathbf{B}$ は正値行列ではない。本関数では、このときコンディションコードを設定し ($\text{ICON}=28000$) 処理を打ち切る。

4. 使用例

n 次の実対称行列 $\mathbf{A}$ 及び n 次の正値対称行列 $\mathbf{B}$ で与えられる一般固有値問題を、一連の関数 `c_dgschl`, `c_dseig1`, `c_dgsbk` により、固有値固有ベクトルを求める。

```
#include <stdio.h>
#include <stdlib.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 5

MAIN__()
{
    int ierr, icon;
    int n, i, j, k, ij, m;
    double a[NMAX*(NMAX+1)/2], b[NMAX*(NMAX+1)/2], vw[2*NMAX];
    double e[NMAX], ev[NMAX][NMAX], epsz;

    /* initialize matrix */
    n = NMAX;
    ij = 0;
    for (i=0; i<n; i++) {
        for (j=0; j<i; j++) {
            a[ij] = n-i;
            b[ij++] = 0;
        }
        a[ij] = n-i;
        b[ij++] = 1;
    }
    /* reduce to standard form */
    epsz = 0;
    ierr = c_dgschl(a, b, n, epsz, &icon);
    if (icon > 10000 ) {
        printf("ERROR: c_dgschl failed with icon = %i\n", icon);
        exit (1);
    }
    /* find eigenvalues and eigenvectors */
    k = NMAX;
    ierr = c_dseig1(a, n, e, (double*)ev, k, &m, vw, &icon);
    if (icon >= 20000 ) {
        printf("ERROR: c_dseig1 failed with icon = %i\n", icon);
        exit (1);
    }
    /* back transformation */
    ierr = c_dgsbk((double*)ev, k, n, m, b, &icon);
    if (icon > 10000 ) {
        printf("ERROR: c_dgsbk failed with icon = %i\n", icon);
        exit (1);
    }
    printf("icon = %i\n", icon);
    /* print eigenvalues and eigenvectors */
    for (i=0; i<m; i++) {
        printf("eigenvalue: %7.4f\n", e[i]);
        printf("eigenvector: ");
        for (j=0; j<n; j++)
            printf("%7.4f ", ev[i][j]);
        printf("\n");
    }
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の GSCHL の項目及び[119]を参照のこと.

c_dhbk1

実行列の固有ベクトルへの逆変換と正規化

<pre>ierr = c_dhbk1(ev, k, n, ind, m, p, pv, dv, &icon);</pre>

1. 機能

n 次の実ヘッセンベルグ行列 $\mathbf{H}$ の m 個の固有ベクトルを, 実行列 $\mathbf{A}$ の固有ベクトルへ逆変換し, $\|\mathbf{x}\|_2 = 1$ となるように正規化する. ただし, $\mathbf{H}$ は $\mathbf{A}$ にハウスホルダー法を適用して得られたものであること. $1 \leq m \leq n$ であること.

2. 引数

呼出し形式:

```
ierr = c_dhbk1((double *)ev, k, n, ind, m, (double *)p, pv, dv, &icon);
```

引数の説明:

ev	double ev[m][k]	入力	実ヘッセンベルグ行列 $\mathbf{H}$ の m 個の固有ベクトル.
		出力	実行行列 $\mathbf{A}$ の固有ベクトル.
k	int	入力	配列 ev, p の整合寸法 ($\geq n$).
n	int	入力	実ヘッセンベルグ行列 $\mathbf{H}$ の次数 n .
ind	int ind[m]	入力	ev に格納する各固有ベクトルが, 実固有ベクトルか複素固有ベクトルかの指示. ind[j-1] = 1 ならば, ev の j-1 行は実固有ベクトル ind[j-1] = -1 ならば, ev の j-1 行は複素固有ベクトルの実部 ind[j-1] = 0 ならば, ev の j-1 行は複素固有ベクトルの虚部
m	int	入力	配列 ind の大きさ ($m \leq m \leq n$).
p	double p[n][k]	入力	ハウスホルダー法による $\mathbf{A}$ から $\mathbf{H}$ への変換行列. (“使用上の注意”b)参照)
pv	double pv[n]	入力	ハウスホルダー法による $\mathbf{A}$ から $\mathbf{H}$ への変換行列. (“使用上の注意”b)参照)
dv	double dv[n]	入力	実行列 $\mathbf{A}$ の平衡化に使われたスケーリングファクタ. 平衡化が行われなかったときは, dv[0] = 0 と指示できる.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	$n = 1$	ev[0][0] = 1
30000	次のいずれかであった. $m < 1$ 又は $m > n$ $k < n$	処理を打ち切る.

3. 使用上の注意

a) 引数 **ev**, **ind** 及び **m**

固有ベクトルの **ev** への格納は、実固有ベクトルは 1 行毎に、複素固有ベクトルは実部と虚部を対にして 2 行毎に格納すること。

なお、関数 `c_dhvec` 呼び出し後の引数 **ev**, **ind**, **m** は、本関数でそのまま入力できるようになっている。

b) 引数 **p** 及び **pv**

関数 `c_dhes1` の引数 **a** と **pv** は、本関数の引数 **p** と **pv** に対応しているので、そのまま使用できる。

c) 引数 **dv**

平衡化によるスケーリングファクタ **dv** の内容については、関数 `c_dblnc` を参照のこと。

4. 使用例

n 次の実行列 **A** を平衡化した後、関数 `c_dhes1` により実ヘッセンベルグ行列に変換し、固有値を関数 `c_dhsqr` で求め、固有ベクトルを `c_dhvec` 及び `c_dhbk1` で求める。

```
#include <stdio.h>
#include <stdlib.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 5

MAIN__()
{
    int ierr, icon;
    int n, i, j, k, m, mk, ind[NMAX];
    double a[NMAX][NMAX], pv[NMAX], aw[NMAX+4][NMAX];
    double er[NMAX], ei[NMAX], ev[NMAX][NMAX];
    double dv[NMAX];

    /* initialize matrix */
    n = NMAX;
    k = NMAX;
    mk = NMAX;
    for (i=0; i<n; i++) {
        a[i][i] = n-i;
        for (j=0; j<i; j++) {
            a[i][j] = n-i;
            a[j][i] = n-i;
        }
    }
    /* balance matrix A */
    ierr = c_dblnc((double*)a, k, n, dv, &icon);
    if (icon > 10000 ) {
        printf("ERROR: c_dblnc failed with icon = %i\n", icon);
        exit (1);
    }
    /* reduce matrix to Hessenberg form */
    ierr = c_dhes1((double*)a, k, n, pv, &icon);
    if (icon != 0 ) {
        printf("ERROR: c_dhes1 failed with icon = %i\n", icon);
        exit (1);
    }
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            aw[i][j] = a[i][j];
    /* find eigenvalues */
    ierr = c_dhsqr((double*)aw, k, n, er, ei, &m, &icon);
    if (icon >= 20000 ) {
        printf("ERROR: c_dhsqr failed with icon = %i\n", icon);
        exit (1);
    }
    for (i=0; i<m; i++) ind[i] = 1;
    /* find eigenvectors for given eigenvalues */
    ierr = c_dhvec((double*)a, k, n, er, ei,
                  ind, m, (double*)ev, mk, (double*)aw, &icon);
}
```

```
if (icon >= 20000 ) {
    printf("ERROR: c_dhvec failed with icon = %i\n", icon);
    exit (1);
}
/* back transformation to find e-vectors of A */
ierr = c_dhbk1((double*)ev, k, n, ind, m, (double*)a, pv, dv, &icon);
if (icon > 10000 ) {
    printf("ERROR: c_dhbk1 failed with icon = %i\n", icon);
    exit (1);
}
printf("icon = %i\n", icon);
/* print eigenvalues and eigenvectors */
i = 0;
k = 0;
while (i<m) {
    if (ind[i] == 0) i++;
    else if (ei[i] == 0) {
        /* real eigenvector */
        printf("eigenvalue: %12.4f\n", er[i]);
        printf("eigenvector:");
        for (j=0;j<n;j++)
            printf("%7.4f ", ev[k][j]);
        printf("\n");
        i++;
        k++;
    }
    else {
        /* complex eigenvector pair */
        printf("eigenvalue: %7.4f+i*%7.4f\n", er[i], ei[i]);
        printf("eigenvector: ");
        for (j=0;j<n;j++)
            printf("%7.4f+i*%7.4f ", ev[k][j], ev[k+1][j]);
        printf("\n");
        printf("eigenvalue: %7.4f+i*%7.4f\n", er[i+1], ei[i+1]);
        printf("eigenvector: ");
        for (j=0;j<n;j++)
            printf("%7.4f+i*%7.4f ", ev[k][j], -ev[k+1][j]);
        printf("\n");
        i = i+2;
        k = k+2;
    }
}
return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の HES1, BLNC, NRML, HBK1 の項目及び[119]を参照のこと。

c_dheig2

エルミート行列の固有値及び固有ベクトル
(ハウスホルダー法, バイセクション法, 逆反復法)

```
ierr = c_dheig2(a, k, n, m, e, evr, evi, vw,
                &icon);
```

1. 機能

n 次のエルミート行列 A の固有値を, バイセクション法により, 大きい方から又は小さい方から m 個求め, 対応する固有ベクトルを逆反復法により求める. 固有ベクトルは, $\|x\|_2=1$ となるように正規化される. $1 \leq m \leq n$ なること.

2. 引数

呼出し形式:

```
ierr = c_dheig2((double *)a, k, n, m, e, (double *)evr, (double *)evi, vw,
                &icon);
```

引数の説明:

a	double a[n][k]	入力	エルミート行列 A . エルミート行列用圧縮格納法. (“1.3.3 エルミート行列の格納方法”参照)
		出力	演算後, 内容は保存されない.
k	int	入力	配列 a, evr, evi の整合寸法 ($k \geq n$).
n	int	入力	エルミート行列 A の次数 n .
m	int	入力	求める固有値の個数 m . $m = +m$ のときは大きい方から求める. $m = -m$ のときは小さい方から求める.
e	double e[m]	出力	エルミート行列 A の固有値.
evr	double evr[m][k]	出力	固有ベクトルの実数部.
evi	double evi[m][k]	出力	固有ベクトルの虚数部. 行方向に格納される. 第 j 番目の固有値 $e[j], j=0, \dots, m -1$ に対応する固有ベクトルの, 第 l 番目の要素は $evr[j][l] + i \cdot evi[j][l]$ と表わせる. ($l=0, \dots, n-1$)
vw	double vw[9n]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.

コード	意味	処理内容
10000	$n=1$ であった.	$e[0] = a[0][0]$ $evr[0][0] = 1$ $evi[0][0] = 0$ とする.
15000	固有ベクトルのすべてを求めることはできなかった.	求められなかった固有ベクトルを 0 ベクトルとする.
20000	固有ベクトルが一つも求められなかった.	すべての固有ベクトルを 0 ベクトルとする.
30000	次のいずれかであった. $n < m$ $k < n$ $m = 0$	処理を打ち切る.

3. 使用上の注意

a) 関数の使用条件

本関数はエルミート行列用であり、一般複素行列については使用できない。一般複素行列を求める場合には、c_dceig2 を用いること。

4. 使用例

5×5 のエルミート行列の固有値及び固有ベクトルを求める。

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 5

MAIN__()
{
    int ierr, icon;
    int n, m, i, j, k;
    double a[NMAX][NMAX], e[NMAX], evr[NMAX][NMAX], evi[NMAX][NMAX], vw[9*NMAX];

    /* initialize matrix */
    n = NMAX;
    k = NMAX;
    for (i=0; i<n; i++)
        for (j=0; j<=i; j++) {
            a[i][j] = n-i;
            a[j][i] = n-i;
        }
    m = n;
    /* find eigenvalues and eigenvectors */
    ierr = c_dheig2((double*)a, k, n, m, e,
                   (double*)evr, (double*)evi, vw, &icon);
    printf("icon = %i\n", icon);
    /* print eigenvalues and eigenvectors */
    for (i=0; i<m; i++) {
        printf("eigenvalue: %7.4f\n", e[i]);
        printf("eigenvector: ");
        for (j=0; j<n; j++)
            printf("{%7.4f, %7.4f} ", evr[i][j], evi[i][j]);
        printf("\n");
    }
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の HEIG2 の項目及び[74], [118], [119]を参照のこと。

c_dhes1

実行列の実ヘッセンベルグ行列への変換 (ハウスホルダー法)

ierr = c_dhes1(a, k, n, pv, &icon);

1. 機能

n 次の実行列 **A** を, ハウスホルダー法 (直交相似変換) により実ヘッセンベルグ行列 **H** へ変換する.

$$\mathbf{H} = \mathbf{P}^T \mathbf{A} \mathbf{P}$$

ただし **P** は変換行列である. $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dhes1((double *) a, k, n, pv, &icon);
```

引数の説明:

a	double	入力	実行列 A .
	a[n][k]	出力	実ヘッセンベルグ行列 H 及び変換行列 P . (“使用上の注意” a) 参照)
k	int	入力	配列 a の整合寸法 ($\geq n$).
n	int	入力	実行列 A の次数 n .
pv	double pv[n]	出力	変換行列 P . (“使用上の注意” a) 参照)
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
10000	$n = 1$ 又は $n = 2$	変換しない.
30000	次のいずれかであった. • $n < 1$ • $k < n$	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

本関数に続いて c_dhsqr 及び c_dhvec を呼び出すことにより実行列 **A** の固有値・固有ベクトルを求めることができる. 本関数の引数 a は c_dhsqr 及び c_dhvec の引数 a に対応している.

本関数に続いて c_dhbk1 を呼び出すことにより, 実ヘッセンベルグ行列 **H** の固有ベクトルを実行列 **A** の固有ベクトルへの逆変換を行うことができる. 本関数の引数 a, pv は c_dhbk1 の引数 p, pv に対応している.

b) 精度

固有値の精度は, 実ヘッセンベルグ行列化を行った時点である程度決定される. そのため, できるだけ精度よく実ヘッセンベルグ行列を求めることが必要であり, 本関数でも若干の考慮が払われている. しかし, 固有値に非常に大きいものと小さいものがあるとき, 小さい方の固有値は, 大きい固有値に比べて変換の影響を受けやすい. したがって, 小さい固有値を精度よく求めることが難しい場合がある.

4. 使用例

n 次の実行列 **A** を関数 c_dhes1 により実ヘッセンベルグ行列に変換し, 固有値を関数 c_dhsqr で求める.

```
#include <stdio.h>
#include <stdlib.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 5

MAIN__()
{
    int ierr, icon;
    int n, i, j, k, m;
    double a[NMAX][NMAX], er[NMAX], ei[NMAX], pv[NMAX];

    /* initialize matrix */
    n = NMAX;
    k = NMAX;
    for (i=0; i<n; i++)
        for (j=0; j<=i; j++) {
            a[i][j] = i-n;
            a[j][i] = n-i;
        }

    /* reduce matrix to Hessenberg form */
    ierr = c_dhes1((double*)a, k, n, pv, &icon);
    if (icon != 0 ) {
        printf("ERROR: c_dhes1 failed with icon = %i\n", icon);
        exit (1);
    }
    /* find eigenvalues of Hessenberg matrix */
    ierr = c_dhsqr((double*)a, k, n, er, ei, &m, &icon);
    if (icon >= 20000 ) {
        printf("ERROR: c_dhsqr failed with icon = %i\n", icon);
        exit (1);
    }
    printf("icon = %i\n", icon);
    /* print eigenvalues */
    for (i=0; i<m; i++) {
        printf("%7.4f+i*%7.4f \n", er[i], ei[i]);
    }
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の HES1 の項目及び[119]を参照のこと.

c_dhrwiz

Hurwitz 多項式の判定

<pre>ierr = c_dhrwiz(a, na, isw, &iflg, &sa, vw, &icon);</pre>

1. 機能

n 次の実係数多項式

$$P(s) = a_1 s^n + a_2 s^{n-1} + \dots + a_n s + a_{n+1}$$

が Hurwitz 多項式 (すべての零点が複素平面の左半面, すなわち, $\text{Re}(s) < 0$ の領域に存在する多項式) であるかどうかを判定する. $P(s)$ が Hurwitz 多項式でない場合は $\alpha > \alpha_0 (\geq 0)$ で $P(s + \alpha)$ が Hurwitz 多項式となる α_0 を探索する.

2. 引数

呼出し形式:

```
ierr = c_dhrwiz(a, na, isw, &iflg, &sa, vw, &icon);
```

引数の説明:

a	double a[n+1]	入力	$P(s)$ の係数. $a[i-1] = a_i (i=1, \dots, n+1)$ のように格納する.
na	int	入力	$P(s)$ の次数 n .
isw	int	入力	制御情報. isw=0: $P(s)$ が Hurwitz 多項式か否かの判定だけを行う. isw=1: $P(s)$ が Hurwitz 多項式か否かの判定を行い, かつ Hurwitz 多項式でない場合には α_0 を探索する. 上記以外の値を入力すると, isw は isw=1 とみなして処理する.
iflg	int	出力	多項式の判定結果. iflg=0: Hurwitz 多項式である. iflg=1: Hurwitz 多項式でない.
sa	double	出力	α_0 の値. $P(s)$ が Hurwitz 多項式の場合は sa=0.0 と出力する.
vw	double vw[n+1]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	α_0 の値が見つからなかった.	処理を打ち切る.
30000	次のいずれかであった. • na < 1 • a[0] = 0	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

本関数の機能は、有理関数 $F(s) = Q(s)/P(s)$ のラプラス逆変換 $f(t)$ を求めることに関連し、 $f(t)$ の大ざっぱな振舞を調べる目的で利用できる。すなわち、 $P(s)$ が Hurwitz 多項式でないときは、 $F(s)$ が $\text{Re}(s) \geq 0$ の領域に特異点を持つことになり、その結果、逆変換 $f(t)$ は $t \rightarrow \infty$ のとき指数関数的に増大する。

有理関数 $F(s)$ のラプラス逆変換 $f(t)$ を求めるには、 $\text{Re}(s) > 0$ における $F(s)$ の正則性が既知か否かに応じて関数 `c_dlaps1`, 又は `c_dlaps2` を使用すればよい。

4. 使用例

多項式 $P(s) = s^4 - 12s^3 + 54s^2 - 108s + 81$ が、Hurwitz 多項式であるか否かを判定する。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
    double sa, a[5], vw[5];
    int isw, na, iflg, neps[9];

    /* generate initial data */
    na = 4;
    a[0] = 1;
    a[1] = -12;
    a[2] = 54;
    a[3] = -108;
    a[4] = 81;
    isw = 1;
    /* is it a Hurwitz polynomial ? */
    ierr = c_dhrwiz(a, na, isw, &iflg, &sa, vw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dhrwiz failed with icon = %d\n", icon);
        exit(1);
    }
    printf("iflg = %i    sa = %12.5e\n", iflg, sa);
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の HRWIZ の項目を参照のこと。

c_dhsqr

実ヘッセンベルグ行列の固有値(2段 QR 法)

ierr = c_dhsqr(a, k, n, er, ei, &m, &icon);

1. 機能

n 次の実ヘッセンベルグ行列 **A** の固有値を, 2 段 QR 法により求める. ただし $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dhsqr((double *) a, k, n, er, ei, &m, &icon);
```

引数の説明:

a	double	入力	実ヘッセンベルグ行列 A .
	a[n][k]	出力	演算後, 内容は保存されない.
k	int	入力	配列 a の整合寸法 ($\geq n$).
n	int	入力	実ヘッセンベルグ行列 A の次数 n .
er	double er[n]	出力	行列 A の固有値の実部.
ei	double ei[n]	出力	行列 A の固有値の虚部.
m	int	出力	求まった固有値の個数.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
10000	$n = 1$	er[0] = a[0][0] ei[0] = 0
15000	固有値のすべてを求めつくすことはできなかった.	m に求まった固有値の個数をセットする. $1 \leq m < n$
20000	固有値が, 一つも求まらなかった.	m = 0
30000	次のいずれかであった. • $n < 1$ • $k < n$	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

通常, 関数 c_dhes1 を実行した後, 本関数によって固有値を求める.

固有ベクトルをも必要とする場合は, 本関数を呼び出す前に配列 a の内容を他の領域へ移しておくこと.

4. 使用例

n 次の実行列 **A** を関数 c_dhes1 により実ヘッセンベルグ行列に変換し, 固有値を関数 c_dhsqr で求める.

```
#include <stdio.h>
#include <stdlib.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 5
```

```
MAIN__()
{
    int ierr, icon;
    int n, i, j, k, m;
    double a[NMAX][NMAX], er[NMAX], ei[NMAX], pv[NMAX];

    /* initialize matrix */
    n = NMAX;
    k = NMAX;
    for (i=0;i<n;i++)
        for (j=0;j<=i;j++) {
            a[i][j] = i-n;
            a[j][i] = n-i;
        }

    /* reduce matrix to Hessenberg form */
    ierr = c_dhes1((double*)a, k, n, pv, &icon);
    if (icon != 0 ) {
        printf("ERROR: c_dhes1 failed with icon = %i\n", icon);
        exit (1);
    }
    /* find eigenvalues of Hessenberg matrix */
    ierr = c_dhsqr((double*)a, k, n, er, ei, &m, &icon);
    if (icon >= 20000 ) {
        printf("ERROR: c_dhsqr failed with icon = %i\n", icon);
        exit (1);
    }
    printf("icon = %i\n", icon);
    /* print eigenvalues */
    for (i=0;i<m;i++) {
        printf("%7.4f+i*%7.4f \n", er[i], ei[i]);
    }
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の HSQR の項目及び [118], [119]を参照のこと.

c_dhvec

実ヘッセンベルグ行列の固有ベクトル(逆反復法)

<pre>ierr = c_dhvec(a, k, n, er, ei, ind, m, ev, mk, aw, &icon);</pre>

1. 機能

n 次の実ヘッセンベルグ行列 A の指定された固有値 λ_j に対応する固有ベクトル $\mathbf{x}_j$ を、逆反復法により求める。固有ベクトルの正規化は行わない。 $n \geq 1$ であること。

2. 引数

呼出し形式:

```
ierr = c_dhvec((double *)a, k, n, er, ei, ind, m, (double *)ev, mk, (double
               *)aw, &icon);
```

引数の説明:

a	double a[n][k]	入力	実ヘッセンベルグ行列 A .
k	int	入力	配列 a, ev 及び aw の整合寸法 ($\geq n$).
n	int	入力	実ヘッセンベルグ行列 A の次数 n .
er	double er[m]	入力	行列 A の固有値の実部を格納する. er[j-1] = $\text{Re}(\lambda_j)$, $j = 1, \dots, m$ (“使用上の注意” a)参照)
ei	double ei[m]	入力	行列 A の固有値の虚部を格納する. ei[j-1] = $\text{Im}(\lambda_j)$, $j = 1, \dots, m$ (“使用上の注意” a)参照)
ind	int ind[m]	入力	固有ベクトルの要不要の指定. ind[j-1] = 0 のとき, λ_j に対応する固有ベクトルを求めない. ind[j-1] = 1 のとき, λ_j に対応する固有ベクトルを求める. $j = 1, \dots, m$. (“使用上の注意” b)参照)
		出力	ev に格納された各固有ベクトルが, 実固有ベクトルか複素固有ベクトルかを示す. ind[j-1] = 1 ならば, ev の $j-1$ 行は実固有ベクトル ind[j-1] = -1 ならば, ev の $j-1$ 行は複素固有ベクトルの実部 ind[j-1] = 0 ならば, ev の $j-1$ 行は複素固有ベクトルの虚部
m	int	入力	配列 er, ei に格納されている固有値の総数 ($\leq n$).
ev	double ev[mk][k]	出力	固有ベクトル $\mathbf{x}$. 固有ベクトルを行方向に格納する. 実固有値に対応する実固有ベクトルは, ev の 1 行に, 複素固有値に対応する複素固有ベクトルは, 実部と虚部とに分けて 2 行に格納する. (“使用上の注意” b)参照)
mk	int	入力	配列 ev の行数. (“使用上の注意” b)参照)
aw	double aw[n+4][k]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	$n = 1$	$ev[0][0] = 1$
15000	ある固有値に対応する固有ベクトルが求まらなかった.	求まらなかった固有ベクトルの ind の情報は消去される.
16000	配列 ev の行数が, 要求されたすべての固有ベクトルを格納するには小さすぎた.	配列 ev に格納できるだけ計算する. 計算されなかったものの ind の情報は消去される.
20000	すべての固有ベクトルが求まらなかった.	ind の情報はすべて消去される.
30000	次のいずれかであった. $m < 1$ 又は $m > n$ $k < n$	処理を打ち切る.

3. 使用上の注意

a) 引数 er 及び ei

j 番目の固有値 λ_j が複素数のとき, 引数 er, ei に格納する固有値 λ_j 及び λ_{j+1} は共役複素数の対になっていなければならない.

b) 固有ベクトルの格納法

固有ベクトルは要求されたものだけが ev の第 1 行から順に格納される. 例えば, 第 1 番目と 2 番目の固有値に対応する固有ベクトルは不要で, 第 3 番目の固有値に対応する固有ベクトルを要求したとき, 要求された固有ベクトルは $ev[0][i]$ に実部を, $ev[1][i]$, $i=0, \dots, n-1$ に虚部を格納する. このとき, 固有値の並びと対応した行には格納されないので注意すること.

引数 mk はこの点を考慮して固有ベクトルの格納のために必要な行数を指定すればよい. もし固有ベクトルの格納に必要な行数が, mk で指定した行数より大きいときは, mk で指定した行数までは計算して格納されるが, 越える部分の指定は無視される. この場合, icon は 16000 となる.

c) 本関数の使用方法

本関数は実ヘッセンベルグ行列の固有ベクトルを求めるためのものである.

実行列の固有ベクトルを部分的に求める場合は,

- 関数 c_dhes1 により実ヘッセンベルグ行列に変換する.
- 関数 c_dhsqr により固有値を求める.
- 本関数により固有ベクトルを求める.
- 最後に関数 c_dhbk1 により実行列の固有ベクトルへ逆変換する.

なお, 実行列の固有値固有ベクトルをすべて求めるときは, 関数 c_deig1 を用いた方がよい.

本関数は, 実行列の固有ベクトルを部分的に求める場合に用いることを前提としている. したがって, 固有ベクトルの正規化は行わない. 実ヘッセンベルグ行列の固有ベクトルを求めるときは, 必要に応じて関数 c_dnrml により正規化すること.

関数 c_dhbk1 や c_dnrml を使用するとき, 本関数の引数 ind, m, ev はそのまま c_dhbk1 や c_dnrml の入力引数として使用できる.

4. 使用例

n 次の実行列 A を平衡化した後、関数 `c_dhes1` により実ヘッセンベルグ行列に変換し、固有値を関数 `c_dhsqr` で求め、固有ベクトルを `c_dhvec` 及び `c_dhbk1` で求める。

```
#include <stdio.h>
#include <stdlib.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 5

MAIN__()
{
    int ierr, icon;
    int n, i, j, k, m, mk, ind[NMAX];
    double a[NMAX][NMAX], pv[NMAX], aw[NMAX+4][NMAX];
    double er[NMAX], ei[NMAX], ev[NMAX][NMAX];
    double dv[NMAX];

    /* initialize matrix */
    n = NMAX;
    k = NMAX;
    mk = NMAX;
    for (i=0;i<n;i++) {
        a[i][i] = n-i;
        for (j=0;j<i;j++) {
            a[i][j] = n-i;
            a[j][i] = n-i;
        }
    }
    /* balance matrix A */
    ierr = c_dblnc((double*)a, k, n, dv, &icon);
    if (icon > 10000 ) {
        printf("ERROR: c_dblnc failed with icon = %i\n", icon);
        exit (1);
    }
    /* reduce matrix to Hessenberg form */
    ierr = c_dhes1((double*)a, k, n, pv, &icon);
    if (icon != 0 ) {
        printf("ERROR: c_dhes1 failed with icon = %i\n", icon);
        exit (1);
    }
    for (i=0;i<n;i++)
        for (j=0;j<n;j++)
            aw[i][j] = a[i][j];
    /* find eigenvalues */
    ierr = c_dhsqr((double*)aw, k, n, er, ei, &m, &icon);
    if (icon >= 20000 ) {
        printf("ERROR: c_dhsqr failed with icon = %i\n", icon);
        exit (1);
    }
    for (i=0;i<m;i++) ind[i] = 1;
    /* find eigenvectors for given eigenvalues */
    ierr = c_dhvec((double*)a, k, n, er, ei,
                  ind, m, (double*)ev, mk, (double*)aw, &icon);
    if (icon >= 20000 ) {
        printf("ERROR: c_dhvec failed with icon = %i\n", icon);
        exit (1);
    }
    /* back transformation to find e-vectors of A */
    ierr = c_dhbk1((double*)ev, k, n, ind, m, (double*)a, pv, dv, &icon);
    if (icon > 10000 ) {
        printf("ERROR: c_dhbk1 failed with icon = %i\n", icon);
        exit (1);
    }
    printf("icon = %i\n", icon);
    /* print eigenvalues and eigenvectors */
    i = 0;
    k = 0;
    while (i<m) {
        if (ind[i] == 0) i++;
        else if (ei[i] == 0) {
            /* real eigenvector */
            printf("eigenvalue: %12.4f\n", er[i]);
            printf("eigenvector:");

```

```
        for (j=0;j<n;j++)
            printf("%7.4f ", ev[k][j]);
        printf("\n");
        i++;
        k++;
    }
    else {
        /* complex eigenvector pair */
        printf("eigenvalue: %7.4f+i*%7.4f\n", er[i], ei[i]);
        printf("eigenvector: ");
        for (j=0;j<n;j++)
            printf("%7.4f+i*%7.4f ", ev[k][j], ev[k+1][j]);
        printf("\n");
        printf("eigenvalue: %7.4f+i*%7.4f\n", er[i+1], ei[i+1]);
        printf("eigenvector: ");
        for (j=0;j<n;j++)
            printf("%7.4f+i*%7.4f ", ev[k][j], -ev[k+1][j]);
        printf("\n");
        i = i+2;
        k = k+2;
    }
}
return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”のHVECの項目及び[118],[119]を参照のこと.

c_dicheb

チェビシェフ級数の不定積分

ierr = c_dicheb(a, b, c, &n, &icon);

1. 機能

区間 $[a, b]$ で定義された n 項のチェビシェフ級数,

$$f(x) = \sum_{k=0}^{n-1} c_k T_k \left(\frac{2x - (b+a)}{b-a} \right)$$

が与えられたとき, その不定積分をチェビシェフ級数,

$$\int f(x) dx = \sum_{k=0}^n \bar{c}_k T_k \left(\frac{2x - (b+a)}{b-a} \right)$$

で表現し, その係数 $\bar{c}_k$ を求める. ここで, 任意定数 $\bar{c}_0$ は便宜上 0 とする. また, $\sum$ は初項を $1/2$ 倍して和をとることを意味する. $a \neq b$, $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dicheb(a, b, c, &n, &icon);
```

引数の説明:

a	double	入力	チェビシェフ級数の定義域の下限 a .
b	double	入力	チェビシェフ級数の定義域の上限 b .
c	double c[n+1]	入力	係数 c_k . $c[k] = c_k$, $k = 0, 1, \dots, n-1$ のように格納する.
		出力	不定積分の係数 $\bar{c}_k$. $c[0] = 0$, $c[k] = \bar{c}_k$, $k = 1, 2, \dots, n$ のように格納される.
n	int	入力	項数 n .
		出力	不定積分の項数 $n+1$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	次のいずれかであった. • $n < 1$ • $a = b$	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

任意の関数の不定積分を求める場合, チェビシェフ級数展開する関数 c_dfcheb と, 本関数を順次呼び出せばよい.

更に, 区間内の任意の点 $v \in [a, b]$ に対する積分値を求める場合は, チェビシェフ級数の求和の関数 c_decheb を続けて呼び出せばよい. (“使用例” 参照)

b) 任意定数の決定法

本関数では、任意定数 $\bar{c}_0$ を便宜上 0 として出力する。そこで、区間内の任意の点 $v \in [a, b]$ における不定積分が値 y_v をとるように定数を定める場合、チェビシェフ級数の求和の関数 `c_decheb` を併用して次のような計算をする。

$$c[0] = 2(y_v - y)$$

ここで y は、`c_decheb` を使って求めた、点 v における不定積分のチェビシェフ級数の値。

チェビシェフ級数に展開された関数 $f(x)$ について、積分区間の上限を変えながら定積分、

$$\int_a^{x_i} f(t) dt, \quad x_i \in [a, b], \quad i = 1, 2, \dots, m$$

を求める場合、端点 a における不定積分の値が、0 となるように任意定数 $\bar{c}_0$ の値を決定し、以後 m 回チェビシェフ級数の求和を行えばよい。（“使用例”参照）

c) 誤差

不定積分の誤差は、末尾 2 項の係数の絶対値和で推定できる。

4. 使用例

関数(1)のチェビシェフ級数を求め、値を計算する。

$$f(x) = \frac{1}{4} + \int_0^x \frac{dt}{1+100t^2}, \quad x \in [0, 1] \quad (1)$$

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 257 /* default value */

double fun(double x); /* function prototype */
double truefun(double x);

MAIN__()
{
    int ierr, icon;
    int i, n, nmin, nmax;
    double epsa, epsr, err, a, b, v, f, h;
    double c[NMAX], tab[NMAX-2];

    /* initialize data */
    epsr = 5e-5;
    epsa = epsr;
    nmin = 9; /* default value */
    nmax = NMAX;
    a = 0;
    b = 1;
    /* expand function as Chebyshev series */
    ierr = c_dfcheb(a, b, fun, epsa, epsr, nmin, nmax, c, &n, &err, tab, &icon);
    if (icon >= 20000) {
        printf("ERROR: icon = %4i\n", icon);
        exit(1);
    }
    /* now calculate integral */
    ierr = c_dicheb(a, b, c, &n, &icon);
    if (icon != 0) {
        printf("ERROR: icon = %4i\n", icon);
        exit(1);
    }
    /* set constant term c0 */
    ierr = c_decheb(a, b, c, n, a, &f, &icon);
    if (icon != 0) {
        printf("ERROR: icon = %4i\n", icon);
        exit(1);
    }
    c[0] = (0.25-f)*2;
```

```

/* now evaluate Chebyshev series at points */
h = 0.05;
printf(" v      integral      error  \n");
for (i=0;i<=20;i++) {
    v = a+i*h;
    ierr = c_decheb(a, b, c, n, v, &f, &icon);
    if (icon != 0) {
        printf("ERROR:  icon = %4i\n", icon);
        exit(1);
    }
    err = truefun(v) - f;
    printf("%4.2f  %12.5e  %12.5e\n", v, f, err);
}
return(0);
}

/* function to expand */
double fun(double x)
{
    return 1/(1+100*x*x);
}

/* true integral function */
double truefun(double x)
{
    return 0.25+atan(10*x)/10;
}

```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”のICHEBの項目を参照のこと.

c_dierf

逆誤差関数 $\text{erf}^{-1}(x)$

<code>ierr = c_dierf(x, &f, &icon);</code>

1. 機能

誤差関数 $\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$ の逆関数 $\text{erf}^{-1}(x)$ の値を、多項式及び有理関数の最良近似式を用いて計算する。ただし $|x| < 1$ であること。

2. 引数

呼出し形式:

```
ierr = c_dierf(x, &f, &icon);
```

引数の説明:

x	double	入力	独立変数 x . (“使用上の注意” a)参照)
f	double	出力	関数値 $\text{erf}^{-1}(x)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
30000	$ x \geq 1$	$f = 0.0$ とする.

3. 使用上の注意

a) 引数 x

引数 x の範囲は $|x| < 1$ であること. この範囲がこの関数の定義領域である.

b) 関数 c_dierf 及び c_dierfc

逆誤差関数 $\text{erfc}^{-1}(x)$ との間の関係,

$$\text{erf}^{-1}(x) = \text{erfc}^{-1}(1 - x)$$

を利用すれば、逆誤差関数の値を関数 `c_dierfc` を用いて計算することもできるが、 $|x| \leq 0.8$ の範囲の x については本関数 `c_dierf` を用いる方が、値が正確で計算時間も短い。

4. 使用例

$\text{erf}^{-1}(x)$ の値を、0 から 0.99 までの 0.01 刻みの x の値に対して計算し数表を作る。

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
    double x, f;
    int i;

    for (i=0; i<100; i++) {
        x = (double)i/100;
        /* calculate inverse error function */
        ierr = c_dierf(x, &f, &icon);
```

```
    if (icon == 0)
        printf("x = %5.2f    f = %f\n", x, f);
    else
        printf("ERROR: x = %5.2f    f = %f    icon = %i\n", x, f, icon);
    }
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の IERF の項目を参照のこと。

c_dierfc

逆余誤差関数 $\operatorname{erfc}^{-1}(x)$

<code>ierr = c_dierfc(x, &f, &icon);</code>

1. 機能

余誤差関数 $\operatorname{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^{\infty} e^{-t^2} dt$ の逆関数 $\operatorname{erfc}^{-1}(x)$ の値を、多項式及び有理関数の最良近似式を用いて計算する。ただし、 $0 < x < 2$ であること。

2. 引数

呼出し形式:

```
ierr = c_dierfc(x, &f, &icon);
```

引数の説明:

x	double	入力	独立変数 x . (“使用上の注意” a)参照)
f	double	出力	関数値 $\operatorname{erfc}^{-1}(x)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
30000	$x \leq 0$ 又は $x \geq 2$	$f = 0.0$ とする.

3. 使用上の注意

a) 引数 x

引数 x の範囲は $0 < x < 2$ であること. この範囲がこの関数の定義領域である.

b) 関数 c_dierfc 及び c_dierf

逆余誤差関数 $\operatorname{erf}^{-1}(x)$ との関係,

$$\operatorname{erfc}^{-1}(x) = \operatorname{erf}^{-1}(1-x)$$

を利用すれば、逆余誤差関数の値を関数 `c_dierf` を用いて計算することもできるが、 $0 < x < 0.2$ の範囲の x については本関数 `c_dierfc` を用いる方が、値が正確で計算時間も短い。

4. 使用例

$\operatorname{erfc}^{-1}(x)$ の値を、0.01 から 1.0 までの 0.01 刻みの x の値に対して計算し数表を作る。

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
    double x, f;
    int i;

    for (i=1; i<=100; i++) {
        x = (double)i/100;
        /* calculate inverse complementary error function */
        ierr = c_dierfc(x, &f, &icon);
        if (icon == 0)
```



```
        printf("x = %5.2f    f = %f\n", x, f);  
    else  
        printf("ERROR: x = %5.2f    f = %f    icon = %i\n", x, f, icon);  
    }  
    return(0);  
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の IERFC の項目を参照のこと.

c_digam1

第1種不完全ガンマ関数 $\gamma(\nu, x)$
<code>ierr = c_digam1(v, x, &f, &icon);</code>

1. 機能

第1種不完全ガンマ関数 $\gamma(\nu, x)$ の値

$$\gamma(\nu, x) = \int_0^x e^{-t} t^{\nu-1} dt$$

を級数展開, 漸近展開及び数値積分により求める。ただし, $\nu > 0$, $x \geq 0$ であること。

2. 引数

呼出し形式:

```
ierr = c_digam1(v, x, &f, &icon);
```

引数の説明:

v	double	入力	独立変数 ν .
x	double	入力	独立変数 x .
f	double	出力	関数値 $\gamma(\nu, x)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	次のいずれかであった. • $\nu \leq 0$ • $x < 0$	$f = 0$ とする.

3. 使用上の注意

a) 引数 **x**

引数 x の範囲が, $x \geq 46.0$ であれば, $\gamma(\nu, x) \approx \Gamma(\nu)$ となるので, Fortran 基本関数の完全ガンマ関数 GAMMA(ν)を利用した方がよい。

4. 使用例

$\gamma(\nu, x)$ の値を ν 及び x を変化させて求め, 関数表を作成する。

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
    double v, x, f;
    int iv, ix;

    for (iv=1; iv<10; iv++) {
```

```

v = (iv+7*(iv-1.0)/3)/10;
for (ix=1;ix<10;ix++) {
  x = (ix+7*(ix-1.0)/3)/10;
  /* calculate incomplete gamma function */
  ierr = c_digam1(v, x, &f, &icon);
  if (icon == 0)
    printf("v = %5.2f    x = %5.2f    f = %f\n", v, x, f);
  else
    printf("ERROR: v = %5.2f    x = %5.2f    f = %f    icon = %i\n",
          v, x, f, icon);
}
}
return(0);
}

```

5. 手法概要

第 1 種不完全ガンマ関数 $\gamma(\nu, x)$ は, $x_1 = 5.5$ を境にして計算方法が異なる.

- $x \leq 2(\nu+1)$ 又は $x < x_1$ のとき, 級数展開を用いて計算する.
- $x > 2(\nu+1)$ かつ $x \geq x_1$ のとき, Fortran 関数の GAMMA 及び Fortran SSL II の DIGAM2 を用いて計算する.

計算方法の詳細については“富士通 SSL II 使用手引書”の IGAM1 の項目及び [140]を参照のこと.

c_digam2

第2種不完全ガンマ関数 $\Gamma(\nu, x)$
<code>ierr = c_digam2(ν, x, &f, &icon);</code>

1. 機能

第2種不完全ガンマ関数 $\Gamma(\nu, x)$ の値,

$$\Gamma(\nu, x) = \int_0^\infty e^{-t} t^{\nu-1} dt = e^{-x} \int_0^\infty e^{-t} (x+t)^{\nu-1} dt$$

を級数展開, 漸近展開及び数値積分により求める. ただし, $\nu \geq 0$, $x \geq 0$ ($\nu = 0$ のとき, $x \neq 0$) であること.

2. 引数

呼出し形式:

```
ierr = c_digam2( $\nu$ ,  $x$ , & $f$ , &icon);
```

引数の説明:

ν	double	入力	独立変数 ν .
x	double	入力	独立変数 x .
f	double	出力	関数値 $\Gamma(\nu, x)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	$x^{\nu-1}e^{-x} > fl_{\max}$ であった.	$f = fl_{\max}$ とする.
30000	次のいずれかであった. $\nu < 0$ $x < 0$ $\nu = 0$ かつ $x = 0$	$f = 0$ とする.

3. 使用上の注意

a) 引数 x

引数 x の範囲が $x \geq \log(fl_{\max})$ となると $\Gamma(\nu, x)$ はアンダフローするほど小さくなる.

$x^{\nu-1}e^{-x} > fl_{\max}$ は $x > 1$ で, ν が非常に大きいときで, そのとき $\Gamma(\nu, x)$ はオーバーフローする.

4. 使用例

$\Gamma(\nu, x)$ の値を ν 及び x を変化させて求め, 関数表を作成する.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
```

```

double v, x, f;
int iv, ix;

for (iv=1;iv<10;iv++) {
  v = (iv+7*(iv-1.0)/3)/10;
  for (ix=1;ix<10;ix++) {
    x = (ix+7*(ix-1.0)/3)/10;
    /* calculate incomplete gamma function */
    ierr = c_digam2(v, x, &f, &icon);
    if (icon == 0)
      printf("v = %5.2f    x = %5.2f    f = %f\n", v, x, f);
    else
      printf("ERROR: v = %5.2f    x = %5.2f    f = %f    icon = %i\n",
             v, x, f, icon);
  }
}
return(0);
}

```

5. 手法概要

第2種不完全ガンマ関数 $\Gamma(\nu, x)$ は, x 又は ν の値によって計算方法が異なる.

- $\nu=0$ かつ $x>0$ のとき, Fortran ルーチン DEXPI を用いて計算する.
- $\nu>0$ かつ $x=0$ のとき, Fortran 基本関数 GAMMA を用いて計算する.
- $\nu>0$ かつ $x>0$ のとき, $x_1 = 40.0$ を境にして以下の計算を行なう.
 - ν =整数 又は $x>x_1$ のとき, 漸近展開式を用いて計算する.
 - $\nu \neq$ 整数 かつ $x \leq x_1$ のとき, 数値積分法を用いて計算する.

計算方法の詳細については“富士通 SSL II 使用手引書”の IGAM2 の項目及び[139], [140]を参照のこと.

c_dindf

逆正規分布関数 $\phi^{-1}(x)$

<code>ierr = c_dindf(x, &f, &icon);</code>

1. 機能

正規分布関数 $\phi(x) = \frac{1}{\sqrt{2\pi}} \int_0^x e^{-\frac{t^2}{2}} dt$ の逆関数 $\phi^{-1}(x)$ の値を, 関数式,

$$\phi^{-1}(x) = \sqrt{2} \operatorname{erf}^{-1}(2x)$$

により, 計算する. ただし $|x| < 1/2$ であること.

2. 引数

呼出し形式:

```
ierr = c_dindf(x, &f, &icon);
```

引数の説明:

x	double	入力	独立変数 x . (“使用上の注意” a)参照)
f	double	出力	関数値 $\phi^{-1}(x)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	$ x \geq 1/2$	$f = 0.0$ とする.

3. 使用上の注意

a) 引数 x

引数 x の範囲は $|x| < 1/2$ であること. この範囲がこの関数の定義領域である.

b) 関数 c_dindf 及び c_dindfc

逆余正規分布関数 $\psi^{-1}(x)$ との関係

$$\phi^{-1}(x) = \psi^{-1}(1/2 - x)$$

を利用すれば, $\phi^{-1}(x)$ の値を関数 c_dindfc を用いて計算することもできるが, $|x| \leq 0.4$ の範囲の x については本関数 c_dindf を用いる方が, 値が正確で計算時間も短い.

4. 使用例

$\phi^{-1}(x)$ の値を, 0 から 0.49 までの 0.01 刻みの x の値に対して計算し数表を作る.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
    double x, f;
    int i;

    for (i=0; i<50; i++) {
        x = (double)i/100;
```

```
/* calculate inverse normal distribution function */
ierr = c_dindf(x, &f, &icon);
if (icon == 0)
    printf("x = %5.2f    f = %f\n", x, f);
else
    printf("ERROR: x = %5.2f    f = %f    icon = %i\n", x, f, icon);
}
return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の INDF の項目を参照のこと.

c_dindfc

逆余正規分布関数 $\psi^{-1}(x)$
<code>ierr = c_dindfc(x, &f, &icon);</code>

1. 機能

余正規分布関数 $\psi(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-\frac{t^2}{2}} dt$ の逆関数 $\psi^{-1}(x)$ の値を, 関係式

$$\psi^{-1}(x) = \sqrt{2} \operatorname{erfc}^{-1}(2x)$$

により計算する. ただし, $0 < x < 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dindfc(x, &f, &icon);
```

引数の説明:

x	double	入力	独立変数 x . (“使用上の注意” a)参照)
f	double	出力	関数値 $\psi^{-1}(x)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
30000	$x \leq 0$ 又は $x \geq 1$	f = 0.0 とする.

3. 使用上の注意

a) 引数 x

引数 x の範囲は $0 < x < 1$ であること. この範囲がこの関数の定義領域である.

b) c_dindfc 及び c_dindf

逆正規分布関数 $\phi^{-1}(x)$ との関係

$$\psi^{-1}(x) = \phi^{-1}(1/2 - x)$$

を利用すれば, $\psi^{-1}(x)$ の値を関数 c_dindf を用いて計算することもできるが, $0 < x < 0.1$ の範囲の x については本関数 c_dindfc を用いる方が, 値が正確で計算時間も短い.

4. 使用例

$\psi^{-1}(x)$ の値を, 0.01 から 0.5 までの 0.01 刻みの x の値に対して計算し数表を作る.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
    double x, f;
    int i;

    for (i=1; i<=50; i++) {
        x = (double)i/100;
```



```
/* calculate inverse complementary normal distribution function */
ierr = c_dindfc(x, &f, &icon);
if (icon == 0)
    printf("x = %5.2f    f = %f\n", x, f);
else
    printf("ERROR: x = %5.2f    f = %f    icon = %i\n", x, f, icon);
}
return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の INDFC の項目を参照のこと.

c_dlaps1

ラプラス変換 (複素右半平面で正則な有理関数)

ierr = c_dlaps1(a, na, b, nb, t, delt, np, epsr, ft, t1, neps, errv, &icon);

1. 機能

(1)で表される有理関数 $F(s)$ が与えられたとき, そのラプラス逆変換 $f(t)$ の値, $f(t_0)$, $f(t_0 + \Delta t)$, ..., $f(t_0 + (\ell - 1)\Delta t)$ を求める.

$$F(s) = \frac{Q(s)}{P(s)} \quad (1)$$

$$Q(s) = b_1 s^m + b_2 s^{m-1} + \dots + b_m s + b_{m+1}$$

$$P(s) = a_1 s^n + a_2 s^{n-1} + \dots + a_n s + a_{n+1}$$

$$n \geq m, a_i, i=1, \dots, n+1, b_j, j=1, \dots, m+1 (a_i, b_j: \text{実数})$$

ただし, $F(s)$ は $\text{Re}(s) > 0$ で正則であるとする.

2. 引数

呼出し形式:

```
ierr = c_dlaps1(a, na, b, nb, t, delt, np, epsr, ft, t1, neps, errv, &icon);
```

引数の説明:

a	double a[n+1]	入力	$P(s)$ の係数. $a[i-1] = a_i, i=1, \dots, n+1$ のように格納する.
na	int	入力	$P(s)$ の次数 n .
b	double b[m+1]	入力	$Q(s)$ の係数. $b[j-1] = b_j, j=1, \dots, m+1$ のように格納する.
nb	int	入力	$Q(s)$ の次数 m .
t	double	入力	$f(t)$ の計算開始値 $t_0 (\geq 0)$.
delt	double	入力	変数 t の増分 $\Delta t (\geq 0)$. $\text{delt} = 0$ の場合は, $f(t_0)$ だけが計算される.
np	int	入力	$f(t)$ の値を求めようとする点の個数 ($\ell \geq 1$).
epsr	double	入力	$f(t)$ の値に対する相対誤差の上限 (≥ 0). $10^{-2} \sim 10^{-7}$ 程度がよい. $\text{epsr} = 0.0$ と入力したときは標準値 10^{-4} が採用される.
ft	double ft[np]	出力	$f(t_0 + i\Delta t)$ の値. $\text{ft}[i] = f(t_0 + i\Delta t)$
t1	double t1[np]	出力	$t_0 + i\Delta t$ の値. $\text{t1}[i] = t_0 + i\Delta t$
neps	int neps[np]	出力	打ち切り項数 N_i . $\text{ft}[i]$ を計算するために用いた項数 N_i が $\text{neps}[i]$ に格納される.
errv	double errv[np]	出力	各結果 $\text{ft}[i]$ の相対誤差. $\text{ft}[i]$ における相対誤差が $\text{errv}[i]$ に格納される.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	結果 $f(t)$ の中に要求精度を満足しないものがある.	処理を続ける. $f(t_0 + i\Delta t)$, $i = 0, 1, \dots, \ell - 1$ の精度は配列 errv に出力される.
30000	次のいずれかであった. • $nb < 0$ 又は $nb > na$ • $t < 0$ 又は $delt < 0$ • $np < 1$ • $epsr < 0$ • $a[0] = 0$	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

有理関数 $F(s)$ は $\text{Re}(s) > 0$ で正則であること. $\text{Re}(s) > 0$ で非正則であったり, あるいは正則性について不明である場合は, 本関数の代りに c_dlaps2 を使用すること.

b) 初期値

$t_0 = 0$ と与えたとき $f(0)$ の値は, 次の初期値定理より計算する.

$$f(0) = \begin{cases} b_1 / a_1 & (n = m + 1) \\ 0 & (n > m + 1) \end{cases}$$

c) 引数 **na** 及び **nb**

$n = m$ の場合, (1)は,

$$F(s) = \frac{Q(s)}{P(s)} = F_1(s) + F_2(s)$$

ただし,

$$F_1(s) \equiv b_1 / a_1$$

$$F_2(s) \equiv \frac{c_2 s^{n-1} + c_3 s^{n-2} + \dots + c_{n+1}}{a_1 s^n + a_2 s^{n-1} + \dots + a_{n+1}}$$

と分解できる. $F_1(s)$ の逆変換 $f_1(t)$ は,

$$f_1(t) = \frac{b_1}{a_1} \delta(t) \quad \delta(t): \text{デルタ関数}$$

となる. $n = m$ のとき, 本関数を用いて逆変換を計算すると $t > 0$ で $F_2(s)$ の逆変換の値が出力される. $t = 0$ では浮動小数点の最大値を出力する.

4. 使用例

$\text{Re}(s) > 0$ で正則な有理関数,

$$F(s) = \frac{s^2 + 4}{s^4 + 12s^3 + 54s^2 + 108s + 81}$$

のラプラス逆変換 $f(t)$ を $t_i = 0.2 + 0.2(i-1)$, $i=1, 2, \dots, 9$ の各点で求める.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
```

```
int ierr, icon;
double t, delt, epsr;
double a[5], b[3], t1[9], ft[9], errv[9];
int i, l, na, nb, neps[9];

/* generate initial data */
nb = 2;
b[0] = 1;
b[1] = 0;
b[2] = 4;
na = 4;
a[0] = 1;
a[1] = 12;
a[2] = 54;
a[3] = 108;
a[4] = 81;
t = 0.2;
delt = 0.2;
l = 9;
epsr = 1e-4;
/* calculate inverse Laplace transform */
ierr = c_dlaps1(a, na, b, nb, t, delt, l, epsr, ft, t1, neps, errv, &icon);
if (icon > 10000) {
    printf("ERROR: c_dlaps1 failed with icon = %d\n", icon);
    exit(1);
}
printf("icon = %i\n", icon);
printf(" t1      ft      errv      neps  \n");
for (i=0;i<l;i++) {
    printf("%4.2f %12.5e %12.5e %4i\n",
           t1[i], ft[i], errv[i], neps[i]);
}
return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の LAPS1 の項目を参照のこと。

c_dlaps2

ラプラス変換 (一般の有理関数)

```
ierr = c_dlaps2(a, na, b, nb, t, delt, np,
               epsr, ft, t1, neps, errv, &iflg,
               vw, &icon);
```

1. 機能

(1)で表される有理関数 $F(s)$ が与えられたとき, そのラプラス逆変換 $f(t)$ の値, $f(t_0)$, $f(t_0 + \Delta t)$, ..., $f(t_0 + (\ell - 1)\Delta t)$ を求める.

$$F(s) = \frac{Q(s)}{P(s)} \quad (1)$$

$$Q(s) = b_1 s^m + b_2 s^{m-1} + \dots + b_m s + b_{m+1}$$

$$P(s) = a_1 s^n + a_2 s^{n-1} + \dots + a_n s + a_{n+1}$$

$$n \geq m, \quad a_i, i = 1, \dots, n+1, \quad b_j, j = 1, \dots, m+1 \quad (a_i, b_i: \text{実数})$$

ここで $F(s)$ は $\text{Re}(s) > 0$ で正則であってもなくてもよい.

2. 引数

呼出し形式:

```
ierr = c_dlaps2(a, na, b, nb, t, delt, np, epsr, ft, t1, neps, errv, &iflg,
               vw, &icon);
```

引数の説明:

a	double a[n+1]	入力	$P(s)$ の係数. $a[i-1] = a_i, i = 1, \dots, n+1$ のように格納する.
na	int	入力	$P(s)$ の次数 n .
b	double b[m+1]	入力	$Q(s)$ の係数. $b[j-1] = b_j, j = 1, \dots, m+1$ のように格納する.
nb	int	入力	$Q(s)$ の次数 m .
t	double	入力	$f(t)$ の計算開始値 $t_0 (\geq 0)$.
delt	double	入力	変数 t の増分 $\Delta t (\geq 0)$. delt = 0 の場合は, $f(t_0)$ だけが計算される.
np	int	入力	$f(t)$ の値を求めようとする点の個数 ($\ell \geq 1$).
epsr	double	入力	$f(t)$ の値に対する相対誤差の上限 (≥ 0). $10^{-2} \sim 10^{-7}$ 程度がよい. epsr = 0.0 と入力したときは標準値 10^{-4} が採用される.
ft	double ft[np]	出力	$f(t_0 + i\Delta t)$ の値. $ft[i] = f(t_0 + i\Delta t)$
t1	double t1[np]	出力	$t_0 + i\Delta t$ の値. $t1[i] = t_0 + i\Delta t$
neps	int neps[np]	出力	打ち切り項数 N_i . $ft[i]$ を計算するために用いた項数 N_i が $neps[i]$ に格納される.
errv	double errv[np]	出力	各結果 $ft[i]$ の相対誤差. $ft[i]$ における相対誤差が $errv[i]$ に格納される.
iflg	int	出力	正則性の判定結果. iflg = 0: $F(s)$ が $\text{Re}(s) > 0$ で正則である. iflg = 1: 上記以外. (“使用上の注意” c)参照)
vw	double vw[na+nb+2]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
10000	結果 $f(t)$ の中に要求精度を満足しないものがある.	処理を続ける. $f(t_0 + i\Delta t)$, $i = 0, 1, \dots, \ell - 1$ の精度は配列 errv に出力される.
20000	$F(s)$ が $\text{Re}(s) > \gamma_0$ で正則となる非負な実数 γ_0 が見つからなかった.	処理を打ち切る.
30000	次のいずれかであった. • $\text{nb} < 0$ 又は $\text{nb} > \text{na}$ • $t < 0$ 又は $\text{delt} < 0$ • $\text{np} < 1$ • $\text{epsr} < 0$ • $\text{a}[0] = 0$	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

有理関数 $F(s)$ は, $\text{Re}(s) > 0$ で正則であってもなくてもよい. しかし, $\text{Re}(s) > 0$ で正則であることが分っているならば, 本関数よりも **c_dlaps1** を用いた方が処理が速い.

b) 初期値

$t_0 = 0$ と与えたとき $f(0)$ の値は, 次の初期値定理より計算する.

$$f(0) = \begin{cases} b_1 / a_1 & (n = m + 1) \\ 0 & (n > m + 1) \end{cases}$$

c) 引数 **iflg**

iflg = 1 と出力されたときは, $F(s)$ が $\text{Re}(s) > 0$ で正則でないことを表す. このことは, $f(t)$ が $t \rightarrow \infty$ のとき指数関数的に増大することをも意味する.

d) 引数 **na** 及び **nb**

$n = m$ の場合, (1)は

$$F(s) = \frac{Q(s)}{P(s)} = F_1(s) + F_2(s)$$

ただし,

$$F_1(s) \equiv b_1 / a_1$$

$$F_2(s) \equiv \frac{c_2 s^{n-1} + c_3 s^{n-2} + \dots + c_{n+1}}{a_1 s^n + a_2 s^{n-1} + \dots + a_{n+1}}$$

と分解できる. $F_1(s)$ の逆変換 $f_1(t)$ は

$$f_1(t) = \frac{b_1}{a_1} \delta(t) \quad \delta(t): \text{デルタ関数}$$

となる. $n = m$ のとき, 本関数を用いて逆変換を計算すると $t > 0$ で $F_2(s)$ の逆変換の値が出力される. $t = 0$ では浮動小数点の最大値を出力する.

4. 使用例

有理関数

$$F(s) = \frac{s^2 + 4}{s^4 - 12s^3 + 54s^2 - 108s + 81}$$

のラプラス逆変換 $f(t)$ を求める. $\text{epsr}=10^{-4}$ とする.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
    double t, delt, epsr;
    double a[5], b[3], t1[9], ft[9], errv[9], vw[8];
    int i, l, na, nb, iflg, neps[9];

    /* generate initial data */
    nb = 2;
    b[0] = 1;
    b[1] = 0;
    b[2] = 4;
    na = 4;
    a[0] = 1;
    a[1] = -12;
    a[2] = 54;
    a[3] = -108;
    a[4] = 81;
    t = 0.2;
    delt = 0.2;
    l = 9;
    epsr = 1e-4;
    /* calculate inverse Laplace transform */
    ierr = c_dlaps2(a, na, b, nb, t, delt, l, epsr, ft, t1,
                   neps, errv, &iflg, vw, &icon);
    if (icon >= 20000) {
        printf("ERROR: c_dlaps2 failed with icon = %d\n", icon);
        exit(1);
    }
    printf("icon = %i    iflg = %i\n", icon, iflg);
    printf(" t1          ft          errv          neps    \n");
    for (i=0; i<l; i++) {
        printf("%4.2f %12.5e %12.5e %4i\n",
              t1[i], ft[i], errv[i], neps[i]);
    }
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の LAPS2 の項目を参照のこと.

c_dlaps3

ラプラス変換 (一般関数)

<pre>ierr = c_dlaps3(fun, t, delt, np, epsr, r0, ft, t1, neps, errv, &icon);</pre>

1. 機能

関数 $F(s)$ (非有理関数でもよい) が与えられたとき, そのラプラス逆変換 $f(t)$ の値, $f(t_0), f(t_0 + \Delta t), \dots, f(t_0 + (\ell - 1)\Delta t)$ を求める.

ここで, $F(s)$ は $\text{Re}(s) > \gamma_0$ (γ_0 : 収束座標) で正則であるとし, γ_0 は既知であるとする.

2. 引数

呼出し形式:

```
ierr = c_dlaps3(fun, t, delt, np, epsr, r0, ft, t1, neps, errv, &icon);
```

引数の説明:

fun	function	入力	複素変数 s に対して関数 $F(s)$ の虚数部を計算するユーザ関数名. ("使用例" 参照) プロトタイプ宣言: double fun(dcomplex s); 引数の説明:
t	double	入力	s dcomplex 入力 複素独立変数 s .
delt	double	入力	$f(t)$ の計算開始値 t_0 (≥ 0).
np	int	入力	変数 t の増分 Δt (≥ 0).
epsr	double	入力	delt = 0 の場合は, $f(t_0)$ だけが計算される.
r0	double	入力	$f(t)$ の値を求めようとする点の個数 ($\ell \geq 1$).
ft	double ft[np]	出力	$f(t)$ の値に対する相対誤差の上限 (≥ 0). $10^{-2} \sim 10^{-7}$ 程度がよい. epsr = 0.0 と入力したときは標準値 10^{-4} が採用される.
t1	double t1[np]	出力	関数 $F(s)$ が $\text{Re}(s) > \gamma_0$ で正則であるとき, $\gamma \geq \gamma_0$ を満たす γ の値. r0 < 0.0 と入力された場合は, 本関数内で r0 = 0.0 とされる.
neps	int neps[np]	出力	$f(t_0 + i\Delta t)$ の値. ft[i] = $f(t_0 + i\Delta t)$
errv	double errv[np]	出力	$t_0 + i\Delta t$ の値. t1[i] = $t_0 + i\Delta t$
icon	int	出力	打ち切り項数 N_i . ft[i] を計算するために用いた項数 N_i が neps[i] に格納される.
			各結果 ft[i] の相対誤差. ft[i] における相対誤差が errv[i] に格納される.
			コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	結果 $f(t)$ の中に要求精度を満足しないものがある.	処理を続ける. $f(t_0 + i\Delta t), i = 0, 1, \dots, \ell - 1$ の精度は配列 errv に出力される.

コード	意味	処理内容
20000	ある np に対して $\exp(r0 \cdot t1[np] + \sigma_0) / t1[np]$ の値がオーバーフローするほど大きくなった. $\sigma_0 = \left\lceil -\frac{\log(\text{epsr})}{2} \right\rceil + 2$	処理を打ち切る. 結果は保証されない.
30000	次のいずれかであった. • $t \leq 0$ 又は $\text{delt} < 0$ • $\text{np} < 1$	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

$F(s)$ が有理関数の場合には, 関数 c_dlaps1 あるいは c_dlaps2 を使用する方が効率が良い.

b) 引数 r0

$F(s)$ が $\text{Re}(s) > \gamma_0$ で正則であるとき, 引数 r0 には $\gamma \geq \gamma_0$ なる γ を入力すること. $\gamma_0 \leq 0$ のときは $r0 = 0$ とすればよい. $r0 < 0$ と入力された場合には, 本関数内で $r0 = 0$ とみなす.

$r0 = 0$ の場合と, $r0 > 0$ の場合とで $f(t)$ が大きく変わるときには $\gamma_0 > 0$ と考えられる. 本関数を使用して γ_0 を指定する方法の詳細については“富士通 SSL II 使用手引書”の LAPS3 の項目を参照のこと.

4. 使用例

$F(s) = \text{Im}(s)$ (s は複素数) のラプラス逆変換 $f(t)$ を求める.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define L 10

double fun(dcomplex s); /* function prototype */

MAIN__()
{
    int ierr, icon;
    double t, delt, epsr, r0;
    double t1[L], ft[L], errv[L];
    int i, l, neps[L];

    /* generate initial data */
    t = 0.2;
    delt = 0.2;
    l = L;
    epsr = 1e-4;
    r0 = 0;
    /* calculate inverse Laplace transform */
    ierr = c_dlaps3(fun, t, delt, l, epsr, r0, ft, t1,
                   neps, errv, &icon);
    if (icon >= 20000) {
        printf("ERROR: c_dlaps3 failed with icon = %d\n", icon);
        exit(1);
    }
    printf("icon = %i\n", icon);
    printf(" t1      ft      errv      neps  \n");
    for (i=0; i<l; i++) {
        printf("%4.2f %12.5e %12.5e %4i\n",
              t1[i], ft[i], errv[i], neps[i]);
    }
    return(0);
}

/* user function */
```

```
double fun(dcomplex s)
{
    return s.im;
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の LAPS3 の項目を参照のこと.

c_dlaxl

実行列の最小二乗解 (ハウスホルダー変換)

<pre>ierr = c_dlaxl(a, k, m, n, b, isw, vw, ivw, &icon);</pre>

1. 機能

実係数連立 1 次方程式,

$$\mathbf{Ax} = \mathbf{b}$$

の最小二乗解 $\tilde{\mathbf{x}}$ を求める. すなわち, $\mathbf{A}$ が $m \times n$, $\text{rank}(\mathbf{A}) = n$ なる実行列であり, $\mathbf{b}$ が m 次元の実定数ベクトルであるとき,

$$\|\mathbf{b} - \mathbf{Ax}\|_2$$

が最小となる解を求める. ただし, $m \geq n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dlaxl((double*)a, k, m, n, b, isw, vw, ivw, &icon);
```

引数の説明:

a	double a[m][k]	入力	係数行列 $\mathbf{A}$. 演算後, 内容は保存されない.
k	int	入力	配列 a の整合寸法 ($\geq n$).
m	int	入力	係数行列 $\mathbf{A}$ の行数 m .
n	int	入力	係数行列 $\mathbf{A}$ の列数 n .
b	double b[m]	入力	定数ベクトル $\mathbf{b}$.
		出力	最小二乗解 $\tilde{\mathbf{x}}$. (“使用上の注意”a)参照).
isw	int	入力	制御情報. 同一の係数行列を持つ $l (\geq 1)$ 組の方程式を解くとき, 次のとおり指定する. isw = 1 のとき, 1 組目の方程式を解く. isw = 2 のとき, 2 組目以降の方程式を解く. ただし, このとき b の値だけを新しい定数ベクトル $\mathbf{b}$ の値に変え, それ以外の引数はそのまま使う. (“使用上の注意”b)参照).
vw	double vw[2n]	作業領域	
ivw	int ivw[n]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	$\text{Rank}(\mathbf{A}) < n$ であった.	処理を打ち切る.
29000	作業用配列を確保できなかった.	処理を打ち切る.

コード	意 味	処 理 内 容
30000	次のいずれかであった。 • $k < n$ • $m < n$ • $n < 1$ • $isw \neq 1, 2$	処理を打ち切る。

3. 使用上の注意

a) 最小二乗解 **b** について

最小二乗解 $\tilde{\mathbf{x}}$ は配列 **b** の先頭 n 要素に格納される。

b) **isw** について

同一の係数行列を持つ連立 1 次方程式をいくつか続けて解く場合には、2 回目以降 $isw = 2$ として解くと、係数行列 **A** の三角行列化過程を省略するので計算時間が少なくなる。

4. 使用例

初期化した **A** と **x** の積 **b** を求め、 $\mathbf{Ax}=\mathbf{b}$ を解く。**A** と得られた解 **y** との積を **b** と比較し、結果を確認する。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define MMAX 110
#define NMAX 100

MAIN__()
{
    int ierr, icon;
    int m, n, i, j, k, isw;
    double eps;
    double a[MMAX][NMAX], aa[MMAX][NMAX], b[MMAX], bb[MMAX];
    double x[MMAX], vw[2*NMAX];
    int ivw[NMAX];

    /* initialize overdetermined system */
    m = MMAX;
    n = NMAX;
    for (i=0; i<n; i++)
        for (j=i; j<n; j++) {
            a[i][j] = n-j;
            a[j][i] = n-j;
        }
    for (i=n; i<m; i++)
        for (j=0; j<n; j++) {
            a[i][j] = 0;
            if (i%n == j) a[i][j] = 1;
        }
    for (i=0; i<m; i++)
        for (j=0; j<n; j++)
            aa[i][j] = a[i][j];
    for (i=0; i<n; i++)
        x[i] = 1;
    k = NMAX;
    /* initialize constant vector b = a*x */
    ierr = c_dmav((double*)a, k, m, n, x, b, &icon);
    for (i=0; i<m; i++)
        bb[i] = b[i];
    isw = 1;
    /* solve overdetermined system of equations */
    ierr = c_dlaxl((double*)a, k, m, n, b, isw, vw, ivw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dlaxl failed with icon = %d\n", icon);
        exit(1);
    }
}
```

```
    }
    /* check least squares solution */
    ierr = c_dmav((double*)aa, k, m, n, b, x, &ierr);
    eps = 1e-6;
    for (i=0; i<m; i++)
        if (fabs((x[i]-bb[i])/bb[i]) > eps) {
            printf("WARNING: result inaccurate\n");
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の LAXL の項目及び[18]を参照のこと.

c_dlaxlm

実行列の最小二乗最小ノルム解 (特異値分解法)
ierr = c_dlaxlm(a, ka, m, n, b, isw, eps, sig, v, kv, vw, &icon);

1. 機能

$m \times n$ の実行列 **A** を係数とする連立 1 次方程式,

$$\mathbf{Ax} = \mathbf{b}$$

の最小二乗最小ノルム解 $\mathbf{x}^+$ を求める.

すなわち, **b** が m 次元の実定数 ベクトルであるとき,

$$\|\mathbf{b} - \mathbf{Ax}\|_2$$

が最小となる n 次元のベクトル **x** の中で,

$$\|\mathbf{x}\|_2$$

が最小となるものを求める. ただし, $m \geq 1, n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dlaxlm((double*)a, ka, m, n, b, isw, eps, sig, (double*)v, kv, vw, &icon);
```

引数の説明:

a	double a[m][ka]	入力	係数行列 A . 演算後, 内容は保存されない.
ka	int	入力	配列 a の整合寸法 ($\geq n$).
m	int	入力	係数行列 A の行数 m .
n	int	入力	係数行列 A の列数, 行列 v の行数 n .
b	double b[Blen]	入力 出力	定数ベクトル b . $Blen = \max(m, n)$. 最小二乗最小ノルム解 $\mathbf{x}^+$. (“使用上の注意”a)参照).
isw	int	入力	制御情報. 同一の係数行列を持つ方程式を 1 組だけ解く場合 及び複数組解く場合に依じて, 次のように指定する. $isw = 0$ のとき, 1 組だけ方程式を解く. $isw = 1$ のとき, 1 組目の方程式を解き, 2 組目以降の方程式を 解くための情報を残す. $isw = 2$ のとき, 2 組目以降の方程式を解く. ただし, このとき b の値だけを新しい定数ベクトル b の値 に変え, それ以外の 引数は, $isw = 1$ として用いたときのままで使う. (“使用上の 注意”b)参照).
eps	double	入力	特異値の相対零判定値. (≥ 0.0) 0.0 のときは標準値が採用される. (“使用上の注意”d)参照).
sig	double sig[n]	出力	A の特異値. (“使用上の注意”c)参照).

v	double v[n][kv]	作業領域	特異値分解 $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^T$ の $\mathbf{U}$ と $\mathbf{V}$ のための作業領域. ("使用上の注意"b)参照).
kv	int	入力	配列 v の整合寸法 ($\geq \min(m+1, n)$).
vw	double vw[n]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
15000	ある特異値が求められなかった.	処理を打ち切る.
29000	作業用配列を確保できなかった.	処理を打ち切る.
30000	次のいずれかであった. • $ka < n$ • $m < 1$ • $n < 1$ • $kv < \min(m+1, n)$ • $eps < 0$ • $isw \neq 0, 1, 2$	処理を打ち切る.

3. 使用上の注意

a) b について

最小二乗最小ノルム解 $\mathbf{x}^+$ は配列 b の先頭 n 要素に格納される.

b) isw について

最小二乗最小ノルム解を一つだけ求めるときには, $isw = 0$ と指定すると, 特異値分解による変換行列を計算しないので, 計算量が少なく有利である.

なお, このとき配列 v の先頭の l 列に, 行列 $\mathbf{V}$ が格納される. この行列 $\mathbf{V}$ は a の上に重ね書きができるように配慮されている. ここで $l = \min(m, n)$.

同一の係数行列を持つ複数組の連立一次方程式の最小二乗最小ノルム解を続けて求める場合には, 1 組目は $isw = 1$ と指定し, 2 組目以降は $isw = 2$ と指定すれば, 2 組目以降は, 係数行列の特異値分解を省略するので計算時間が少なくなる. なお, このときには, a の先頭の l 列には行列 $\mathbf{U}$ が, v の先頭の l 列には行列 $\mathbf{V}$ が格納されている.

c) sig について

特異値はすべて非負で, 大きさの減少する順に格納される. $icon = 15000$ の場合には, 求められなかった特異値は-1 とし, 大小順に整列しない.

d) eps について

入力引数 eps は A の階数を決定するための重要な役目を果たすので, その選択は慎重に行わなければならない.

この判定基準 eps より小さい特異値は 0 とみなされる. eps として 0.0 が入力されたときは, $eps = 16\mu$ を標準値として用いる. ただし, μ は丸め誤差の単位である. 0 より小さい値が入力されたときは $icon = 30000$ となる.

e) 本関数の使用について

本関数は A に階数落ち ($\text{rank}(\mathbf{A}) < \min(m, n)$) があるか, 又はそのおそれがあるときに用いるべきで, 階数落ちがないことが明白な場合には, 関数 c_dvlaxl を用いるのが合理的である.

4. 使用例

初期化した **A** と **x** の積 **b** を求め、 $\mathbf{Ax} = \mathbf{b}$ を解く。**A** と得られた解 **y** との積を **b** と比較し、結果を確認する。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define MMAX 110
#define NMAX 100

MAIN__()
{
    int ierr, icon;
    int m, n, i, j, ka, kv, isw;
    double eps;
    double a[MMAX][NMAX], aa[MMAX][NMAX], b[MMAX], bb[MMAX];
    double x[MMAX], sig[NMAX], v[NMAX][NMAX], vw[NMAX];

    /* initialize overdetermined system */
    m = MMAX;
    n = NMAX;
    for (i=0; i<n; i++)
        for (j=i; j<n; j++) {
            a[i][j] = n-j;
            a[j][i] = n-j;
        }
    for (i=n; i<m; i++)
        for (j=0; j<n; j++) {
            a[i][j] = 0;
            if (i%n == j) a[i][j] = 1;
        }
    for (i=0; i<m; i++)
        for (j=0; j<n; j++)
            aa[i][j] = a[i][j];
    for (i=0; i<n; i++)
        x[i] = 1;
    ka = NMAX;
    kv = NMAX;
    /* initialize constant vector b = a*x */
    ierr = c_dmav((double*)a, ka, m, n, x, b, &icon);
    for (i=0; i<m; i++)
        bb[i] = b[i];
    isw = 0;
    eps = 0;
    /* solve overdetermined system of equations */
    ierr = c_dlaxlm((double*)a, ka, m, n, b, isw,
                   eps, sig, (double*)v, kv, vw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dlaxlm failed with icon = %d\n", icon);
        exit(1);
    }
    /* check least squares solution */
    ierr = c_dmav((double*)aa, ka, m, n, b, x, &icon);
    eps = 1e-6;
    for (i=0; i<m; i++)
        if (fabs((x[i]-bb[i])/bb[i]) > eps) {
            printf("WARNING: result inaccurate\n");
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の LAXLM の項目及び[41]を参照のこと。

c_dlcx

複素行列の連立 1 次方程式(クラウト法)
ierr = c_dlcx(za, k, n, zb, epsz, isw, &is, zvw, ip, &icon);

1. 機能

複素係数連立 1 次方程式をクラウト法で解く.

$$\mathbf{Ax} = \mathbf{b}$$

ただし, $\mathbf{A}$ は $n \times n$ の正則な複素行列, $\mathbf{b}$ は n 次元の複素定数ベクトル, $\mathbf{x}$ は n 次元の解ベクトルである. また, $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dlcx((dcomplex*)za, k, n, zb, epsz, isw, &is, zvw, ip, &icon);
```

引数の説明:

za	dcomplex za[n][k]	入力	係数行列 $\mathbf{A}$. 演算後, 内容は保存されない.
k	int	入力	配列 za の整合寸法 ($\geq n$).
n	int	入力	係数行列 $\mathbf{A}$ の次数 n .
zb	dcomplex zb[n]	入力	定数ベクトル $\mathbf{b}$.
		出力	解ベクトル $\mathbf{x}$.
epsz	double	入力	ピボットの相対零判定値 (≥ 0.0). 0.0 のときは標準値が採用される. (“使用上の注意” a)参照).
isw	int	入力	制御情報. 同一の係数行列を持つ l (≥ 1)組の方程式を解くとき, 次のとおり指定する. $isw=1$ のとき, 1 組目の方程式を解く. $isw=2$ のとき, 2 組目以降の方程式を解く. ただし, このとき $\mathbf{b}$ の値だけを新しい定数ベクトル $\mathbf{b}$ の値に変え, それ以外の引数はそのまま使う. (“使用上の注意” b)参照).
is	int	出力	行列 $\mathbf{A}$ の行列式を求めるための情報. 演算後の配列 za の n 個の対角要素と is の値を掛け合わせると行列式が得られる.
zvw	dcomplex zvw[n]	作業領域	
ip	int ip[n]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.

コード	意 味	処 理 内 容
20000	係数行列のある行の要素がすべて零であったか又はピボットが相対的に零となった. 係数行列は非正則の可能性が高い.	処理を打ち切る.
30000	次のいずれかであった. • $k < n$ • $n < 1$ • $\text{epsz} < 0$ • $\text{isw} \neq 1, 2$	処理を打ち切る.

3. 使用上の注意

a) epsz について

ピボットの相対零判定値 epsz に 10^{-5} を設定したとすると, この値は次の意味を持っている. すなわち, クラウト法による LU 分解の過程でピボットの実部と虚部の値が, 同時に 10 進 s 桁以上の桁落ちを生じた場合にそのピボットを相対的に零とみなし, $\text{icon} = 20000$ として処理を打ち切る.

epsz の標準値は丸め誤差の単位を μ としたとき, $\text{epsz} = 16\mu$ である.

なお, ピボットが小さくなくても計算を続行させる場合には, epsz へ極小の値を与えればよいが, その結果は保証されない.

b) isw について

同一の係数行列を持つ連立一次方程式をいくつか続けて解く場合には, 2 回目以降 $\text{isw} = 2$ として解くと, 係数行列 A の LU 分解過程を省略するので計算時間が少なくなる. なお, この場合 is の値は, $\text{isw} = 1$ のときの値が保障される.

4. 使用例

行列 A とベクトル x の積 b を求め, $Ax = b$ を解く. 得られた解と元の x の値を比較し, 結果を確認する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 100

MAIN__()
{
    int ierr, icon;
    int n, i, j, k, is, isw;
    double epsz, eps;
    dcomplex za[NMAX][NMAX];
    dcomplex zb[NMAX], zx[NMAX], zvw[NMAX];
    int ip[NMAX];

    /* initialize matrix and vector */
    n = NMAX;
    k = NMAX;
    for (i=0; i<n; i++) {
        for (j=i; j<n; j++) {
            za[i][j].re = n-j;
            za[i][j].im = n-j;
            za[j][i].re = n-j;
            za[j][i].im = n-j;
        }
        zx[i].re = i+1;
        zx[i].im = i+1;
    }
}
```

```

}
/* initialize constant vector zb = za*zx */
ierr = c_dmcv((dcomplex*)za, k, n, n, zx, zb, &icon);
epsz = 1e-6;
isw = 1;
/* solve system of equations */
ierr = c_dlcx((dcomplex*)za, k, n, zb, epsz, isw, &is, zvw, ip, &icon);
if (icon != 0) {
    printf("ERROR: c_dlcx failed with icon = %d\n", icon);
    exit(1);
}
/* check result */
eps = 1e-6;
for (i=0;i<n;i++)
    if (fabs((zb[i].re-zx[i].re)/zx[i].re) > eps ||
        fabs((zb[i].im-zx[i].im)/zx[i].im) > eps) {
        printf("WARNING: result inaccurate\n");
        exit(1);
    }
printf("Result OK\n");
return(0);
}

```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の LCX の項目及び[7], [34], [83]を参照のこと.

c_dlesq1

最小二乗近似多項式

ierr = c_dlesq1(x, y, n, &m, w, c, vw, &icon);

1. 機能

n 個の離散点 $x_1, x_2, \dots, x_n$ に対して, 観測値 y_i 及び重み関数値 $w(x_i)$, $i=1, 2, \dots, n$ が与えられたとき, これに対する最小二乗近似多項式を求める.

すなわち, m 次の多項式 $\bar{y}_m(x)$ を,

$$\bar{y}_m(x) = c_0 + c_1x + \dots + c_mx^m$$

とするとき,

$$\delta_m^2 = \sum_{i=1}^n w(x_i) [y_i - \bar{y}_m(x_i)]^2$$

を最小にするような $\bar{y}_m(x)$ を求める.

ここで m は $0 \leq m \leq k$ なる範囲で,

$$\text{AIC} = n \log \delta_m^2 + 2m$$

なる量を最小にするように選ばれ, このときの m を最小二乗近似多項式の最良なる次数とする.

ただし, $0 \leq k \leq n-1$ とする. また重み関数値 $w(x_i)$ は $w(x_i) \geq 0$, $n \geq 2$ なる条件を満たすものであること.

2. 引数

呼出し形式:

```
ierr = c_dlesq1(x, y, n, &m, w, c, vw, &icon);
```

引数の説明:

x	double x[n]	入力	離散点 x_i .
y	double y[n]	入力	観測値 y_i .
n	int	入力	離散点の個数 n .
m	int	入力	求める近似多項式の次数の上限 k . ただし, $m = -k$ ($k > 0$) と入力すると, 無条件に k 次の近似多項式を求める.
		出力	得られた近似多項式の次数 m ($\leq k$). したがって, $m = -k$ と入力したときは, 出力時の m の値は常に k となる.
w	double w[n]	入力	重み関数値 $w(x_i)$.
			(“使用上の注意”a) 参照)
c	double c[Clen]	出力	得られた近似多項式の係数 c_i . $Clen = k+1$. 引数 m の出力時の値を m ($0 \leq m \leq k$) とすると, $c_0, c_1, \dots, c_m$ の順に格納する. このとき, $m < k$ となった場合は, 要素 $c[i]$, $i = m+1, \dots, k$ には 0.0 を格納する.
vw	double vw[7*n]	作業領域	

icon int 出力 コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
10000	$m = -k$ ($k > 0$) と指定されたが, k 次の近似多項式が一意的に決まらなかった.	一意的に決めることができる範囲で最高次 (しかし k 次よりは低い) の近似多項式を出力する.
30000	次のいずれかであった. • $n < 2$ • $k > n-1$ • w の中に負のものがある.	処理を打ち切る.

3. 使用上の注意

a) 重み関数 $w(x_i)$ の与え方

観測値 $\{y_i\}$ がオーダ的にそろっている場合は, $w(x_i) = 1$, $i = 1, 2, \dots, n$ でよいが, オーダ的にまちまちである場合は $w(x_i) = 1/y_i^2$ ($y_i = 0$ のときは $w(x_i) = 1$) と与えるのがよい.

b) 離散点の個数

離散点の個数 n は次数の上限 k に比べてなるべく多くとるべきである. 理論的には $10k \leq n$ を満たすのがよいとされている.

4. 使用例

関数 $f(x) = \sin(x)\sqrt{x}$ の 5 次の最小二乗近似多項式を求める.

```
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 10
#define MMAX 5

MAIN__()
{
    int ierr, icon;
    int i, n, m;
    double x[NMAX], y[NMAX], w[NMAX], c[MMAX+1], vw[7*NMAX];
    double h, p;

    /* initialize data */
    n = NMAX;
    p = 0;
    h = 1.0/n;
    for (i=0; i<n; i++) {
        w[i] = 1;
        x[i] = p;
        y[i] = sin(p)*sqrt(p);
        p = p + h;
    }
    m = MMAX;
    /* calculate polynomial least squares coefficients */
    ierr = c_dlesq1(x, y, n, &m, w, c, vw, &icon);
    printf("icon = %i    m = %i\n", icon, m);
    for (i=0; i<m; i++)
        printf("%12.4e ", c[i]);
    printf("\n");
}
```

```
        return(0);  
    }
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の LESQ1 の項目及び[89]を参照のこと.

c_dlminf

1変数関数の極小化 (微係数不要, 2次補間法)

```
ierr = c_dlminf(&a, b, fun, epsr, &max, &f,
               &icon);
```

1. 機能

1変数の実関数 $f(x)$ が与えられたとき, 区間 $[a, b]$ で $f(x)$ の極小点 x^* とその関数値 $f(x^*)$ を求める.

ただし, $f(x)$ は2階までの連続な微係数を持つものと仮定する.

2. 引数

呼出し形式:

```
ierr = c_dlminf(&a, b, fun, epsr, &max, &f, &icon);
```

引数の説明:

a	double	入力	区間 $[a, b]$ の端点 a .
		出力	極小点 x^* .
b	double	入力	区間 $[a, b]$ の端点 b .
fun	function	入力	$f(x)$ を計算するユーザ関数名. プロトタイプ宣言: double fun(double x); 引数の説明: x double 入力 変数 x.
epsr	double	入力	収束判定値. $\text{epsr} = 0$ のときは標準値が採用される. ("使用上の注意" a)参照)
max	int	入力	関数 $f(x)$ の評価回数の上限. ("使用上の注意" b)参照)
		出力	実際に評価した回数.
f	double	出力	関数値 $f(x^*)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	指定された関数の評価回数内で, 収束条件が満たされなかった.	引数 a, f には最後の値を格納する.
30000	次のいずれかであった. • $\text{epsr} < 0$ • $\text{max} = 0$	処理を打ち切る.

3. 使用上の注意

a) 引数 **epsr** の与え方

本関数の収束判定は, 反復の過程で点 x^* をはさむ2点 x_1, x_2 に関して

$$|x_1 - x_2| \leq \max(1, |\tilde{x}|) \cdot \text{epsr}$$

となった場合に、点 $\tilde{x}$ を極小点 x^* とみなし反復を終了する。ここで $\tilde{x}$ は、 $\tilde{x} = x_1$ ($f(x_1) \leq f(x_2)$ のとき)、又は $\tilde{x} = x_2$ ($f(x_1) > f(x_2)$ のとき)である。

本関数では、関数 $f(x)$ が点 x^* の近傍で近似的に 2 次関数であることを前提としており、極小点 x^* を丸め誤差程度まで正しく求めるためには、

$$\text{epsr} \approx \sqrt{\mu}, \mu \text{ は丸め誤差の単位}$$

程度に与えるのが適当である。

epsr の標準値は $2 \cdot \sqrt{\mu}$ である。

b) 引数 max の与え方

関数の評価回数は、変数 x に対して、 $f(x)$ を計算することを 1 回と数える。すなわち、ユーザ関数 fun の呼び出し回数に一致する。

この評価回数は、関数 $f(x)$ の性質、区間 $[a, b]$ の与え方、収束判定値に依存する。

一般には、区間 $[a, b]$ が適当に与えられ、収束判定値として標準値を採用した場合には、max = 400 程度が妥当である。

指定評価回数内で収束条件が満たされず、icon = 10000 として戻った場合でも、再度本関数を呼び出すことにより、継続して反復することができる。しかし、この場合は引数 max には、負の値で追加評価回数を指定し、他の引数の内容は保存したままで呼び出すこと。

c) 引数 a 及び引数 b

本関数では、 $f(x)$ が区間 $[a, b]$ で唯一の極小点を持つ場合、その値を許容誤差の範囲内で求める。区間 $[a, b]$ に多くの極小点を持つ場合には、いずれの極小点に収束するかは保証されない。

したがって、求める極小点 x^* を含む区間の端点 a 及び b の値は、可能なかぎり x^* の近くにとることが望ましい。

4. 使用例

関数 $f(x) = x^4 - 4x^3 - 6x^2 - 16x + 4$ の区間 $[-5, 5]$ での極小値を求める。

```
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

double fun(double x); /* user function prototype */

MAIN__()
{
    int ierr, icon;
    double a, b, f, epsr, eps, exact;
    int max;

    /* initialize data */
    a = -5;
    b = 5;
    epsr = 0;
    max = 400;
    /* find minimum of function */
    ierr = c_dlminf(&a, b, fun, epsr, &max, &f, &icon);
    printf("icon = %i   max = %i   a = %12.4e   f = %12.4e\n", icon, max, a, f);
    /* check result */
    exact = 4;
    eps = 1e-6;
    if (fabs((a-exact)/exact) > eps)
```



```
        printf("Inaccurate result\n");
    else
        printf("Result OK\n");
    return(0);
}

/* user function */
double fun(double x)
{
    return((((x-4)*x-6)*x-16)*x+4);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の LMINF の項目を参照のこと。

c_dlming

1 変数関数の極小化 (微係数要, 3 次補間法)

<pre>ierr = c_dlming(&a, b, fun, grad, epsr, &max, &f, &icon);</pre>

1. 機能

1 変数の実関数 $f(x)$ とその導関数 $g(x)$ が与えられたとき, 区間 $[a, b]$ で $f(x)$ の極小化 x^* とその関数値 $f(x^*)$ を求める. ただし, $f(x)$ は 3 階までの連続な微係数を持つものと仮定する.

2. 引数

呼出し形式:

```
ierr = c_dlming(&a, b, fun, grad, epsr, &max, &f, &icon);
```

引数の説明:

a	double	入力	区間 $[a, b]$ の端点 a .
		出力	極小点 x^* .
b	double	入力	区間 $[a, b]$ の端点 b .
fun	function	入力	$f(x)$ を計算するユーザ関数名. プロトタイプ宣言: double fun(double x); 引数の説明: x double 入力 変数 x .
grad	function	入力	$g(x)$ を計算するユーザ関数名. プロトタイプ宣言: double grad(double x); 引数の説明: x double 入力 変数 x .
epsr	double	入力	収束判定値 (≥ 0). epsr = 0.0 のときは標準値が採用される. ("使用上の注意" a)参照)
max	int	入力	関数 $f(x)$ 及び $g(x)$ の評価回数の上限 ($\neq 0$). ("使用上の注意" b)参照)
		出力	実際に評価した回数 (> 0).
f	double	出力	関数値 $f(x^*)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	指定された関数の評価回数内で, 収束条件が満たされなかった.	引数 a, f には最後の値を格納する.
20000	epsr が小さすぎる.	処理を打ち切る. 引数 a, f には最後の値を格納する.

コード	意味	処理内容
30000	次のいずれかであった. • $\text{epsr} < 0$ • $\text{max} = 0$	処理を打ち切る.

3. 使用上の注意

a) 引数 **epsr**

本関数の収束判定は, 反復の過程で点 x^* をはさむ 2 点 x_1, x_2 に関して

$$|x_1 - x_2| \leq \max(1, |\tilde{x}|) \cdot \text{epsr}$$

となった場合に, 点 $\tilde{x}$ を極小点とみなし反復を終了する. ここで $\tilde{x}$ は, $\tilde{x} = x_1$ ($f(x_1) \leq f(x_2)$ のとき), 又は $\tilde{x} = x_2$ ($f(x_1) > f(x_2)$ のとき)である.

本関数では, 関数 $f(x)$ が点 x^* の近傍で近似的に 3 次関数であることを前提としており, 極小点 x^* を丸め誤差程度まで正しく求めるためには, $\text{epsr} \approx \mu^{1/2}$ (μ は丸め誤差の単位)程度に与えるのが適当である. epsr の標準値は $2\mu^{1/2}$ である.

b) 引数 **max**

関数の評価回数は, 変数 x に対して, $f(x)$ を計算することを 1 回, $g(x)$ を計算することを 1 回と数える. すなわち, ユーザ関数 **fun** 及び **grad** の呼び出し回数に一致する.

この評価回数は, 関数 $f(x)$, $g(x)$ の性質, 区間 $[a, b]$ の与え方, 収束判定値に依存する. 一般には, 区間 $[a, b]$ が適当に与えられ, 収束判定値として標準値を採用した場合には, $\text{max} = 400$ 程度が妥当である.

指定評価回数内で収束条件が満たされず, $\text{icon} = 10000$ として戻った場合でも, 再度本関数を呼び出すことにより, 継続して反復することができる. しかし, この場合は引数 **max** には, 負の値で追加評価回数を指定し, 他の引数の内容は保存したままで呼び出すこと.

c) 引数 **a** 及び **b**

本関数では, $f(x)$ が区間 $[a, b]$ で唯一の極小点をもつ場合, その値を許容誤差の範囲内で求める. 区間 $[a, b]$ に多くの極小点を持つ場合には, いずれの極小点に収束するかは保証されない.

4. 使用例

$f(x) = x^4 - 4x^3 - 6x^2 - 16x + 4$ の区間 $[-5, 5]$ の極小値を求める. ただし, $f(x)$ の導関数 $g(x)$ は $g(x) = 4x^3 - 12x^2 - 12x - 16$ であり, 収束判定値は標準値を与える.

```
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

double fun(double x); /* user function prototype */
double grad(double x); /* derivative prototype */

MAIN__()
{
    int ierr, icon;
    double a, b, f, epsr, eps, exact;
    int max;

    /* initialize data */
    a = -5;
    b = 5;
    epsr = 0;
    max = 400;
    /* find minimum of function */
    ierr = c_dlming(&a, b, fun, grad, epsr, &max, &f, &icon);
    printf("icon = %i    max = %i    a = %12.4e    f = %12.4e\n", icon, max, a, f);
    /* check result */
}
```

```
    exact = 4;
    eps = 1e-6;
    if (fabs((a-exact)/exact) > eps)
        printf("Inaccurate result\n");
    else
        printf("Result OK\n");
    return(0);
}

/* user function */
double fun(double x)
{
    return((((x-4)*x-6)*x-16)*x+4);
}

/* derivative function */
double grad(double x)
{
    return ((4*x-12)*x-12)*x-16;
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”のLMINGの項目を参照のこと.

c_dlowp

実係数低次代数方程式 (5 次以下)
ierr = c_dlowp(a, n, z, &icon);

1. 機能

実係数低次代数方程式,

$$a_0x^n + a_1x^{n-1} + \dots + a_n = 0 \quad (n \leq 5, a_0 \neq 0)$$

の根を逐次代入法, ニュートン法, フェラリー法, ベアストウ法, 2 次方程式の根の公式により求める.

2. 引数

呼出し形式:

```
ierr = c_dlowp(a, n, z, &icon);
```

引数の説明:

a	double a[n+1]	入力	代数方程式の係数. a[i]=a _i (i=0, ..., n)
n	int	入力	代数方程式の次数 n.
z	dcomplex z[n]	出力	n 個の根.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
10000	5 次方程式の 1 つの実根を求めるとき 50 回の逐次代入を行なっても $f(x_k)f(x_{k+1}) < 0$ とならなかった.	x_{k+1} をニュートン法の初期値として処理を続ける.
30000	次のいずれかであった. $a_0 = 0$ $n \leq 0$ $n > 5$	処理を打ち切る.

3. 使用例

実係数 3 次代数方程式 $z^3 - 6z^2 + 11z - 6 = 0$ の根を求める.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 5

MAIN__()
{
    int ierr, icon;
    dcomplex z[NMAX];
    double a[NMAX+1];
    int n, i;
```

```
/* initialize data */
n = 3;
a[0] = 1;
a[1] = -6;
a[2] = 11;
a[3] = -6;
/* find roots of polynomial */
ierr = c_dlowp(a, n, z, &icon);
printf("icon = %i\n", icon);
for (i=0; i<n; i++)
    printf("z[%i] = {%12.4e, %12.4e}\n", i, z[i].re, z[i].im);
printf("exact roots are: {1, 0}, {2, 0} and {3, 0}\n");
return(0);
}
```

4. 手法概要

異なった次数 (≤ 5) の実係数代数方程式に関して、根を求めるのに使用される公式を以下に示す。

- 1 次方程式: 割り算で計算する.
- 2 次方程式: 根の公式を使って計算する. (関数: `c_drqdr` を使用)
- 3 次方程式: ニュートン法と根の公式を使って計算する.
- 4 次方程式: フェラリー法とベアストウ法を使って計算する.
- 5 次方程式: ニュートン法と逐次代入法を使って計算する.

計算方法の詳細については“富士通 SSL II 使用手引書”の LOWP の項目及び[124]を参照のこと.

c_dlprs1

線形計画問題 (改訂シンプレックス法)

```
ierr = c_dlprs1(a, k, m, n, epsz, &imax,
               &isw, nbv, b, kb, vw, ivw,
               &icon);
```

1. 機能

次の線形計画問題を改訂シンプレックス法により解く.

“条件

$$\begin{aligned} \sum_{j=1}^n a_{ij}x_j &\leq d_i \quad i=1,2,\dots,m_l \\ \sum_{j=1}^n a_{ij}x_j &\geq d_i \quad i=m_l+1,m_l+2,\dots,m_l+m_g \\ \sum_{j=1}^n a_{ij}x_j &= d_i \quad i=m_l+m_g+1,m_l+m_g+2,\dots,m_l+m_g+m_e \\ x_j &\geq 0, \quad j=1,2,\dots,n \end{aligned}$$

のもとで, 線形の目的関数

$$z = \sum_{j=1}^n c_j x_j + c_0 = \mathbf{c}^T \mathbf{x} + c_0$$

を最小 (又は最大) にせよ.”

本関数では, 解法が2段階に分かれている.

- 第1段階: 可能基底解を求める.
- 第2段階: 最適解を求める.

初期可能基底を入力することもできる. また d_i の符号に制約はない.

以降, $m = m_l + m_g + m_e$ とし, 要素 $\{a_{ij}\}$ によりなる $m \times n$ 行列を係数行列 $\mathbf{A}$, $\mathbf{d} = (d_1, d_2, \dots, d_m)^T$ を定数ベクトル, $\mathbf{c} = (c_1, c_2, \dots, c_n)^T$ を係数ベクトル, c_0 を定数項と呼ぶ.

ここで, $n \geq 1$, $m_l \geq 0$, $m_g \geq 0$, $m_e \geq 0$, $m \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dlprs1((double *)a, k, m, n, epsz, &imax, &isw, nbv, (double *)b,
               kb, vw, ivw, &icon);
```

引数の説明:

a	double a[m+1][k]	入力	係数行列 $\mathbf{A}$, 定数ベクトル $\mathbf{d}$, 係数ベクトル $\mathbf{c}$, 定数項 c_0 . (“使用上の注意”b)参照)
k	int	入力	配列 a の整合寸法 ($k > n$).
m	int m[3]	入力	条件式の個数. m[0] = m_l , m[1] = m_g , m[2] = m_e

n	int	入力	変数の個数 n .
epsz	double	入力	反復過程での要素 (係数及び定数項), 基底逆行列 $\mathbf{B}^{-1}$ を求めるときのパivotに対する相対零判定値. 0 のときは, 標準値が採用される. (“使用上の注意”e)参照)
imax	int	入力	反復回数の上限. (“使用上の注意”d)参照)
isw	int	出力	実際に反復した回数.
		入力	制御情報. 最小化問題か最大化問題か, 及び初期可能基底を与えるか否かを指定する. $isw = 10d_1 + d_0$ なる値を入力する. ここで, $0 \leq d_0, d_1 \leq 1$ であること. d_0 目的関数の最小化問題か最大化問題のどちらを解くか指定する. 0 最小化問題を解く. 1 最大化問題を解く. d_1 初期可能基底が提供されるかどうかを指定する. 0 基底を与えない. 1 基底を与える.
		出力	最適解又は可能基底解が得られた場合, $d_1 = 1$ となる. それ以外の場合は, 入力した値が保存される.
nbv	int nbv[m]	入力	可能基底解の基底変数の番号. ($isw = 10$ 又は $isw = 11$ と指定したとき) (“使用上の注意”a)参照)
b	double b[m+1][kb]	出力	最適解又は可能基底解の基底変数の番号.
		出力	最適解又は可能基底解に対する基底逆行列 $\mathbf{B}^{-1}$, 最適解又は可能基底解 $\mathbf{x}_B$, シンプレックス乗数 $\boldsymbol{\pi}$, 目的関数の値 q . (“使用上の注意”c)参照)
kb	int	入力	配列 $\mathbf{b}$ の整合寸法. ($kb \geq m+1$)
vw	double vw[Rlen]	作業領域	$Rlen = 2n + m + m_l + m_g + 1$
ivw	int ivw[llen]	作業領域	$llen = n + m_l + m_g$
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	可能基底解は得られたが最適解は存在しない.	配列 $\mathbf{b}$ には, 可能基底解と対応する基底逆行列, シンプレックス乗数, 目的関数の値を格納する.
11000	第2段階の途中で反復回数が, 指定された上限に達した. 可能基底解は得られている.	配列 $\mathbf{nbv}$ には, 可能基底解の基底変数の番号を格納する.

コード	意味	処理内容
20000	可能解は存在しない. epsz の値が適切でない可能性がある.	処理を打ち切る.
21000	isw = 10 又は isw = 11 のとき, 与えられた変数の組は基底ではない.	
22000	isw = 10 又は isw = 11 のとき, 与えられた変数の組を基底とする基底解は, 可能解でない.	
23000	第 1 段階の途中で, 基底変数の交換が行えなくなった. epsz の値が適切でない可能性がある.	
24000	第 1 段階の途中で反復回数が, 指定された上限に達した.	
30000	次のいずれかであった. • $m[0], m[1], m[2]$ の中に負のものがあつた. • $n < 1$ • $imax = 0$ • $epsz < 0$ • $m[0] + m[1] + m[2] < 1$ • $m[0] + m[1] + m[2] \geq k$ • nbv の要素の中に 0 又は負のものがあつた. • nbv の要素の中に, $n + m[0] + m[1]$ より大きいものがあつた. • nbv の要素の中に, 同じものがあつた. • isw の与え方に誤りがあつた.	処理を打ち切る.

3. 使用上の注意

a) 引数 nbv の内容

isw = 10 又は 11 として基底変数の番号を nbv に指定する場合, i 番目 ($i \leq m_l + m_g$) の不等式条件に対するスラック変数の番号は $n + i$ とすること.

同様に計算結果の基底解に i 番目のスラック変数が含まれる場合は, $n + i$ として与えられる.

一方, 人為変数については, 入力時に基底変数として指定することはできない. ただし, 計算結果の基底解の中に人為変数が混入する場合, nbv に 0 の値が与えられる. すなわち, i 番目の基底変数が人為変数であるときは, $nbv[i-1] = 0$ になっている.

可能基底解が得られているとき (icon = 0, 10000 又は 11000 のとき), $nbv[i-1] = 0$ は i 番目の条件式がむだな条件式であることを示している. (むだな条件式とは, $nbv[i-1] \neq 0$ なるいくつかの条件式より導かれる条件式である.)

b) 配列 a の内容

配列 a に格納する各値の内容を図 dlprs1-1 に示す. このうち係数ベクトル $\mathbf{c}^T$ は符号を逆にして設定する必要がある.

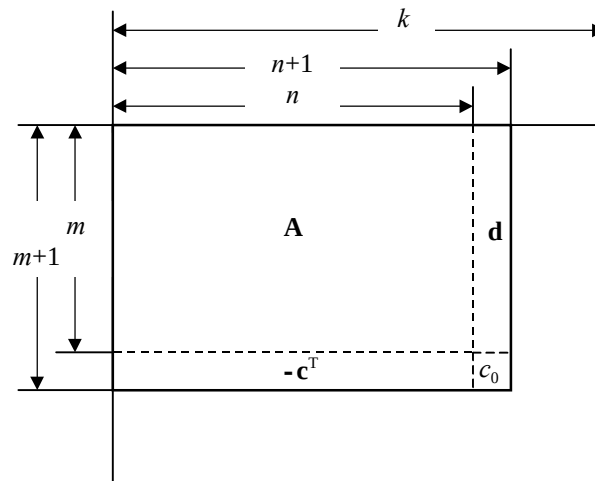


図 dlprs1-1 配列 a の内容

c) 配列 b の内容

配列 b に出力される内容を図 dlprs1-2に示す.

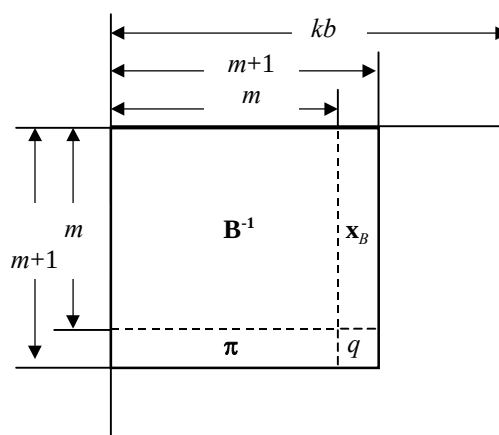


図 dlprs1-2 配列 b の内容

d) 引数 imax の与え方

反復回数は, 可能基底解 (第 1 段階では人為変数を含んだ問題の可能基底解) に対する最適性基準の判定回数を指す.

imax の標準値は $10m$ である.

指定反復回数内で最適解が得られず, icon = 11000 として戻った場合でも, 再度本関数を呼び出すことにより, 継続して反復することができる. しかし, この場合は引数 imax には, 負の値で追加反復回数を指定し, 他の引数の内容は保存したまま呼び出すこと.

e) 引数 epsz の与え方

入力データ(配列 a の内容)の要素の絶対値の最大値を $a_{\max}$ とすると, 反復過程での要素の絶対値が $\text{epsz} \cdot a_{\max}$ より小さい要素は零とみなされる.

また, 基底逆行列 B^{-1} を LU 分解により求める場合には, ピボットの相対零判定値として使用される.

epsz の標準値は丸め誤差の単位を μ としたとき, $\text{epsz} = 16\mu$ である.

入力データ (配列 a の内容) は, 行又は列毎に定数を乗じて, 絶対値をできるだけ揃えておくことが望ましい. なお, $\text{icon} = 20000$ 又は 23000 となった場合は, epsz の値が適切でない可能性がある. このような場合は, epsz の値を標準値とするか又は変更して再実行してみるとよい.

f) 変数 x_j に符号の制約がない場合の対策

本関数では, $x_j \geq 0, j=1,2,\dots,n$ を原則としている. しかし, x_j に符号の制約がない場合でも以下のように問題を変換することにより本関数を使用することができる.

- x_j を $x_j^+ - x_j^-$ に置き換える.
- 条件 $x_j^+ \geq 0, x_j^- \geq 0$ を付加する.

4. 使用例

変数の個数が 3, 条件式の個数が 5 の線形計画問題を解く.

```
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define ML 2
#define MG 2
#define ME 1
#define M ML+MG+ME
#define N 3

MAIN__()
{
    int ierr, icon;
    double a[M+1][N+1] = {{
        2, 3, -3, 210},
        {-1, -3, 1, -40},
        {2.5, 0, 5, 50},
        {-1.5, -0.5, 1.5, -120},
        {1, 1, 1, 80},
        {-3, -4, 1, -70}};
    double epsz, b[M+1][M+1], vw[2*N+M+ML+MG+1], eps;
    int k, n, imax, isw, kb, ivw[N+ML+MG], nbv[M];
    int m[] = {ML, MG, ME};
    const double minval = 222.5;

    /* initialize data */
    n = N;
    k = N+1;
    kb = M+1;
    isw = 1;
    imax = 20;
    epsz = 0;
    /* minimize */
    ierr = c_dlprs1((double*)a, k, m, n, epsz, &imax, &isw,
        nbv, (double*)b, kb, vw, ivw, &icon);
    printf("icon = %i   imax = %i   isw = %i   obj. fun. value = %f\n",
        icon, imax, isw, b[M][M]);
    /* check result */
    eps = 1e-6;
    if (fabs((b[M][M]-minval)/minval) > eps)
        printf("Inaccurate result\n");
    else
        printf("Result OK\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の LPRS1 の項目及び [26], [125]を参照のこと.

c_dlsbix

実対称バンド行列の連立 1 次方程式(ブロック対角ピボッティング手法)

<pre>ierr = c_dlsbix(a, n, nh, mh, b, epsz, isw, vw, ivw, &icon);</pre>

1. 機能

実係数連立 1 次方程式,

$$\mathbf{Ax} = \mathbf{b}$$

をブロック対角ピボッティング手法 (同名手法にはクラウト法とガウス消去法の関係に似た二つの考え方があり, ここでは後者) で解く. ただし, $\mathbf{A}$ は $n \times n$, バンド幅 h の対称バンド行列, $\mathbf{b}$ は n 次元の実定数ベクトル, $\mathbf{x}$ は n 次元の解ベクトルである. $n > h \geq 0$ であること.

2. 引数

呼出し形式:

```
ierr = c_dlsbix(a, n, nh, mh, b, epsz, isw, vw, ivw, &icon);
```

引数の説明:

a	double a[Alen]	入力	係数行列 $\mathbf{A}$. 対称バンド行列用圧縮格納法. $Alen = n(h+1) - h(h+1)/2$ 格納法の詳細については“1.3.4 バンド行列の格納方法”を参照のこと.
n	int	入力	係数行列 $\mathbf{A}$, 定数ベクトル $\mathbf{b}$ 及び解ベクトル $\mathbf{x}$ の次数 n .
nh	int	入力	$\mathbf{A}$ のバンド幅 h .
		出力	演算後, 内容は保存されない.
mh	int	入力	最大許容バンド幅 h_m ($n > mh \geq nh$). (“使用上の注意” a)参照)
b	double b[n]	入力	定数ベクトル $\mathbf{b}$.
		出力	解ベクトル $\mathbf{x}$.
epsz	double	入力	ピボットの相対零判定値 (≥ 0). 0.0 のときは標準値が採用される. (“使用上の注意” b)参照)
isw	int	入力	制御情報. 同一の係数行列を持つ複数組の方程式を解くとき, 次のとおり指定する. $isw=1$ のとき, 1 組目の方程式を解く. $isw=2$ のとき, 2 組目以降の方程式を解く. このとき $\mathbf{b}$ の値だけを新しい定数ベクトル $\mathbf{b}$ の値に変え, それ以外の引数はそのまま使う. (“使用上の注意” c)参照)
vw	double vw[Vwlen]	作業領域	$Vwlen = n(h_m + 1) - h_m(h_m + 1)/2$
ivw	int ivw[2n]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
20000	ピボットが相対的に零となった. 係数行列は非正則の可能性が強い.	処理を打ち切る.
25000	演算中にバンド幅が許容最大値を超えた.	処理を打ち切る.
30000	次のいずれかであった. • $nh < 0$ • $mh < nh$ • $mh \geq n$ • $epsz < 0$ • $isw \neq 1, 2$	処理を打ち切る.

3. 使用上の注意

a) 引数 **mh**

ピボットリングのための行と列の交換により, 行列のバンド幅は一般に増大する. 許容最大バンド幅 h_m はこの増大分を見越して適当に設定すること. もしも MDM^T 分解の途中にバンド幅が h_m を超過する場合には, $icon = 25000$ として処理が打ち切られる.

b) 引数 **epsz**

ピボットの相対零判定 $epsz$ に 10^{-8} を設定したとすると, この値は次の意味を持っている. すなわち, ブロック対角ピボットリング手法による MDM^T 分解の過程でピボットの値 (1×1 若しくは 2×2 ピボットが作る小行列式)が, 10^{-8} 桁以上の桁落ちを生じた場合に, そのピボットの値を相対的に零と見なし, $icon = 20000$ として処理を打ち切る.

$epsz$ の標準値は丸めの誤差の単位を μ としたとき, 16μ である. なお, ピボットが小さくなっても計算を続行させる場合には, $epsz$ に極小の値を与えればよいが, その結果は保証されない.

c) 引数 **isw**

同一の係数行列を持つ連立 1 次方程式をいくつか続けて解く場合には, 2 回目以降 $isw = 2$ として解くと, 係数行列の MDM^T 分解過程を省略するので計算時間が少なくなる.

d) 領域の節約について

a と vw が同じ領域であっても差支えがないように考慮されているので, 領域を節約するために a と vw を同じ領域にとることができる.

e) 固有値の数及び行列式

行列 A の正, 負固有値の数及び行列式の求め方については本関数が使用する `c_dsbmdm` の使用例を参照すること.

4. 使用例

同一の係数行列を持つ複数組の連立 1 次方程式を解く.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define min(a,b) ((a) < (b) ? (a) : (b))
#define max(a,b) ((a) > (b) ? (a) : (b))

#define NMAX 100
#define NHMAX 50
#define NRHS 2
```

```
MAIN__()
{
    int ierr, icon;
    int n, nh, mh, i, j, ij, jj, isw, jmin, cnt;
    double epsz, eps;
    double a[NMAX*(NHMAX+1)-NHMAX*(NHMAX+1)/2], b[NRHS][NMAX], x[NRHS][NMAX];
    double vw[NMAX*(NHMAX+1)-NHMAX*(NHMAX+1)/2];
    int ivw[2*NMAX];

    /* initialize matrix */
    n = NMAX;
    nh = 2;
    mh = NHMAX;
    ij = 0;
    for (i=0;i<n;i++) {
        jmin = max(i-nh, 0);
        for (j=jmin;j<=i;j++)
            if (i-j == 0)
                a[ij++] = 10;
            else if (i-j == 1)
                a[ij++] = -3;
            else
                a[ij++] = -6;
    }
    /* initialize RHS vectors */
    for (cnt=0;cnt<NRHS;cnt++) {
        for (i=0;i<n;i++)
            x[cnt][i] = (cnt+1)*(i+1);
        /* initialize constant vector b = a*x */
        ierr = c_dmsbv(a, n, nh, &x[cnt][0], &b[cnt][0], &icon);
    }
    isw = 1;
    epsz = 1e-6;
    /* solve systems of equations */
    for (cnt=0;cnt<NRHS;cnt++) {
        ierr = c_dlsbix(a, n, &nh, mh, &b[cnt][0], epsz, isw, vw, ivw, &icon);
        if (icon != 0) {
            printf("ERROR: c_dlsbix failed with icon = %d\n", icon);
            exit(1);
        }
        /* check solution vector */
        eps = 1e-6;
        for (i=0;i<n;i++)
            if (fabs((x[cnt][i]-b[cnt][i])/b[cnt][i]) > eps) {
                printf("WARNING: result inaccurate\n");
                exit(1);
            }
        printf("Result OK\n");
        if (cnt == 0) isw = 2;
    }
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”のLSBIXの項目及び[15]を参照のこと。

c_dlsix

実対称行列の連立 1 次方程式 (ブロック対角ピボッティング手法)

<pre>ierr = c_dlsix(a, n, b, epsz, isw, vw, ip, ivw, &icon);</pre>

1. 機能

実係数連立 1 次方程式(1)

$$\mathbf{Ax} = \mathbf{b} \quad (1)$$

をブロック対角ピボッティング手法 (同名手法にはクラウト法とガウス消去法の関係に似た二つの考え方があり, ここでは前者) で解く. ただし, $\mathbf{A}$ は $n \times n$ 対称行列, $\mathbf{b}$ は n 次元の実定数ベクトル, $\mathbf{x}$ は n 次元の解ベクトルである. $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dlsix(a, n, b, epsz, isw, vw, ip, ivw, &icon);
```

引数の説明:

a	double a[Alen]	入力	係数行列 $\mathbf{A}$. 対称行列用圧縮格納法. $Alen = n(n+1)/2$ 格納法の詳細については“1.3.2 対称行列の格納方法”を参照のこと.
n	int	出力	演算後, 内容は保存されない.
b	double b[n]	入力	係数行列 $\mathbf{A}$, 定数ベクトル $\mathbf{b}$ 及び解ベクトル $\mathbf{x}$ の次数 n .
		出力	定数ベクトル $\mathbf{b}$.
epsz	double	入力	解ベクトル $\mathbf{x}$.
		入力	ピボットの相対零判定値 (≥ 0). 0.0 のときは標準値が採用される. (“使用上の注意” a)参照)
isw	int	入力	制御情報. 同一の係数行列を持つ複数組の方程式を解くとき, 次のとおり指定する. $isw=1$ のとき, 1 組目の方程式を解く. $isw=2$ のとき, 2 組目以降の方程式を解く. ただし, このとき b の値だけを新しい定数ベクトル $\mathbf{b}$ の値に変え, それ以外の引数はそのまま使う. (“使用上の注意” b)参照)
vw	double vw[2n]	作業領域	
ip	int ip[n]	作業領域	
ivw	int ivw[n]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	ピボットが相対的に零となった. 係数行列は非正則の可能性が強い.	処理を打ち切る.
30000	次のいずれかであった. • $n < 1$ • $\text{epsz} < 0$ • $\text{isw} \neq 1, 2$	処理を打ち切る.

3. 使用上の注意

a) 引数 **epsz**

ピボットの相対零判定 epsz に 10^{-8} を設定したとすると, この値は次の意味を持っている. すなわち, ブロック対角ピボッティング手法による MDM^T 分解の過程でピボットの値 (1×1 若しくは 2×2 ピボットが作る小行列式)が, 10 進 s 桁以上の桁落ちを生じた場合に, そのピボットの値を相対的に零と見なし, $\text{icon} = 20000$ として処理を打ち切る.

epsz の標準値は丸めの誤差の単位を μ としたとき, 16μ である. なお, ピボットが小さくなっても計算を続行させる場合には, epsz に極小の値を与えればよいが, その結果は保証されない.

b) 引数 **isw**

同一の係数行列を持つ連立 1 次方程式をいくつか続けて解く場合には, 2 回目以降 $\text{isw} = 2$ として解くと, 係数行列の MDM^T 分解過程を省略するので計算時間が少なくなる.

c) 固有値の数及び行列式

行列 **A** の正, 負固有値の数及び行列式の求め方については本関数が使用する `c_dsmdm` の使用例を参照すること.

4. 使用例

同一の係数行列を持つ複数組の連立 1 次方程式を求める.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 100
#define NRHS 2

MAIN__()
{
    int ierr, icon;
    int n, i, j, ij, isw, cnt;
    double epsz, eps, pi, an, ar;
    double a[NMAX*(NMAX+1)/2], b[NRHS][NMAX], x[NRHS][NMAX], vw[2*NMAX];
    int ip[NMAX], ivw[NMAX];

    /* initialize matrix */
    n = NMAX;
    ij = 0;
    pi = 2*asin(1);
    an = 1.0/(n+1);
    ar = pi*an;
    an = sqrt(2*an);
    for (i=1; i<=n; i++)
        for (j=1; j<=i; j++) {
            a[ij++] = an*sin(i*j*ar);
        }
    isw = 1;
```



```
    epsz = 1e-6;
    /* initialize RHS vectors */
    for (cnt=0;cnt<NRHS;cnt++) {
        for (i=0;i<n;i++)
            x[cnt][i] = (cnt+1)*(i+1);
        /* initialize constant vector b = a*x */
        ierr = c_dmsv(a, n, &x[cnt][0], &b[cnt][0], &icon);
    }
    /* solve systems of equations */
    for (cnt=0;cnt<NRHS;cnt++) {
        ierr = c_dlsix(a, n, &b[cnt][0], epsz, isw, vw, ip, ivw, &icon);
        if (icon != 0) {
            printf("ERROR: c_dlsix failed with icon = %d\n", icon);
            exit(1);
        }
        /* check solution vector */
        eps = 1e-6;
        for (i=0;i<n;i++)
            if (fabs((x[cnt][i]-b[cnt][i])/b[cnt][i]) > eps) {
                printf("WARNING: result inaccurate\n");
                exit(1);
            }
        printf("Result OK\n");
        if (cnt == 0) isw = 2;
    }
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”のLSIXの項目及び[15]を参照のこと。

c_dlstx

正値対称 3 項行列の連立 1 次方程式(変形コレスキー法)

<pre>ierr = c_dlstx(d, sd, n, b, epsz, isw, &icon);</pre>

1. 機能

実係数連立 1 次方程式,

$$\mathbf{Ax} = \mathbf{b}$$

を変形コレスキー法で解く. ただし, $\mathbf{A}$ は $n \times n$ の正値対称な実 3 項行列, $\mathbf{b}$ は n 次元の実定数ベクトル, $\mathbf{x}$ は n 次元の解ベクトルである. また, $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dlstx(d, sd, n, b, epsz, isw, &icon);
```

引数の説明:

d	double d[n]	入力	係数行列 $\mathbf{A}$ の対角部分. 演算後, 内容は保存されない.
sd	double sd[n-1]	入力	係数行列 $\mathbf{A}$ の副対角部分. 演算後, 内容は保存されない.
n	int	入力	係数行列 $\mathbf{A}$ の次数 n .
b	double b[n]	入力	定数ベクトル $\mathbf{b}$.
		出力	解ベクトル $\mathbf{x}$.
epsz	double	入力	ピボットの相対零判定値 (≥ 0.0). 0.0 のときは標準値が採用される. (“使用上の注意” a)参照).
isw	int	入力	制御情報. 同一の係数行列を持つ $l (\geq 1)$ 組の方程式を解くとき, 次のとおり指定する. $isw=1$ のとき, 1 組目の方程式を解く. $isw=2$ のとき, 2 組目以降の方程式を解く. ただし, このとき $\mathbf{b}$ の値だけを新しい定数ベクトル $\mathbf{b}$ の値に変え, それ以外の引数 はそのまま使う. (“使用上の注意” b)参照).
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
10000	ピボットが負となった. 係数行列は正値でない.	処理は続行する.
20000	ピボットが相対的に零となった. 係数行列は非正則の可能性が強い.	処理を打ち切る.
30000	次のいずれかであった. $n < 1$ $epsz < 0$ $isw \neq 1, 2$	処理を打ち切る.

3. 使用上の注意

a) epsz について

ピボットの相対零判定値 epsz に 10^{-5} を設定したとすると、この値は次の意味を持っている。すなわち、変形コレスキー法による LDL^T 分解の過程でピボットの値が 10 進 s 桁以上の桁落ちを生じた場合にそのピボットを相対的に零とみなし、 $\text{icon} = 20000$ として処理を打ち切る。

epsz の標準値は丸め誤差の単位を μ としたとき、 $\text{epsz} = 16\mu$ である。

なお、ピボットが小さくなっても計算を続行させる場合には、 epsz へ極小の値を与えればよいが、その結果は保証されない。

b) isw について

同一の係数行列を持つ連立一次方程式をいくつか続けて解く場合には、2 回目以降 $\text{isw} = 2$ として解くと、係数行列 $\mathbf{A}$ の LDL^T 分解過程を省略するので計算時間が少なくなる。

c) 行列式

行列 $\mathbf{A}$ の行列式は、 n 個の配列要素 $d[i], i = 0, \dots, n-1$ を掛け合わせて得られる。

d) ピボットが負になった場合

分解過程でピボットが負になった場合は、ピボットティングを行っていないので計算誤差は大きい可能性があるから注意すること。

4. 使用例

行列 $\mathbf{A}$ とベクトル $\mathbf{x}$ の積 $\mathbf{b}$ を求め、 $\mathbf{Ax} = \mathbf{b}$ を解く。得られた解と $\mathbf{x}$ を比較して結果を確認する。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 100

MAIN__()
{
    int ierr, icon;
    int n, i, isw;
    double epsz, eps;
    double d[NMAX], sd[NMAX-1], b[NMAX], x[NMAX];

    /* initialize matrix and vector */
    n = NMAX;
    for (i=0; i<n-1; i++) {
        sd[i] = -1;
        d[i] = 10;
    }
    d[n-1] = 10;
    for (i=0; i<n; i++)
        x[i] = i+1;
    /* initialize constant vector b */
    b[0] = d[0]*x[0] + sd[0]*x[1];
    for (i=1; i<n-1; i++) {
        b[i] = sd[i-1]*x[i-1] + d[i]*x[i] + sd[i]*x[i+1];
    }
    b[n-1] = sd[n-2]*x[n-2] + d[n-1]*x[n-1];
    epsz = 1e-6;
    isw = 1;
    /* solve system of equations */
    ierr = c_dlstx(d, sd, n, b, epsz, isw, &icon);
    if (icon > 10000) {
        printf("ERROR: c_dlstx failed with icon = %d\n", icon);
        exit(1);
    }
}
```

```
    }  
    /* check solution vector */  
    eps = 1e-6;  
    for (i=0;i<n;i++)  
        if (fabs((x[i]-b[i])/b[i]) > eps) {  
            printf("WARNING: result inaccurate\n");  
            exit(1);  
        }  
    printf("Result OK\n");  
    return(0);  
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の LSTX の項目を参照のこと.

c_dltx

実3項行列の連立1次方程式(ガウス消去法)

```
ierr = c_dltx(sbd, d, spd, n, b, epsz, isw,
              &is, ip, vw, &icon);
```

1. 機能

実係数連立1次方程式をガウス消去法で解く。

$$\mathbf{Ax} = \mathbf{b}$$

ただし、 $\mathbf{A}$ は $n \times n$ の正則な実3項行列、 $\mathbf{b}$ は n 次元の実定数ベクトル、 $\mathbf{x}$ は n 次元の解ベクトルである。また、 $n \geq 1$ であること。

2. 引数

呼出し形式:

```
ierr = c_dltx(sbd, d, spd, n, b, epsz, isw, &is, ip, vw, &icon);
```

引数の説明:

sbd	double sbd[n-1]	入力	係数行列 $\mathbf{A}$ の下副対角部分. 演算後, 内容は保存されない.
d	double d[n]	入力	係数行列 $\mathbf{A}$ の対角部分. 演算後, 内容は保存されない.
spd	double spd[n-1]	入力	係数行列 $\mathbf{A}$ の上副対角部分. 演算後, 内容は保存されない.
n	int	入力	係数行列 $\mathbf{A}$ の次数 n .
b	double b[n]	入力	定数ベクトル $\mathbf{b}$.
		出力	解ベクトル $\mathbf{x}$.
epsz	double	入力	ピボットの相対零判定値 (≥ 0.0). 0.0 のときは標準値が採用される. (“使用上の注意” a)参照).
isw	int	入力	制御情報. 同一の係数行列を持つ $l (\geq 1)$ 組の方程式を解くとき, 次のとおり指定する. $isw = 1$ のとき, 1 組目の方程式を解く. $isw = 2$ のとき, 2 組目以降の方程式を解く. ただし, このとき b の値だけを新しい定数ベクトル $\mathbf{b}$ の値に変え, それ以外の引数はそのまま使う. (“使用上の注意” b)参照).
is	int	出力	行列 $\mathbf{A}$ の行列式を求めるための情報. (“使用上の注意” c)参照).
ip	int ip[n]	作業領域	
vw	double vw[n]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	ピボットが相対的に零となった. 係数行列は非正則の可能性が高い.	処理を打ち切る.

コード	意味	処理内容
30000	次のいずれかであった。 • $n < 1$ • $\text{epsz} < 0$ • $\text{isw} \neq 1, 2$	処理を打ち切る。

3. 使用上の注意

a) **epsz** について

本関数では、ピボットの相対零判定の際、**epsz** の値を含んだ関係式にて行う。

epsz の標準値は丸め誤差の単位を μ としたとき、 $\text{epsz} = 16\mu$ である。

なお、ピボットが小さくなくても計算を続行させる場合には、**epsz** へ極小の値を与えればよいが、その結果は保証されない。

b) **isw** について

同一の係数行列を持つ連立一次方程式をいくつか続けて解く場合には、2 回目以降 $\text{isw} = 2$ として解くと、係数行列 **A** の LU 分解過程を省略するので計算時間が少なくなる。なお、この場合 **is** の値は、 $\text{isw} = 1$ のときの値が保障される。

c) 行列式

行列 **A** の行列式は、 n 個の配列要素 $d[i], i = 0, \dots, n-1$ の積と **is** の値を掛け合わせて得られる。

4. 使用例

行列 **A** とベクトル **x** の積 **b** を求め、 $\mathbf{Ax} = \mathbf{b}$ を解く。得られた解と **x** を比較して結果を確認する。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 100

MAIN__()
{
    int ierr, icon;
    int n, i, isw, is;
    double epsz, eps;
    double sbd[NMAX-1], d[NMAX], spd[NMAX-1], b[NMAX], x[NMAX], vw[NMAX];
    int ip[NMAX];

    /* initialize matrix and vector */
    n = NMAX;
    for (i=0; i<n-1; i++) {
        sbd[i] = -1;
        spd[i] = -1;
        d[i] = 10;
    }
    d[n-1] = 10;
    for (i=0; i<n; i++)
        x[i] = i+1;
    /* initialize constant vector b */
    b[0] = d[0]*x[0] + spd[0]*x[1];
    for (i=1; i<n-1; i++) {
        b[i] = sbd[i-1]*x[i-1] + d[i]*x[i] + spd[i]*x[i+1];
    }
    b[n-1] = sbd[n-2]*x[n-2] + d[n-1]*x[n-1];
    epsz = 1.0e-6;
    isw = 1;
    /* solve system of equations */
    ierr = c_dltx(sbd, d, spd, n, b, epsz, isw, &is, ip, vw, &icon);
```

```
if (icon != 0) {
    printf("ERROR: c_dltx failed with icon = %d\n", icon);
    exit(1);
}
/* check solution vector */
eps = 1e-6;
for (i=0;i<n;i++)
    if (fabs((x[i]-b[i])/b[i]) > eps) {
        printf("WARNING: result inaccurate\n");
        exit(1);
    }
printf("Result OK\n");
return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の LTX の項目を参照のこと.

c_dlux

LU 分解された実行列の連立 1 次方程式

ierr = c_dlux(b, fa, k, n, isw, ip, &icon);

1. 機能

LU 分解された実係数行列を持つ連立 1 次方程式

$$\mathbf{LUx} = \mathbf{Pb} \quad (1)$$

を解く。ただし、 $\mathbf{L}$, $\mathbf{U}$ はそれぞれ $n \times n$ の下三角行列と単位上三角行列、 $\mathbf{P}$ は置換行列(係数行列を LU 分解するときの部分ピボッティングによる行の入換えを行う)、 $\mathbf{b}$ は n 次元の実定数ベクトル、 $\mathbf{x}$ は n 次元の解ベクトルである。なお、方程式 (1) のかわりに

$$\mathbf{Ly} = \mathbf{Pb} \quad (2)$$

$$\mathbf{Uz} = \mathbf{b} \quad (3)$$

のいずれかを解くこともできる。 $n \geq 1$ であること。

2. 引数

呼出し形式:

```
ierr = c_dlux(b, (double*)fa, k, n, isw, ip, &icon);
```

引数の説明:

b	double b[n]	入力	定数ベクトル $\mathbf{b}$.
		出力	解ベクトル $\mathbf{x}, \mathbf{y}, \mathbf{z}$ のうちいずれか.
fa	double fa[n][k]	入力	行列 $\mathbf{L} + (\mathbf{U} - \mathbf{I})$. (“使用上の注意”参照)
k	int	入力	配列 fa の整合寸法($\geq n$).
n	int	入力	行列 $\mathbf{L}, \mathbf{U}$ の次数 n .
isw	int	入力	制御情報. • isw=1 のとき, (1) の $\mathbf{x}$ を求める. • isw=2 のとき, (2) の $\mathbf{y}$ を求める. • isw=3 のとき, (3) の $\mathbf{z}$ を求める.
ip	int ip[n]	入力	部分ピボッティングによる行の入換えの履歴を示すトランスポジションベクトル. (“使用上の注意”参照)
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	係数行列が非正則であった.	処理を打ち切る.
30000	次のいずれかであった. • $n < 1$ 又は $k < n$ • $isw \neq 1, 2, 3$ • ip に誤りがあった	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

連立 1 次方程式を解く場合、関数 `c_dvalu` を呼び出して、係数行列を LU 分解させてから、本関数を呼び出せば方程式を解くことができる。このとき入力引数 `fa` と `ip` は `c_dvalu` の `a` と `ip` にそれぞれ対応している。しかし、通常は関数 `c_dvlax` を呼び出せば、一度に解が求められる。

4. 使用例

係数行列を LU 分解してから、連立 1 次方程式を解く。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 100

MAIN__()
{
    int ierr, icon;
    int n, i, j, k, is, isw;
    double epsz, eps;
    double fa[NMAX][NMAX];
    double b[NMAX], x[NMAX], vw[NMAX];
    int ip[NMAX];

    /* initialize matrix and vector */
    n = NMAX;
    k = NMAX;
    for (i=0;i<n;i++) {
        for (j=i;j<n;j++) {
            fa[i][j] = n-j;
            fa[j][i] = n-j;
        }
        x[i] = i+1;
    }
    /* initialize constant vector zb = za*zx */
    ierr = c_dmav((double*)fa, k, n, n, x, b, &icon);
    epsz = 1e-6;
    /* perform LU decomposition */
    ierr = c_dvalu((double*)fa, k, n, epsz, ip, &is, vw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dlu failed with icon = %d\n", icon);
        exit(1);
    }
    isw = 1;
    /* solve system of equations using LU factors */
    ierr = c_dlux(b, (double*)fa, k, n, isw, ip, &icon);
    if (icon != 0) {
        printf("ERROR: c_dlux failed with icon = %d\n", icon);
        exit(1);
    }
    /* check result */
    eps = 1e-6;
    for (i=0;i<n;i++)
        if (fabs((b[i]-x[i])/x[i]) > eps) {
            printf("WARNING: result inaccurate\n");
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の LUX の項目及び[7], [34], [83]を参照のこと。

c_dmv

実行列と実ベクトルの積

<code>ierr = c_dmv(a, k, m, n, x, y, &icon);</code>

1. 機能

$m \times n$ の実行列 **A** と実ベクトル **x** の積 **y** を求める.

$$\mathbf{y} = \mathbf{Ax} \quad (1)$$

ここで **x** は n 次元ベクトル, **y** は m 次元ベクトルである. $m, n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dmv((double*)a, k, m, n, x, y, &icon);
```

引数の説明:

a	double a[m][k]	入力	行列 A .
k	int	入力	配列 a の整合寸法 ($\geq n$).
m	int	入力	行列 A の行数 m .
n	int	入力	行列 A の列数 n (“使用上の注意” 参照).
x	double x[n]	入力	ベクトル x .
y	double y[m]	入力	ベクトル y ’.
		出力	“使用上の注意” (2) の計算を行うときにのみ使用する.
		出力	行列 A とベクトル x の積 y .
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	次のいずれかであった. $m < 1$ $n = 0$ $k < n$	処理を打ち切る.

3. 使用上の注意

a) 残差ベクトルの計算

本関数は, 通常(1)の計算を主とするが, $n = -n$ として任意のベクトル **y**’ を引数 **y** に与えると,

$$\mathbf{y} = \mathbf{y}' - \mathbf{Ax} \quad (2)$$

なる計算ができる.

具体的には, 連立 1 次方程式の残差ベクトル,

$$\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}$$

の計算等を行う場合に利用できる。

4. 使用例

行列とベクトルの積を計算する。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 100

MAIN__()
{
    int ierr, icon;
    int m, n, i, j, k;
    double eps;
    double a[NMAX][NMAX], x[NMAX], y[NMAX];

    /* initialize matrix and vector */
    m = NMAX;
    n = NMAX;
    k = NMAX;
    for (i=0;i<n;i++) {
        for (j=0;j<n;j++)
            a[i][j] = 1.0/(j+1);
        x[i] = i+1;
    }
    /* perform matrix vector multiply */
    ierr = c_dmav((double*)a, k, m, n, x, y, &icon);
    if (icon != 0) {
        printf("ERROR: c_dmav failed with icon = %d\n", icon);
        exit(1);
    }
    /* check vector */
    eps = 1e-6;
    for (i=0;i<n;i++)
        if (fabs((y[i]-n)/n) > eps) {
            printf("WARNING: result inaccurate\n");
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}
```

5. 手法概要

本関数は標準的なアルゴリズムを用いている。

この方法の原理及び、本関数の手順については“富士通 SSL II 使用手引書”の MAV の項目の手法概要を参照のこと。

c_dmcv

複素行列と複素ベクトルの積

ierr = c_dmcv(za, k, m, n, zx, zy, &icon);

1. 機能

$m \times n$ の複素行列 $\mathbf{A}$ と複素ベクトル $\mathbf{x}$ の積 $\mathbf{y}$ を求める.

$$\mathbf{y} = \mathbf{Ax} \quad (1)$$

ここで $\mathbf{x}$ は n 次元の複素ベクトル, $\mathbf{y}$ は m 次元の複素ベクトルである. $m, n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dmcv((dcomplex*)za, k, m, n, zx, zy, &icon);
```

引数の説明:

za	dcomplex za[m][k]	入力	複素行列 $\mathbf{A}$.
k	int	入力	配列 za の整合寸法 ($\geq n$).
m	int	入力	行列 $\mathbf{A}$ の行数 m .
n	int	入力	行列 $\mathbf{A}$ の列数 n (“使用上の注意” a)参照).
zx	dcomplex zx[n]	入力	複素ベクトル $\mathbf{x}$.
zy	dcomplex zy[m]	入力	複素ベクトル $\mathbf{y}'$. “使用上の注意” a) (2)の計算を行うときにのみ使用する.
		出力	行列 $\mathbf{A}$ と複素ベクトル $\mathbf{x}$ の積 $\mathbf{y}$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	次のいずれかであった. $m < 1$ $n = 0$ $k < n$	処理を打ち切る.

3. 使用上の注意

a) 残差ベクトルの計算

本関数は, 通常(1)の計算を主とするが, $n = -n$ として任意の複素ベクトル $\mathbf{y}'$ を引数 zy に与えると

$$\mathbf{y} = \mathbf{y}' - \mathbf{Ax} \quad (2)$$

なる計算ができる.

具体的には, 連立 1 次方程式の残差ベクトル

$$\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}$$

の計算等を行う場合に利用できる.

4. 使用例

複素行列と複素ベクトルの積を計算する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 100

MAIN__()
{
    int ierr, icon;
    int m, n, i, j, k;
    double eps;
    dcomplex za[NMAX][NMAX], zx[NMAX], zy[NMAX], sum;

    /* initialize matrix and vector */
    m = NMAX;
    n = NMAX;
    k = NMAX;
    for (i=0;i<n;i++) {
        for (j=0;j<n;j++) {
            za[i][j].re = 1.0/(j+1);
            za[i][j].im = 1.0/(j+1);
        }
        zx[i].re = i+1;
        zx[i].im = i+1;
    }
    /* perform complex matrix vector multiply */
    ierr = c_dmcv((dcomplex*)za, k, m, n, zx, zy, &icon);
    if (icon != 0) {
        printf("ERROR: c_dmcv failed with icon = %d\n", icon);
        exit(1);
    }
    /* check vector */
    eps = 1e-6;
    for (i=0;i<n;i++) {
        sum.re = 0;
        sum.im = 0;
        for (j=0;j<n;j++) {
            sum.re = sum.re + za[i][j].re*zx[j].re - za[i][j].im*zx[j].im;
            sum.im = sum.im + za[i][j].im*zx[j].re + za[i][j].re*zx[j].im;
        }
        if (fabs((zy[i].re-sum.re)) > eps ||
            fabs((zy[i].im-sum.im)) > eps) {
            printf("WARNING: result inaccurate\n");
            exit(1);
        }
    }
    printf("Result OK\n");
    return(0);
}
```

5. 手法概要

本関数は標準的なアルゴリズムを用いている.

この方法の原理及び, 本関数の手順については“富士通 SSL II 使用手引書”の MCV の項目の手法概要を参照のこと.

c_dmdmx

MDM ^T 分解された実対称行列の連立 1 次方程式

ierr = c_dmdmx(b, fa, n, ip, &icon);

1. 機能

MDM^T 分解された実対称係数行列を持つ連立 1 次方程式,

$$\mathbf{P}^{-1}\mathbf{MDM}^T\mathbf{P}^{-T}\mathbf{x}=\mathbf{b}$$

を解く。ただし、 $\mathbf{M}$ は単位下三角行列、 $\mathbf{D}$ は高々次数 2 の対称なブロックから成る対称ブロック対角行列、 $\mathbf{P}$ は置換行列 (係数行列を MDM^T 分解するときのピボットティングによる行の入換えを行う)、 $\mathbf{b}$ は n 次元の実定数ベクトル、 $\mathbf{x}$ は n 次元の解ベクトルである。ここで $d_{k+1,k} \neq 0$, なら $m_{k+1,k} = 0$ である。 $n \geq 1$ であること。

2. 引数

呼出し形式:

```
ierr = c_dmdmx(b, fa, n, ip, &icon);
```

引数の説明:

b	double b[n]	入力	定数ベクトル $\mathbf{b}$.
		出力	解ベクトル $\mathbf{x}$.
fa	double fa[Falen]	入力	行列 $\mathbf{D} + (\mathbf{M} - \mathbf{I})$. 対称行列用圧縮格納法. $Falen = n(n+1)/2$ 格納法の詳細については“1.3.2 対称行列の格納方法”を参照のこと.
n	int	入力	行列 $\mathbf{M}$, 行列 $\mathbf{D}$, 定数ベクトル $\mathbf{b}$ 及び解ベクトルの $\mathbf{x}$ の次数 n .
ip	int ip[n]	入力	ピボットティングによる行の入換えの履歴を示すトランスポジションベクトル. (“使用上の注意” 参照)
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	係数行列が非正則であった.	処理を打ち切る.
30000	次のいずれかであった. • $n < 1$ • ip に誤りがあった.	処理を打ち切る.

3. 使用上の注意

a) 引数 fa, ip 及び MDM^T 分解

連立 1 次方程式を解く場合、関数 c_dsmdm を呼び出して、係数行列を MDM^T 分解させてから、本関数を呼び出せば方程式を解くことができる。しかし、通常は関数 c_dlsix を呼び出せば、一度に解が求められる。

本関数の入力引数 fa, ip は、関数 c_dsmdm の出力引数 a, ip と同じである。

4. 使用例

実対称行列を MDM^T 分解させてから、連立 1 次方程式を解く。

```

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 100

MAIN__()
{
    int ierr, icon;
    int n, i, j, ij;
    double epsz, eps, pi, an, ar;
    double a[NMAX*(NMAX+1)/2], b[NMAX], x[NMAX], vw[2*NMAX];
    int ip[NMAX], ivw[NMAX];

    /* initialize matrix */
    n = NMAX;
    ij = 0;
    pi = 2*asin(1);
    an = 1.0/(n+1);
    ar = pi*an;
    an = sqrt(2*an);
    for (i=1;i<=n;i++)
        for (j=1;j<=i;j++) {
            a[ij++] = an*sin(i*j*ar);
        }
    epsz = 1e-6;
    /* initialize RHS vector */
    for (i=0;i<n;i++)
        x[i] = i+1;
    /* initialize constant vector b = a*x */
    ierr = c_dmsv(a, n, x, b, &icon);
    /* MDM decomposition of system */
    ierr = c_dsmdm(a, n, epsz, ip, vw, ivw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dsmdm failed with icon = %d\n", icon);
        exit(1);
    }
    /* solve decomposed system of equations */
    ierr = c_dmdmx(b, a, n, ip, &icon);
    if (icon != 0) {
        printf("ERROR: c_dmdmx failed with icon = %d\n", icon);
        exit(1);
    }
    /* check solution vector */
    eps = 1e-6;
    for (i=0;i<n;i++)
        if (fabs((x[i]-b[i])/b[i]) > eps) {
            printf("WARNING: result inaccurate\n");
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}

```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”のMDMXの項目を参照のこと。

c_dmgs

行列の積 (実行列・実対称行列)

<code>ierr = c_dmgs(a, ka, b, c, kc, n, vw, &icon);</code>

1. 機能

$n \times n$ の実行列 **A** と, $n \times n$ の実対称行列 **B** の積 **C** を求める.

$$\mathbf{C} = \mathbf{AB}$$

ここで **C** は, $n \times n$ の実行列である. $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dmgs((double*)a, ka, b, (double*)c, kc, n, vw, &icon);
```

引数の説明:

a	double a[n][ka]	入力	行列 A .
ka	int	入力	配列 a の整合寸法 ($\geq n$).
b	double b[Blen]	入力	行列 B . 対称行列用圧縮格納法. $Blen = n(n+1)/2$ 格納法の詳細については“1.3.2 対称行列の格納方法”を参照のこと.
c	double c[n][kc]	出力	行列 C . (“使用上の注意”参照)
kc	int	入力	配列 c の整合寸法 ($\geq n$).
n	int	入力	行列 A , B 及び C の行数 n .
vw	double vw[n]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	次のいずれかであった. $n < 1$ $ka < n$ $kc < n$	処理を打ち切る.

3. 使用上の注意

a) 領域の節約について

配列 a 上の内容を保存する必要のない場合は, 次のように呼び出すことにより領域が節約できる.

```
ierr = c_dmgs(a, ka, b, a, ka, n, vw, &icon);
```

この場合, 配列 a に, 一般行列の格納方法で格納された行列 **C** が得られる.

4. 使用例

実行列 **A** と実対称行列 **B** の積 **C** を求める。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 5

/* print symmetric matrix */
void prtsymmat(double a[], int n)
{
    int ij, i, j;
    printf("symmetric matrix format\n");
    ij = 0;
    for (i=0;i<n;i++) {
        for (j=0;j<=i;j++)
            printf("%7.2f ",a[ij++]);
        printf("\n");
    }
}

/* print general matrix */
void prtgenmat(double *a, int k, int n, int m)
{
    int i, j;
    printf("general matrix format\n");
    for (i=0;i<n;i++) {
        for (j=0;j<m;j++)
            printf("%7.2f ",a[i*k+j]);
        printf("\n");
    }
}

MAIN__()
{
    int ierr, icon;
    int n, i, j, ij, ka, kc;
    double a[NMAX][NMAX], b[NMAX*(NMAX+1)/2], c[NMAX][NMAX], vw[NMAX];

    n = NMAX;
    /* initialize symmetric matrix */
    ij = 0;
    for (i=0;i<n;i++)
        for (j=0;j<=i;j++) {
            b[ij++] = i-j+1;
        }
    /* initialize general matrix */
    for (i=0;i<n;i++)
        for (j=0;j<n;j++) {
            a[i][j] = j+1;
        }
    ka = NMAX;
    kc = NMAX;
    /* matrix multiply */
    ierr = c_dmgs((double*)a, ka, b, (double*)c, kc, n, vw, &icon);
    /* print matrices */
    printf("a: \n");
    prtgenmat((double*)a, ka, n, n);
    printf("b: \n");
    prtsymmat(b, n);
    printf("c: \n");
    prtgenmat((double*)c, kc, n, n);
    return(0);
}
```

c_dminf1

多変数関数の極小化 (微係数不要, 改訂準ニュートン法)

<pre>ierr = c_dminf1(x, n, fun, epsr, &max, &f, g, h, vw, &icon);</pre>

1. 機能

n 変数の実関数 $f(\mathbf{x})$ と, 初期ベクトル $\mathbf{x}_0$ が与えられたとき, 改訂準ニュートン法により, $f(\mathbf{x})$ の極小値を与えるベクトル $\mathbf{x}^*$ と, その関数値 $f(\mathbf{x}^*)$ を求める.

ただし, $f(\mathbf{x})$ は 2 階までの連続な偏導関数を持つものと仮定する. また, $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dminf1(x, n, fun, epsr, &max, &f, g, h, vw, &icon);
```

引数の説明:

x	double x[n]	入力	初期ベクトル $\mathbf{x}_0$.
		出力	ベクトル $\mathbf{x}^*$.
n	int	入力	変数の個数 n .
fun	function	入力	$f(\mathbf{x})$ を計算するユーザ関数名. プロトタイプ宣言: double fun(double x[]); 引数の説明: x double 入力 任意な変数ベクトル $\mathbf{x}$. x[n]
epsr	double	入力	収束判定値. epsr = 0 のときは標準値が採用される. (使用上の注意 a)参照)
max	int	入力	関数 $f(\mathbf{x})$ の評価回数の上限. (使用上の注意 b)参照)
		出力	実際に評価した回数.
f	double	出力	関数値 $f(\mathbf{x}^*)$.
g	double g[n]	出力	$\mathbf{x}^*$ における傾斜ベクトル.
h	double h[Hlen]	出力	$\mathbf{x}^*$ におけるヘシアン行列. $Hlen = n(n+1)/2$. icon = 10000 のときに再度本関数を呼ぶ場合, この配列の内容を変更してはならない. (使用上の注意 b)参照)
vw	double vw[3n+1]	作業領域	icon = 10000 のときに再度本関数を呼ぶ場合, この配列の内容を変更してはならない. (使用上の注意 b)参照)
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード：

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	指定された関数の評価回数内で, 収束条件が満たされなかった.	処理を打ち切る. 引数 x, f, g, h には最後の値を格納する.
20000	計算の過程で, 関数の局所的減少が満たされなかった. epsr が小さすぎるか, 傾斜ベクトルの差分近似の誤差が計算限界を超えた.	処理を打ち切る. 引数 x, f には, 最後の値を格納する.
30000	次のいずれかであった. $n < 1$ $\text{epsr} < 0$ $\text{max} = 0$	処理を打ち切る.

3. 使用上の注意

a) 引数 **epsr** の与え方

本関数の収束判定は, 反復ベクトル $\mathbf{x}_k$ において,

$$\|\mathbf{x}_{k+1} - \mathbf{x}_k\|_{\infty} \leq \max(1, \|\mathbf{x}_k\|_{\infty}) \cdot \text{epsr}$$

となった場合に, $\mathbf{x}_{k+1}$ を極小点 $\mathbf{x}^*$ とみなし反復を終了する.

本関数では, 関数 $f(\mathbf{x})$ が極小点 $\mathbf{x}^*$ の近傍で近似的に 2 次関数であることを前提としており, 関数値 $f(\mathbf{x}^*)$ を丸め誤差程度まで正しく求めるためには,

$$\text{epsr} \approx \sqrt{\mu}, \quad \mu \text{ は丸め誤差の単位}$$

程度に与えるのが適当である.

epsr の標準値は $2 \cdot \sqrt{\mu}$ である.

b) 引数 **max** の与え方

関数の評価回数は, 変数ベクトル $\mathbf{x}$ に対して, $f(\mathbf{x})$ を計算することを 1 回と数える. すなわち, ユーザ関数 `fun` の呼び出し回数に一致する.

この評価回数は, 関数 $f(\mathbf{x})$ の性質, 初期ベクトル, 収束判定値に依存する.

一般には, 初期ベクトルが良く, 収束判定値として標準値を採用した場合には, $\text{max} = 400 \cdot n$ 程度が妥当である.

指定評価回数内で収束条件が満たされず, $\text{icon} = 10000$ として戻った場合でも, 再度本関数を呼び出すことにより, 継続して反復することができる. しかし, この場合は引数 max には, 負の値で追加評価回数を指定し, 他の引数の内容は保存したままで呼び出すこと.

4. 使用例

関数 $f(\mathbf{x}) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$ の最小点を初期ベクトル $\mathbf{x}_0 = (-1.2, 1.0)^T$ により求める.

```
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define N 2

double fun(double x[]); /* user function prototype */
```

```
MAIN__()
{
    int ierr, icon;
    double f, x[N], g[N], h[N*(N+1)/2], vw[3*N+1], epsr, eps, exact;
    int max, n;

    /* initialize data */
    x[0] = -1.2;
    x[1] = 1;
    n = N;
    epsr = 1e-3;
    max = 400*n;
    /* find minimum of function */
    ierr = c_dminf1(x, n, fun, epsr, &max, &f, g, h, vw, &icon);
    printf("icon = %i    max = %i    x = (%12.4e, %12.4e)    f = %12.4e\n",
           icon, max, x[0], x[1], f);
    /* check result */
    exact = 0;
    eps = 1e-6;
    if (fabs(f-exact) > eps)
        printf("Inaccurate result\n");
    else
        printf("Result OK\n");
    return(0);
}

/* user function */
double fun(double x[])
{
    return(pow(1-x[0],2)+100*pow((x[1]-x[0]*x[0]),2));
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の MINF1 の項目及び[33]を参照のこと。

c_dming1

多変数関数の極小化 (微係数要, 準ニュートン法)

```
ierr = c_dming1(x, n, fun, grad, epsr, &max,
                &f, g, h, vw, &icon);
```

1. 機能

n 変数の実関数 $f(\mathbf{x})$ とその導関数 $\mathbf{g}(\mathbf{x})$, 及び初期ベクトル $\mathbf{x}_0$ が与えられたとき, 準ニュートン法により, $f(\mathbf{x})$ の極小値を与えるベクトル $\mathbf{x}^*$ と, その関数値 $f(\mathbf{x}^*)$ を求める.

ただし, $f(\mathbf{x})$ は 2 階までの連続な偏導関数を持つものと仮定する. $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dming1(x, n, fun, grad, epsr, &max, &f, g, h, vw, &icon);
```

引数の説明:

x	double x[n]	入力	初期ベクトル $\mathbf{x}_0$.
		出力	ベクトル $\mathbf{x}^*$.
n	int	入力	変数の個数 n .
fun	function	入力	$f(\mathbf{x})$ を計算するユーザ関数名. プロトタイプ宣言: double fun(double x[]);
			引数の説明:
		x	double 入力 変数ベクトル $\mathbf{x}$.
			x[n]
grad	function	入力	$\mathbf{g}(\mathbf{x})$ を計算するユーザ関数名. プロトタイプ宣言: void grad(double x[], double g[]);
			引数の説明:
		x	double 入力 変数ベクトル $\mathbf{x}$.
			x[n]
		g	double 出力 傾斜ベクトル.
			g[n]
			$g[i-1] = \partial f / \partial x_i, i = 1, \dots, n$
epsr	double	入力	収束判定値 (≥ 0). 0 のときは標準値が採用される. ("使用上の注意" a)参照)
max	int	入力	関数 $f(\mathbf{x})$ 及び $\mathbf{g}(\mathbf{x})$ の評価回数の上限 ($\neq 0$). ("使用上の注意" b)参照)
		出力	実際に評価した回数.
f	double	出力	関数値 $f(\mathbf{x}^*)$.
g	double g[n]	出力	傾斜ベクトル $\mathbf{x}^*$.
h	double h[Hlen]	出力	$\mathbf{x}^*$ におけるヘシアン行列の逆行列. 対称行列用圧縮格納法. $Hlen = n(n+1)/2$ 格納法の詳細については"1.3.2 対称行列の格納方法" を参照のこと.

vw double 作業領域
 vw[3n+1]
 icon int 出力 コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	指定された関数の評価回数内で, 収束条件が満たされなかった.	引数 x, f, g, h には, 最後の値を格納する.
20000	計算の過程で, $g_k^T p_k \geq 0$ となり, 関数の局所的減少がみられなかった.	処理を打ち切る. 引数 x 及び f には, 最後の値を格納する.
25000	ある探索方向に沿って関数が単調に減少している.	処理を打ち切る.
30000	次のいずれかであった. • $n < 1$ • $\text{epsr} < 0$ • $\text{max} = 0$	処理を打ち切る.

3. 使用上の注意

a) 引数 epsr

本関数の収束判定は, 反復ベクトル $\mathbf{x}_k$ において,

$$\|\mathbf{x}_{k+1} - \mathbf{x}_k\|_{\infty} \leq \max(1, \|\mathbf{x}_k\|_{\infty}) \cdot \text{epsr}$$

となった場合に, $\mathbf{x}_{k+1}$ を極小点 $\mathbf{x}^*$ とみなし反復を終了する. 本関数では, 関数 $f(\mathbf{x})$ が極小点 $\mathbf{x}^*$ の近傍で近似的に 2 次関数であることを前提としており, 関数値 $f(\mathbf{x}^*)$ を丸め誤差程度まで正しく求めるためには, $\text{epsr} \approx \mu^{1/2}$ (μ は丸め誤差の単位) 程度に与えるのが適当である. epsr の標準値は $\mu^{1/2}/8$ である.

b) 引数 max

関数の評価回数は, 変数ベクトル $\mathbf{x}$ に対して, $f(\mathbf{x})$ を計算することを 1 回, $\mathbf{g}(\mathbf{x})$ を計算することを n 回と数える.

関数の評価回数は, 関数の性質, 初期ベクトル, 収束判定値などに依存する. 一般には, 初期ベクトルが良く, 収束判定値として標準値採用した場合は, $\text{max} = 400 \cdot n$ 程度が妥当である.

指定評価回数内で収束条件が満たされず, icon = 10000 として戻った場合でも, 再度本関数を呼び出すことにより, 継続して反復することができる. しかし, この場合は引数 max には, 負の値で追加評価回数を指定し, 他の引数の内容は保存したままで呼び出すこと.

4. 使用例

$f(\mathbf{x}) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$ の最小点 $\mathbf{x}^*$ を初期ベクトル $\mathbf{x}_0 = (-1.2, 1.0)^T$ により求める.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define N 2

double fun(double x[]); /* user function prototype */
void grad(double x[], double g[]); /* derivative prototype */

MAIN__()
{
    int ierr, icon;
    double x[N], g[N], h[N*(N+1)/2], vw[3*N+1], f, epsr;
    int i, n, max;
```

```

/* initialize data */
n = N;
x[0] = -1.2;
x[1] = 1;
epsr = 1e-4;
max = 400*n;
/* find minimum of function */
ierr = c_dming1(x, n, fun, grad, epsr, &max, &f, g, h, vw, &icon);
if (icon >= 20000) {
    printf("ERROR in c_dming1. icon = %i", icon);
    exit(1);
}
printf("icon = %i    max = %i    f = %12.4e\n", icon, max, f);
printf("x: ");
for (i=0;i<n;i++) printf("%12.4e  ",x[i]);
printf("\n");
return(0);
}

/* user function */
double fun(double x[])
{
    return pow(1-x[0],2)+100*pow(x[1]-x[0]*x[0],2);
}

/* derivative function */
void grad(double x[], double g[])
{
    g[0] = -2*(1-x[0])-400*x[0]*(x[1]-x[0]*x[0]);
    g[1] = 200*(x[1]-x[0]*x[0]);
}

```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”のMING1の項目及び[95]を参照のこと.

c_dmsbv

実対称バンド行列と実ベクトルの積

ierr = c_dmsbv(a, n, nh, x, y, &icon);

1. 機能

$n \times n$, 上, 下バンド幅 h の実対称バンド行列 $\mathbf{A}$ と, 実ベクトル $\mathbf{x}$ の積 $\mathbf{y}$ を求める.

$$\mathbf{y} = \mathbf{A}\mathbf{x} \quad (1)$$

ここで, $\mathbf{x}$ 及び $\mathbf{y}$ は n 次元ベクトルである. $n > h \geq 0$ であること.

2. 引数

呼出し形式:

```
ierr = c_dmsbv(a, n, nh, x, y, &icon);
```

引数の説明:

a	double a[Alen]	入力	行列 $\mathbf{A}$. 対称バンド行列用圧縮格納法. $Alen = n(h+1) - h(h+1)/2$ 格納法の詳細については“1.3.4 バンド行列の格納方法”を参照のこと.
n	int	入力	行列 $\mathbf{A}$ の次数 n . (“使用上の注意” 参照)
nh	int	入力	上, 下バンド幅 h .
x	double x[n]	入力	ベクトル $\mathbf{x}$.
y	double y[n]	入力	ベクトル $\mathbf{y}'$. “使用上の注意” の(2)の計算を行うときのみ使用する.
		出力	行列 $\mathbf{A}$ とベクトル $\mathbf{x}$ の積 $\mathbf{y}$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	次のいずれかであった. $n = 0$ $nh < 0$ $nh \geq n$	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

本関数は通常(1)の計算を主とするが, $n = -n$ として, 任意のベクトル $\mathbf{y}'$ を引数 $\mathbf{y}$ に与えると

$$\mathbf{y} = \mathbf{y}' - \mathbf{A}\mathbf{x} \quad (2)$$

なる計算ができる.

具体的には, 連立 1 次方程式の残差ベクトル

$$\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}$$

の計算等を行う場合に利用できる.

4. 使用例

実対称バンド行列と実ベクトルの積を計算する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define max(a,b) ((a) > (b) ? (a) : (b))

#define NMAX 5
#define NHMAX 2

MAIN__()
{
    int ierr, icon;
    int n, nh, i, j, ij, jmin;
    double a[NMAX*(NHMAX+1)-NHMAX*(NHMAX+1)/2], x[NMAX], y[NMAX];

    /* initialize matrix and vector */
    n = NMAX;
    nh = NHMAX;
    ij = 0;
    for (i=0;i<n;i++) {
        jmin = max(i-nh, 0);
        for (j=jmin;j<=i;j++)
            a[ij++] = i-j+1;
    }
    for (i=0;i<n;i++)
        x[i] = i;
    /* perform matrix vector multiply */
    ierr = c_dmsbv(a, n, nh, x, y, &icon);
    if (icon != 0) {
        printf("ERROR: c_dmsbv failed with icon = %d\n", icon);
        exit(1);
    }
    /* print result */
    for (i=0;i<n;i++)
        printf("%7.2f ", y[i]);
    printf("\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”のMSBVの項目を参照のこと.

c_dmsgm

行列の積 (実対称行列・実行列)

ierr = c_dmsgm(a, b, kb, c, kc, n, vw, &icon);

1. 機能

$n \times n$ の実対称行列 **A** と, $n \times n$ の実行列 **B** の積 **C** を求める.

$$\mathbf{C} = \mathbf{AB}$$

ここで, **C** は $n \times n$ の実行列である. $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dmsgm(a, (double*)b, kb, (double*)c, kc, n, vw, &icon);
```

引数の説明:

a	double a[Alen]	入力	行列 A . 対称行列用圧縮格納法. $Alen = n(n+1)/2$ 格納法の詳細については“1.3.2 対称行列の格納方法”を参照のこと.
b	double b[n][kb]	入力	行列 B .
kb	int	入力	配列 b の整合寸法 ($\geq n$).
c	double c[n][kc]	出力	行列 C . (“使用上の注意” 参照)
kc	int	入力	配列 c の整合寸法 ($\geq n$).
n	int	入力	行列 A , B 及び C の次数 n .
vw	double vw[n]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	次のいずれかであった. • $n < 1$ • $kb < n$ • $kc < n$	処理を打ち切る.

3. 使用上の注意

a) 領域の節約について

配列 a 上の内容を保存する必要のない場合は, 引数 c のかわりに a を指定することにより領域が節約できる. このとき引数 c のほうが大きなサイズを必要とするので, 十分なサイズをとっておくこと.

4. 使用例

実対称行列と実行列の積を計算する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 5

/* print symmetric matrix */
void prtsymmat(double a[], int n)
{
    int ij, i, j;
    printf("symmetric matrix format\n");
    ij = 0;
    for (i=0;i<n;i++) {
        for (j=0;j<=i;j++)
            printf("%7.2f ",a[ij++]);
        printf("\n");
    }
}

/* print general matrix */
void prtgenmat(double *a, int k, int n, int m)
{
    int i, j;
    printf("general matrix format\n");
    for (i=0;i<n;i++) {
        for (j=0;j<m;j++)
            printf("%7.2f ",a[i*k+j]);
        printf("\n");
    }
}

MAIN__()
{
    int ierr, icon;
    int n, i, j, ij, kb, kc;
    double b[NMAX][NMAX], a[NMAX*(NMAX+1)/2], c[NMAX][NMAX], vw[NMAX];

    n = NMAX;
    /* initialize symmetric matrix */
    ij = 0;
    for (i=0;i<n;i++)
        for (j=0;j<=i;j++) {
            a[ij++] = i-j+1;
        }
    /* initialize general matrix */
    for (i=0;i<n;i++)
        for (j=0;j<n;j++) {
            b[i][j] = i+1;
        }
    kb = NMAX;
    kc = NMAX;
    /* matrix multiply */
    ierr = c_dmsgm(a, (double*)b, kb, (double*)c, kc, n, vw, &icon);
    /* print matrices */
    printf("a: \n");
    prtsymmat(a, n);
    printf("b: \n");
    prtgenmat((double*)b, kb, n, n);
    printf("c: \n");
    prtgenmat((double*)c, kc, n, n);
    return(0);
}
```

c_dmssm

行列の積 (実対称行列)

ierr = c_dmssm(a, b, c, kc, n, vw, &icon);

1. 機能

$n \times n$ の実対称行列 **A** と実対称行列 **B** の積 **C** を求める.

$$\mathbf{C} = \mathbf{AB}$$

ここで **C** は, $n \times n$ の実行列である. $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dmssm(a, b, (double *)c, kc, n, vw, &icon);
```

引数の説明:

a	double a[Alen]	入力	行列 A . 対称行列用圧縮格納法. $Alen = n(n+1)/2$ 格納法の詳細については“1.3.2 対称行列の格納方法”を参照のこと.
b	double b[Blen]	入力	行列 B . 対称行列用圧縮格納法. $Blen = n(n+1)/2$ 格納法の詳細については“1.3.2 対称行列の格納方法”を参照のこと.
c	double c[n][kc]	出力	行列 C . (“使用上の注意”参照)
kc	int	入力	配列 c の整合寸法 ($\geq n$).
n	int	入力	行列 A , B 及び C の次数 n .
vw	double vw[n]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	次のいずれかであった. • $n < 1$ • $kc < n$	処理を打ち切る.

3. 使用上の注意

a) 領域の節約について

配列 a 上の内容を保存する必要のない場合は, 引数 c のかわりに a を指定することにより領域が節約できる. このとき引数 c のほうが大きなサイズを必要とするので, 十分なサイズをとっておくこと.

4. 使用例

実対称行列の積を計算する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 5

/* print symmetric matrix */
void prtsymmat(double a[], int n)
{
    int ij, i, j;
    printf("symmetric matrix format\n");
    ij = 0;
    for (i=0;i<n;i++) {
        for (j=0;j<=i;j++)
            printf("%7.2f ",a[ij++]);
        printf("\n");
    }
}

/* print general matrix */
void prtgenmat(double *a, int k, int n, int m)
{
    int i, j;
    printf("general matrix format\n");
    for (i=0;i<n;i++) {
        for (j=0;j<m;j++)
            printf("%7.2f ",a[i*k+j]);
        printf("\n");
    }
}

MAIN__()
{
    int ierr, icon;
    int n, i, j, ij, kc;
    double a[NMAX*(NMAX+1)/2], b[NMAX*(NMAX+1)/2], c[NMAX][NMAX], vw[NMAX];

    n = NMAX;
    /* initialize symmetric matrices */
    ij = 0;
    for (i=0;i<n;i++)
        for (j=0;j<=i;j++) {
            a[ij] = i-j+1;
            b[ij++] = i-j+1;
        }
    kc = NMAX;
    /* matrix matrix multiply */
    ierr = c_dmssm(a, b, (double*)c, kc, n, vw, &icon);
    /* print matrices */
    printf("a: \n");
    prtsymmat(a, n);
    printf("b: \n");
    prtsymmat(b, n);
    printf("c: \n");
    prtgenmat((double*)c, kc, n, n);
    return(0);
}
```

c_dmsv

実対称行列と実ベクトルの積

<code>ierr = c_dmsv(a, n, x, y, &icon);</code>

1. 機能

$n \times n$ の実対称行列 $\mathbf{A}$ と、実ベクトル $\mathbf{x}$ の積 $\mathbf{y}$ を求める.

$$\mathbf{y} = \mathbf{Ax} \quad (1)$$

ここで、 $\mathbf{x}$ 及び $\mathbf{y}$ は n 次元ベクトルである。 $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dmsv(a, n, x, y, &icon);
```

引数の説明:

a	double a[Alen]	入力	行列 $\mathbf{A}$. 対称行列用圧縮格納法. $Alen = n(n+1)/2$ 格納法の詳細については“1.3.2 対称行列の格納方法”を参照のこと.
n	int	入力	行列 $\mathbf{A}$ の次数 n .
x	double x[n]	入力	ベクトル $\mathbf{x}$.
y	double y[n]	入力	ベクトル $\mathbf{y}'$. “使用上の注意”の(2)の計算を行うときのみ使用する.
		出力	行列 $\mathbf{A}$ とベクトル $\mathbf{x}$ の積 $\mathbf{y}$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	$n = 0$	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

本関数は通常(1)の計算を主とするが、 $n = -n$ として、任意のベクトル $\mathbf{y}'$ を引数 $\mathbf{y}$ に与えると、

$$\mathbf{y} = \mathbf{y}' - \mathbf{Ax} \quad (2)$$

なる計算ができる.

具体的には、連立 1 次方程式の残差ベクトル、

$$\mathbf{r} = \mathbf{b} - \mathbf{Ax}$$

の計算等を行う場合に利用できる.

4. 使用例

実対称行列と実ベクトルの積を計算する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 5

MAIN__()
{
    int ierr, icon;
    int n, i, j, ij;
    double a[NMAX*(NMAX+1)/2], x[NMAX], y[NMAX];

    /* initialize matrix and vector */
    n = NMAX;
    ij = 0;
    for (i=0;i<n;i++)
        for (j=0;j<=i;j++) {
            a[ij++] = i-j+1;
        }
    for (i=0;i<n;i++)
        x[i] = i;
    /* perform matrix vector multiply */
    ierr = c_dmsv(a, n, x, y, &icon);
    if (icon != 0) {
        printf("ERROR: c_dmsv failed with icon = %d\n", icon);
        exit(1);
    }
    /* print result */
    for (i=0;i<n;i++)
        printf("%7.2f ", y[i]);
    printf("\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”のMSVの項目を参照のこと.

c_dndf

正規分布関数 $\phi(x)$
<code>ierr = c_dndf(x, &f, &icon);</code>

1. 機能

正規分布関数

$$\phi(x) = \frac{1}{\sqrt{2\pi}} \int_0^x e^{-\frac{t^2}{2}} dt,$$

の値を, 関係式

$$\phi(x) = \operatorname{erf}\left(x / \sqrt{2}\right) / 2$$

より計算する.

2. 引数

呼出し形式:

```
ierr = c_dndf(x, &f, &icon);
```

引数の説明:

x	double	入力	独立変数 x .
f	double	出力	関数値 $\phi(x)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.

3. 使用上の注意

a) x の範囲

引数 x の範囲に制限はない.

b) `c_dndf` 及び `c_dndfc`

余正規分布関数 $\psi(x)$ との関係,

$$\phi(x) = 1/2 - \psi(x),$$

を利用すれば $\phi(x)$ の値を関数 `c_dndfc` を用いて計算することもできる. ただし, このように計算すると $|x| < 2$ の範囲の x については本関数 `c_dndf` を直接用いるのに比べ精度が悪くなるので注意すること.

4. 使用例

$\phi(x)$ の値を, 0.0 から 10.0 での 0.1 刻みの x の値に対して計算して数表を作る.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
    double x, f;
    int i;
```



```
for (i=0;i<=100;i++) {
    x = (double)i/10;
    /* calculate normal distribution function */
    ierr = c_dndf(x, &f, &icon);
    if (icon == 0)
        printf("x = %5.2f    f = %f\n", x, f);
    else
        printf("ERROR: x = %5.2f    f = %f    icon = %i\n", x, f, icon);
}
return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の NDF の項目を参照のこと.

c_dndfc

余正規分布関数 $\psi(x)$

<code>ierr = c_dndfc(x, &f, &icon);</code>

1. 機能

余正規分布関数,

$$\psi(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-\frac{t^2}{2}} dt,$$

の値を, 関係式,

$$\psi(x) = \operatorname{erfc}\left(x / \sqrt{2}\right) / 2$$

より計算する.

2. 引数

呼出し形式:

```
ierr = c_dndfc(x, &f, &icon);
```

引数の説明:

x	double	入力	独立変数 x .
f	double	出力	関数値 $\psi(x)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.

3. 使用上の注意

a) x の範囲

引数 x の範囲に制限はない.

b) `c_dndfc` 及び `c_dndf`

正規分布関数 $\phi(x)$ との関係,

$$\psi(x) = 1/2 - \phi(x),$$

を利用すれば $\psi(x)$ の値を関数 `c_dndf` を用いて計算することもできる. ただし, このように計算すると $|x| > 2$ の範囲の x については本関数 `c_dndfc` を直接用いるのに比べ精度が悪くなるので注意すること.

4. 使用例

$\psi(x)$ の値を, 0.0 から 10.0 までの 0.1 刻みの x の値に対して計算して数表を作る.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
    double x, f;
    int i;
```

```
for (i=0;i<=100;i++) {
    x = (double)i/10;
    /* calculate complementary normal distribution function */
    ierr = c_dndfc(x, &f, &icon);
    if (icon == 0)
        printf("x = %5.2f    f = %f\n", x, f);
    else
        printf("ERROR: x = %5.2f    f = %f    icon = %i\n", x, f, icon);
}
return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の NDFC の項目を参照のこと.

c_dnlpg1

非線形計画問題 (微係数要, パウエル法)

<pre>ierr = c_dnlpg1(x, n, fun, grad, func, jac, m, epsr, &max, &f, vw, k, ivw, &icon);</pre>

1. 機能

n 変数の実関数 $f(\mathbf{x})$ とその導関数 $\mathbf{g}(\mathbf{x})$ 及び初期ベクトル $\mathbf{x}_0$ が与えられたとき, 制約条件

$$\begin{aligned} c_i(\mathbf{x}) &= 0, \quad i = 1, 2, \dots, m_1 \\ c_i(\mathbf{x}) &\geq 0, \quad i = m_1 + 1, m_1 + 2, \dots, m_1 + m_2 \end{aligned}$$

のもとで, $f(\mathbf{x})$ の極小値を与えるベクトル $\mathbf{x}^*$ とその関数値 $f(\mathbf{x}^*)$ を求める.

ただし, $\{c_i(\mathbf{x})\}$ のヤコビアン $\mathbf{J}(\mathbf{x})$ も関数として与えられるものとする. また, $f(\mathbf{x})$ は 2 階までの連続な偏微係数を持つものと仮定する.

ここで, $\mathbf{x} = (x_1, x_2, \dots, x_n)^T$ である. また, m_1, m_2 はそれぞれ等式制約, 不等式制約の個数である.

以降, $m = m_1 + m_2$ とする. $n \geq 1, m_1 \geq 0, m_2 \geq 0$ かつ $m \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dnlpg1(x, n, fun, grad, func, jac, m, epsr, &max, &f, vw, k, ivw,
                &icon);
```

引数の説明:

x	double x[n]	入力	初期ベクトル $\mathbf{x}_0$.
		出力	ベクトル $\mathbf{x}^*$.
n	int	入力	変数の個数 n .
fun	function	入力	関数 $f(\mathbf{x})$ を計算するユーザ関数名. プロトタイプ宣言: double fun(double x[]);
			引数の説明:
		x	double 入力 変数ベクトル $\mathbf{x}$.
			x[n]
grad	function	入力	関数 $\mathbf{g}(\mathbf{x})$ を計算するユーザ関数名. プロトタイプ宣言: void grad(double x[], double g[]);
			引数の説明:
		x	double 入力 変数ベクトル $\mathbf{x}$.
			x[n]
		g	double 出力 $f(\mathbf{x})$ の傾斜ベクトル.
			g[i - 1] = $\{\partial f / \partial x_i\}$,
			$i = 1, \dots, n$

func	function	入力	関数 $\{c_i(\mathbf{x})\}$ を計算するユーザ関数名. プロトタイプ宣言: void func(double x[], double c[]); 引数の説明: x double 入力 変数ベクトル $\mathbf{x}$. x[n] c double 出力 制約値のベクトル. c[m]
jac	function	入力	関数 $\mathbf{J}(\mathbf{x})$ を計算するユーザ関数名. $\mathbf{J} = \begin{bmatrix} \frac{\partial c_1}{\partial x_1} & \frac{\partial c_1}{\partial x_2} & \dots & \frac{\partial c_1}{\partial x_n} \\ \frac{\partial c_2}{\partial x_1} & \frac{\partial c_2}{\partial x_2} & \dots & \frac{\partial c_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial c_m}{\partial x_1} & \frac{\partial c_m}{\partial x_2} & \dots & \frac{\partial c_m}{\partial x_n} \end{bmatrix}$ プロトタイプ宣言: void jac(double x[], double cj[], int k); 引数の説明: x double 入力 変数ベクトル $\mathbf{x}$. x[n] cj double 出力 ヤコビアン行列. cj[m*k] $c_j [(i-1)*k + (j-1)]$ $= \partial c_i / \partial x_j$ $1 \leq i \leq m$ かつ $1 \leq j \leq n$ であること. k int 入力 配列 cj の整合寸法.
m	int m[2]	入力	制約式の個数. m[0] = m_1 , m[1] = m_2 .
epsr	double	入力	収束判定値. epsr = 0 のときは標準値が採用される. ("使用上の注意" a)参照)
max	int	入力	関数 $f(\mathbf{x})$, $\mathbf{g}(\mathbf{x})$, $c(\mathbf{x})$ 及び, $\mathbf{J}(\mathbf{x})$ の評価回数の上限. ("使用上の注意" b)参照)
f	double	出力	実際に評価した回数.
vw	double vw[Rlen]	出力 作業領域	関数値 $f(\mathbf{x}^*)$. $Rlen = k*(m[0]+m[1]+2*n+12)$.
k	int	入力	配列 vw の整合寸法. $k \geq m[0]+m[1]+n+4$.
ivw	int ivw[Ilen]	作業領域	$Ilen = 2*(m[0]+m[1]+n+4)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード：

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
10000	指定された関数の評価回数内で, 収束条件が満たされなかった.	処理を打ち切る. 引数 x, f には, 最後の値を格納する.
20000	計算の過程で, 関数の局所的減少が満たされなかった. $epsr$ が小さすぎる.	
21000	制約式を満たす解が存在しないか, 又は初期値 x_0 の選び方が適切でない可能性がある. 初期値 x_0 を変えて再実行するとよい.	処理を打ち切る.
30000	次のいずれかであった. • $n < 1$ • $epsr < 0$ • $m[0] < 0$ • $m[1] < 0$ • $m[0] + m[1] < 1$ • $k < m[0] + m[1] + n + 4$ • $max = 0$	処理を打ち切る.

3. 使用上の注意

a) 引数 $epsr$ の与え方

本関数の収束判定は, 反復ベクトル x_k において,

$$\|x_{k+1} - x_k\|_{\infty} \leq \max(1, \|x_k\|_{\infty}) \cdot epsr$$

となった場合に, x_{k+1} を極小点 x^* とみなし反復を終了する.

本関数では, 関数 $f(x)$ が極小点 x^* の近傍で近似的に 2 次関数であることを前提としており, 関数値 $f(x^*)$ を丸め誤差程度まで正しく求めるためには,

$$epsr \approx \sqrt{\mu}, \mu \text{ は丸め誤差の単位}$$

程度に与えるのが適当である.

$epsr$ の標準値は $2 \cdot \sqrt{\mu}$ である.

b) 引数 max の与え方

関数の評価回数は, 変数ベクトル x に対して, $f(x)$ を計算することを 1 回, $g(x)$ を計算することを n 回, $c(x)$ を計算することを m 回, $J(x)$ を計算することを nm 回と数える.

関数の評価回数は, 関数の性質, 初期ベクトル, 収束判定値に依存する.

一般には, 初期ベクトルが良く, 収束判定として標準値を採用した場合には, $max = 800 \cdot nm$ 程度が妥当である.

指定評価回数内で収束条件が満たされず, $icon = 10000$ として戻った場合でも, 再度本関数を呼び出すことにより, 継続して反復することができる. しかし, この場合は引数 max には, 負の値で追加評価回数を指定し, 他の引数の内容は保存したままで呼び出すこと.

4. 使用例

2 変数の実関数 $f(x_1, x_2) = x_1^2 - 2x_1x_2 + 2x_2^2 - 10x_1 + x_2$ を制約条件,

$$c_1(x_1, x_2) = 0.5x_1^2 + 1.5x_2^2 - 2 = 0$$

$$c_2(x_1, x_2) = -x_1 + x_2 \geq 0$$

のもとで最小化する. 初期ベクトルは $\mathbf{x}_0 = (-2, 2)^T$ とする.

```
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define N 2
#define M1 1
#define M2 1

double fun(double x[]);
void grad(double x[], double g[]);
void func(double x[], double c[]);
void jac(double x[], double *cj, int k);

MAIN__()
{
    int ierr, icon;
    double x[N], epsr, f, vw[M1+M2+2*N+12][M1+M2+N+4], eps;
    int n, m[2], max, k, ivw[2*(M1+M2+N+4)];

    /* initialize data */
    x[0] = -2;
    x[1] = 2;
    n = N;
    m[0] = M1;
    m[1] = M2;
    epsr = 1e-3;
    max = 800*(M1+M2)*N;
    k = M1+M2+N+4;
    /* minimize */
    ierr = c_dnlpg1(x, n, fun, grad, func, jac,
                    m, epsr, &max, &f, (double*)vw, k, ivw, &icon);
    printf("icon = %i    max = %i    f = %f\n", icon, max, f);
    printf("x[0] = %f    x[1] = %f\n", x[0], x[1]);
    /* check result */
    eps = 1e-5;
    if (fabs((f+8)/8) > eps)
        printf("Inaccurate result\n");
    else
        printf("Result OK\n");
    return(0);
}

/* objective function */
double fun(double x[])
{
    return((x[0]-2*x[1]-10)*x[0] + (2*x[1]+1)*x[1]);
}

/* gradient function */
void grad(double x[], double g[])
{
    g[0] = 2*x[0]-2*x[1]-10;
    g[1] = -2*x[0]+4*x[1]+1;
    return;
}

/* constraint function */
void func(double x[], double c[])
{
    c[0] = 0.5*x[0]*x[0]+1.5*x[1]*x[1]-2;
    c[1] = -x[0]+x[1];
    return;
}
```

```
/* Jacobian function */
void jac(double x[], double *cj, int k)
{
    cj[0] = x[0];      /* [0][0] */
    cj[1] = 3*x[1];    /* [0][1] */
    cj[k] = -1;        /* [1][0] */
    cj[k+1] = 1;       /* [1][1] */
    return;
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の NLPG1 の項目及び[86], [87]を参照のこと.

c_dnolbr

連立非線型方程式 (ブレント法)

```
ierr = c_dnolbr(x, n, fun, epsz, epst, fc,
               &m, &fnor, vw, &icon);
```

1. 機能

連立非線型方程式 (1) をブレント (Brent)法により解く.

$$\left. \begin{aligned} f_1(x_1, x_2, \dots, x_n) &= 0 \\ f_2(x_1, x_2, \dots, x_n) &= 0 \\ &\vdots \\ f_n(x_1, x_2, \dots, x_n) &= 0 \end{aligned} \right\} \quad (1)$$

すなわち, $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_n(\mathbf{x}))^T$, $\mathbf{x} = (x_1, x_2, \dots, x_n)^T$ として, 与えられた初期ベクトル $\mathbf{x}_0$ より, (2) の解を求める.

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \quad (2)$$

ここで, $\mathbf{0}$ は n 次元の 0 ベクトルである. $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dnolbr(x, n, fun, epsz, epst, fc, &m, &fnor, vw, &icon);
```

引数の説明:

x	double x[n]	入力	求めようとする解の初期ベクトル $\mathbf{x}_0$.
		出力	解ベクトル.
n	int	入力	方程式の元数 n .
fun	function	入力	$f_k(\mathbf{x})$ を計算するユーザ関数名. プロトタイプ宣言: double fun(double x[], int k);
			引数の説明:
x	double	入力	変数ベクトル $\mathbf{x}$.
	x[n]		
k	int	入力	方程式における式番号を 与える ($1 \leq k \leq n$). 例えば $k = k$ と入力されたとき関 数 fun には $f_k(\mathbf{x})$ の値を 代入すること.
epsz	double	入力	収束判定値 (≥ 0). $\ \mathbf{f}(\mathbf{x}_i)\ _{\infty} \leq \text{epsz}$ となったとき, 収束したと見なす. ("使用上の注意" a)参照)
epst	double	入力	収束判定値 (≥ 0). $\ \mathbf{x}_i - \mathbf{x}_{i-1}\ _{\infty} \leq \text{epst} \cdot \ \mathbf{x}_i\ _{\infty}$ となったとき, 収束したと見 なす. ("使用上の注意" a)参照)

fc	double	入力	解ベクトルの探索範囲を示す数値 (> 0). 反復の過程で $\ \mathbf{x}_i\ _\infty > \text{fc} \cdot \max(\ \mathbf{x}_0\ _\infty, 1)$ となったとき解ベクトルの探索を止める. (“使用上の注意” b)参照)
m	int	入力	反復回数の上限 (> 0). (“使用上の注意” c)参照)
		出力	実際に反復した回数.
fnor	double	出力	求まった解ベクトルにおける $\ f(\mathbf{x}_i)\ _\infty$ の値.
vw	double vw[Vwlen]	作業領域	Vwlen = n*(n+3)
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
1	収束判定 $\ f(\mathbf{x}_i)\ _\infty \leq \text{epsz}$ が満たされた.	
2	収束判定 $\ \mathbf{x}_i - \mathbf{x}_{i-1}\ _\infty \leq \text{epst} \cdot \ \mathbf{x}_i\ _\infty$ が満たされた.	
10000	与えられた反復回数内で指定された収束条件を満たさなかった.	引数 x には最後の反復ベクトル $\mathbf{x}_i$ が格納される.
20000	解ベクトルの探索範囲内で解ベクトルを見つけられなかった. (引数 fc の項参照)	
25000	計算の過程で関数 $f(\mathbf{x})$ のヤコビアンが 0 になった.	
30000	次のいずれかであった. • $n \leq 0$ • $\text{epsz} < 0$ • $\text{epst} < 0$ • $\text{fc} \leq 0$ • $m \leq 0$	処理を打ち切る.

3. 使用上の注意

a) epsz と epst の与え方

本関数は, 収束判定法として二とおり用意しており, 少なくともどちらか一方が満たされたとき反復を止める. もし利用者がどちらか一方の判定法だけを適用したい場合は, 他方の判定値を 0.0 と与えればよい. 具体的には以下のような与え方ができる.

① $\text{epsz} = \varepsilon_A$ (> 0), $\text{epst} = 0$ のとき

$\|\mathbf{x}_i - \mathbf{x}_{i-1}\|_\infty = 0$ とならない限り, $\|f(\mathbf{x}_i)\|_\infty \leq \varepsilon_A$ が満たされるか反復回数が m になるまで反復する.

② $\text{epsz} = 0$, $\text{epst} = \varepsilon_B$ (> 0) のとき

$\|f(\mathbf{x}_i)\|_\infty = 0$ とならない限り, $\|\mathbf{x}_i - \mathbf{x}_{i-1}\|_\infty \leq \varepsilon_B \cdot \|\mathbf{x}_i\|_\infty$ が満たされるか反復回数が m になるまで反復する.

③ $\text{epsz} = 0$, $\text{epst} = 0$ のとき

$\|f(\mathbf{x}_i)\|_\infty = 0$ 又は $\|\mathbf{x}_i - \mathbf{x}_{i-1}\|_\infty = 0$ とならない限り m 回反復する.

無条件に m 回反復したいときは, 上記③の与え方をすればよい.

b) fc の意味

方程式の性質によっては, 初期ベクトル $\mathbf{x}_0$ の近傍に解が無いことがある. このようなとき, 反復ベクトル $\mathbf{x}_i$ は $\mathbf{x}_0$ から大きく離れ, その結果 $f(\mathbf{x})$ の計算時にオーバーフローなどの不都合を起こす場合がある.

引数 fc はこれを防ぐために設けられ, 解ベクトルの探索範囲を指定するものである.

標準値としては, 100 程度でよい.

c) m の与え方

解ベクトルへ収束するまでの反復回数は, 方程式の性質及び収束判定値に依存する. 一般に初期ベクトルが悪いとき, あるいは収束判定値が小さいときは, 引数 m は大きめに与えた方がよい.

元数 n が 10 程度なら m は 50 程度でよい.

4. 使用例

2 元の連立非線型方程式:

$$\begin{aligned}x_1 \cdot (1 - x_2^2) &= 2.25 \\x_1 \cdot (1 - x_2^3) &= 2.625\end{aligned}$$

の解を初期ベクトル $\mathbf{x}_0 = (5.0, 0.8)^T$ より求める. 真の解は $\mathbf{x} = (3.0, 0.5)^T$ 及び $\mathbf{x} = (81/32, -1/3)^T$ である.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

#define N 2

double fun(double x[], int k); /* user function prototype */

MAIN__()
{
    int ierr, icon;
    double x[N], epsz, epst, fc, fnor, vw[N*(N+3)];
    int m, n;

    n = N;
    x[0] = 5.0;
    x[1] = 0.8;
    epsz = 1e-5;
    epst = 0;
    fc = 100;
    m = 20;
    /* solve equations */
    ierr = c_dnolbr(x, n, fun, epsz, epst, fc, &m, &fnor, vw, &icon);
    printf("icon = %i   m = %i   fnor = %f   x[0] = %12.4e   x[1] = %12.4e\n",
           icon, m, fnor, x[0], x[1]);
    return(0);
}

/* user function */
double fun(double x[], int k)
{
    double res;
    switch (k) {
    case (1):
        res = x[0]*(1-x[1]*x[1])-2.25;
        break;
    case (2):
        res = x[0]*(1-x[1]*x[1]*x[1])-2.625;
        break;
    }
}
```

```
    return(res);  
}
```

5. 手法概要

本関数は, 連立非線型方程式 (1) をブレント (Brent) 法を用いて解ベクトルを求める.

計算方法の詳細については“富士通 SSL II 使用手引書”の NOLBR の項目及び[24]を参照のこと.

c_dnolf1

関数二乗和の極小化 (微係数不要, 改訂マルカート法)

<pre>ierr = c_dnolf1(x, n, fun, m, epsr, &max, f, &sums, vw, k, &icon);</pre>

1. 機能

n 変数の m 個の実関数 $f_1(\mathbf{x})$, $f_2(\mathbf{x})$, ..., $f_m(\mathbf{x})$ と初期ベクトル $\mathbf{x}_0$ が与えられたとき, 改訂マルカート法 (Levenberg-Marquardt-Morrison 法; LMM 法) により,

$$F(\mathbf{x}) = \sum_{i=1}^m (f_i(\mathbf{x}))^2$$

の極小値を与えるベクトル $\mathbf{x}^*$ とその関数値 $F(\mathbf{x}^*)$ を求める. 本関数では, 微係数は不要であるが, $f_i(\mathbf{x})$, $i = 1, \dots, m$ は 1 階までの連続な偏微分をもつものと仮定する. また, $m \geq n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dnolf1(x, n, fun, m, epsr, &max, f, &sums, vw, k, &icon);
```

引数の説明:

x	double x[n]	入力	変数ベクトル $\mathbf{x}_0$.
		出力	ベクトル $\mathbf{x}^*$.
n	int	入力	変数の個数 n .
fun	function	入力	$f_i(\mathbf{x})$ を計算するユーザ関数名. プロトタイプ宣言: void fun(double x[], double f[]);
			引数の説明:
x	double x[n]	入力	変数ベクトル $\mathbf{x}$.
f	double f[m]	出力	関数値 $f_i(\mathbf{x})$. $f[i-1] = f_i(\mathbf{x})$, $i = 1, 2, \dots, m$.
m	int	入力	関数の個数 m .
epsr	double	入力	収束判定値. $\text{epsr} = 0$ のときは標準値が採用される. ("使用上の注意" a)参照)
max	int	入力	関数の評価回数の上限. ("使用上の注意" b)参照)
f	double f[m]	出力	実際に評価した回数.
		出力	関数値 $f_i(\mathbf{x}^*)$. $f[i-1] = f_i(\mathbf{x}^*)$, $i = 1, 2, \dots, m$.
sums	double	出力	関数二乗和 $F(\mathbf{x}^*)$ の値.
vw	double vw[Rlen]	作業領域	$Rlen = k \cdot (n+2)$.
k	int	入力	配列 vw の整合寸法. $k \geq n+m$.

icon int 出力 コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	指定された関数の評価回数内で, 収束条件が満たされなかった.	処理を打ち切る. 引数 x, f, sums には, 最後の値を格納する.
20000	計算の過程で, Marquardt 数が上限を超えた. epsr が小さすぎるか, ヤコビアン行列の差分近似の誤差が計算限界を超えた.	処理を打ち切る. 引数 x, f, sums には, 最後の値を格納する.
30000	次のいずれかであった. • $n < 1$ • $m < n$ • $epsr < 0$ • $max = 0$ • $k < n+m$	処理を打ち切る.

3. 使用上の注意

a) 引数 epsr の与え方

本関数では, $F(\mathbf{x})$ が極小点 $\mathbf{x}^*$ の近傍で近似的に 2 次関数であることを前提としており, $F(\mathbf{x}^*)$ を丸め誤差程度まで正しく求めるためには,

$$epsr \approx \sqrt{\mu}, \quad \mu \text{ は丸め誤差の単位}$$

程度に与えるのが適当である.

epsr の標準値は $2 \cdot \sqrt{\mu}$ である.

b) 引数 max の与え方

関数の評価回数は, 変数ベクトル $\mathbf{x}$ に対して, $f_i(\mathbf{x}), i = 1, \dots, m$ を計算することを 1 回と数える. すなわち, ユーザ関数 fun の呼び出し回数に一致する.

この関数の評価回数は, 関数 $\{f_i(\mathbf{x})\}$ の性質, 初期ベクトル, 収束判定値などに大きく依存する.

一般には, 初期ベクトルが良く, 収束判定値として標準値を採用した場合には, $max = 100 \cdot n \cdot m$ 程度が妥当である.

指定評価回数内で収束条件が満たされず, icon = 10000 として戻った場合でも, 再度本関数を呼び出すことにより, 継続して反復することができる. しかし, この場合は引数 max には, 負の値で追加評価回数を指定し, 他の引数の内容は保存したままで呼び出すこと.

4. 使用例

$F(x_1, x_2) = f_1^2(x_1, x_2) + f_2^2(x_1, x_2)$, ここで:

$$\begin{aligned} f_1(x_1, x_2) &= 1 - x_1 \\ f_2(x_1, x_2) &= 10(x_2 - x_1^2) \end{aligned}$$

の極小点を $\mathbf{x}_0 = (-1.2, 1.0)^T$ から出発して求める.

```
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */
```

```

#define N 2
#define M 2

void fun(double x[], double y[]); /* user function prototype */

MAIN__()
{
    int ierr, icon;
    double f[N], x[N], vw[N+2][M+N], epsr, sums, eps, exact;
    int max, n, m, k;

    /* initialize data */
    x[0] = -1.2;
    x[1] = 1;
    n = N;
    m = M;
    epsr = 1e-3;
    max = 100*n*m;
    k = m+n;
    /* find minimum of sum of squares */
    ierr = c_dnolf1(x, n, fun, m, epsr, &max, f, &sums, (double*)vw, k, &icon);
    printf("icon = %i    max = %i    sums = %12.4e\n", icon, max, sums);
    printf("x = (%12.4e, %12.4e)    f = (%12.4e, %12.4e)\n",
           x[0], x[1], f[0], f[1]);
    /* check result */
    exact = 0;
    eps = 1e-6;
    if (fabs(sums-exact) > eps)
        printf("Inaccurate result\n");
    else
        printf("Result OK\n");
    return(0);
}

/* user function */
void fun(double x[], double y[])
{
    y[0] = 1 - x[0];
    y[1] = (x[1] - x[0]*x[0])*10;
    return;
}

```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”のNOLF1の項目及び[69], [82]を参照のこと.

c_dnolg1

関数二乗和の極小化 (微係数要, 改訂マルカート法)

ierr = c_dnolg1(x, n, fun, jac, m, epsr, &max, f, &sums, vw, k, &icon);

1. 機能

n 変数の m 個の実関数 $f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_m(\mathbf{x})$ とそのヤコビアン $\mathbf{J}(\mathbf{x})$, 及び初期ベクトル $\mathbf{x}_0$ が与えられたとき, 改訂マルカート法 (Levenberg-Marquardt-Morrison 法, 以下 LMM 法と略す)により,

$$F(\mathbf{x}) = \sum_{i=1}^m (f_i(\mathbf{x}))^2$$

の極小値を与えるベクトル $\mathbf{x}^*$ とその関数値 $F(\mathbf{x}^*)$ を求める. 本関数では $f_i(\mathbf{x}), i=1, \dots, m$ は 1 階までの連続な偏微分をもつものと仮定する. また, $m \geq n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dnolg1(x, n, fun, jac, m, epsr, &max, f, &sums, vw, k, &icon);
```

引数の説明:

x	double x[n]	入力	初期ベクトル $\mathbf{x}_0$.
		出力	ベクトル $\mathbf{x}^*$.
n	int	入力	変数の個数 n .
fun	function	入力	$f_i(\mathbf{x})$ を計算する関数名. プロトタイプ宣言: void fun(double x[], double f[]);
		引数の説明:	
		x	double 入力 変数ベクトル $\mathbf{x}$. x[n]
		f	double 出力 変数ベクトル $\mathbf{x}$ に対する 関数値 $f_i(\mathbf{x})$. f[i-1] = f[i] ($i=1, 2, \dots, m$).
jac	function	入力	ヤコビアン $\mathbf{J}(\mathbf{x})$ を計算する関数名. $\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \dots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$ プロトタイプ宣言: void jac(double x[], double g[], long k);
		引数の説明:	
		x	double 入力 変数ベクトル $\mathbf{x}$. x[n]

			g	double g[m*k]	出力	変数ベクトル $\mathbf{x}$ に対する ヤコビアン行列 $\mathbf{J}(\mathbf{x})$. $g[(i-1)*k+(j-1)]$ $= \partial f_i / \partial x_j$, $i=1,\dots,m$ 及び $j=1,\dots,n$. g を 2 次元配列と考えたと きの整合寸法.
			k	int	入力	
m	int					関数の個数 m .
epsr	double	入力				収束判定値(≥ 0). 0.0 のときは, 標準値が採用される. (“使用上の注意”a)参照)
max	int	入力				関数の評価回数の上限($\neq 0$). (“使用上の注意”b)参照)
		出力				実際に評価した回数.
f	double f[m]	出力				関数値 $f_i(\mathbf{x}^*)$.
sums	double	出力				関数二乗和 $F(\mathbf{x}^*)$ の値.
vw	double vw[Vwlen]	作業領域				$Vwlen = k \cdot (n+2)$.
k	int	入力				vw を 2 次元配列として利用するときの整合寸法. $k \geq m+n$.
icon	int	出力				コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	指定された反復回数内で収束条件が満たされなかった.	引数 x, f, sums には最後の値を格納する.
20000	計算の過程で, Marquardt 数 v_k が上界を超えた. epsr が小さすぎるか, ヤコビアン行列の差分近似の誤差が計算限界を超えた.	処理を打ち切る. 引数 x, f, sums には最後の値を格納する.
30000	次のいずれかであった. $n < 1$ $m < n$ $\text{epsr} < 0$ $\text{max} = 0$ $k < m+n$	処理を打ち切る.

3. 使用上の注意

a) epsr

本関数では反復過程において, 次のような収束判定を行っている.

- $F(\mathbf{x}_{k+1}) < F(\mathbf{x}_k)$ かつ $\|\mathbf{x}_{k+1} - \mathbf{x}_k\|_\infty \leq \max(1, \|\mathbf{x}_k\|_\infty) \cdot \text{epsr}$ を満たすとき, $\mathbf{x}_{k+1}$ を極小点 $\mathbf{x}^*$ とみなす.
- $F(\mathbf{x}_{k+1}) \geq F(\mathbf{x}_k)$ かつ $\|\mathbf{x}_{k+1} - \mathbf{x}_k\|_\infty \leq \max(1, \|\mathbf{x}_k\|_\infty) \cdot \text{epsr}$ を満たすとき, $\mathbf{x}_k$ を極小点 $\mathbf{x}^*$ とみなす.

本関数では, $F(\mathbf{x})$ が極小点 $\mathbf{x}^*$ の近傍で近似的に 2 次関数であることを前提としており, $F(\mathbf{x}^*)$ を丸め誤差程度まで正しく求めるためには, $\text{epsr} = \mu^{1/2}$ (μ は丸め誤差の単位)程度に与えるのが適当である. epsr の標準値は $2\mu^{1/2}$ である.

b) max の与え方

関数の評価回数は, 変数ベクトル $\mathbf{x}$ に対して, $f_i(\mathbf{x}), i=1, \dots, m$ を計算することを 1 回, $\mathbf{J}(\mathbf{x})$ を計算することを n 回と数える. この関数の評価回数は, 関数 $\{f_i(\mathbf{x})\}$ の性質, 初期ベクトル, 収束判定値などに大きく依存する.

一般には、初期ベクトルが良く、収束判定値として標準値を採用した場合は、 $\max = 100 \cdot n \cdot m$ 程度が妥当である。

指定評価回数内で収束条件が満たされず $\text{icon} = 10000$ として戻った場合でも、再度 `c_dnolg1` を呼び出すことにより、継続して反復することができる。しかし、この場合は引数 `max` には、負の値で追加評価回数を指定し、他の引数の内容は保存したままで呼び出すこと。

4. 使用例

$F(x_1, x_2) = f_1^2(x_1, x_2) + f_2^2(x_1, x_2)$, の極小点 $\mathbf{x}^*$ を $\mathbf{x}_0 = (-1.2, 1.0)^T$ から出発して求める。ここで $f_1(x_1, x_2) = 1 - x_1$, $f_2(x_1, x_2) = 10(x_2 - x_1^2)$ である。

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define N 2
#define M 2

void fun(double x[], double y[]); /* user function prototype */
void jac(double x[], double *g, int k); /* derivative prototype */

MAIN__()
{
    int ierr, icon;
    double x[N], f[M], sums, epsr, vw[N+2][M+N];
    int i, n, m, k, max;

    /* initialize data */
    n = N;
    m = M;
    k = M+N;
    x[0] = -1.2;
    x[1] = 1;
    epsr = 1e-3;
    max = 100*n*m;
    /* find minimum of function */
    ierr = c_dnolg1(x, n, fun, jac, m, epsr, &max,
                   f, &sums, (double*)vw, k, &icon);
    if (icon >= 20000) {
        printf("ERROR in c_dnolg1. icon = %i", icon);
        exit(1);
    }
    printf("icon = %i    max = %i    sums = %12.4e\n", icon, max, sums);
    printf("x: ");
    for (i=0; i<n; i++) printf("%12.4e ", x[i]);
    printf("\n");
    printf("f: ");
    for (i=0; i<m; i++) printf("%12.4e ", f[i]);
    printf("\n");
    return(0);
}

/* user function */
void fun(double x[], double y[])
{
    y[0] = 1-x[0];
    y[1] = 10*(x[1]-x[0]*x[0]);
    return;
}

/* derivative function */
void jac(double x[], double *g, int k)
{
    g[0] = -1;
    g[1] = 0;
    g[k] = -20*x[0];
    g[k+1] = 10;
    return;
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の NOLG1 の項目及び[69], [82]を参照のこと.

c_dnrml

実行列の固有ベクトルの正規化

ierr = c_dnrml(ev, k, n, ind, m, mode, &icon);

1. 機能

n 次の実行列の m 個の固有ベクトル $\mathbf{x}_j$ ($j = 1, \dots, m$) が与えられたとき(1)又は(2)の正規化を行い $\mathbf{y}_j$ を求める.

$$\mathbf{y}_j = \mathbf{x}_j / \|\mathbf{x}_j\|_\infty, \quad (1)$$

$$\mathbf{y}_j = \mathbf{x}_j / \|\mathbf{x}_j\|_2. \quad (2)$$

ただし, $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dnrml((double *) ev, k, n, ind, m, mode, &icon);
```

引数の説明:

ev	double	入力	m 個の固有ベクトル $\mathbf{x}_j$, $j = 1, \dots, m$. (“使用上の注意”参照)
	ev[m][k]	出力	正規化された固有ベクトル $\mathbf{y}_j$, $j = 1, \dots, m$.
k	int	入力	配列 ev の整合寸法 ($\geq n$).
n	int	入力	実行列の次数 n .
ind	int ind[m]	入力	ev に格納する各固有ベクトルが, 実固有ベクトルか複素固有ベクトルかの指示. j 行目が, 実固有ベクトルならば ind[j-1] = 1, 複素固有ベクトルの実部ならば ind[j-1] = -1, 虚部ならば ind[j-1] = 0 と与えること.
m	int	入力	配列 ind の大きさ. ($m \leq m \leq n$)
mode	int	入力	正規化の方法の指示 mode = 1 のときは(1)の正規化を行う. mode = 2 のときは(2)の正規化を行う.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	$n = 1$	ev[0][0] = 1.
30000	次のいずれかであった. $m < 1$ 又は $m > n$ $k < n$ - mode $\neq 1$ 又は 2 - ind に誤りがあった	処理を打ち切る.

3. 使用上の注意

a) **ev, ind** 及び **m**

固有ベクトル **ev** への格納は, 実固有ベクトルは 1 行ごとに, 複素固有ベクトルは実部と虚部を対にして 2 行ごとに格納すること.

実対称行列の固有ベクトルの正規化を行う場合は, **ind** の内容をすべて 1 で与えればよい.

なお, **c_dhvec** あるいは **c_dhbk1** 呼び出し後の引数 **ev, ind, m** は本関数でそのまま入力できるようになっている.

4. 使用例

n 次の実行列の固有ベクトルを $\|x\|_\infty = 1$ となるように正規化しなおす.

```
#include <stdio.h>
#include <stdlib.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 5

MAIN__()
{
    int ierr, icon;
    int n, i, j, k, mode, m, ind[NMAX];
    double a[NMAX][NMAX], er[NMAX], ei[NMAX], ev[NMAX][NMAX], vw[NMAX];

    /* initialize matrix */
    n = NMAX;
    k = NMAX;
    for (i=0; i<n; i++) {
        a[i][i] = n-i;
        for (j=0; j<i; j++) {
            a[i][j] = n-i;
            a[j][i] = n-i;
        }
    }
    mode = 0;
    /* find eigenvalues and eigenvectors */
    ierr = c_deig1((double*)a, k, n, mode, er, ei, (double*)ev, vw, &icon);
    if (icon > 10000 ) {
        printf("ERROR: c_deig1 failed with icon = %i\n", icon);
        exit (1);
    }
    /* initialize ind array */
    m = n;
    mode = 1;
    i = 0;
    while (i<m) {
        if (ei[i] == 0) ind[i++] = 1;
        else {
            ind[i++] = -1;
            ind[i++] = 0;
        }
    }
    /* normalize eigenvectors */
    ierr = c_dnrml((double*)ev, k, n, ind, m, mode, &icon);
    if (icon > 10000 ) {
        printf("ERROR: c_dnrml failed with icon = %i\n", icon);
        exit (1);
    }
    printf("icon = %i\n", icon);
    /* print eigenvalues and eigenvectors */
    i = 0;
    k = 0;
    while (i<m) {
        if (ind[i] == 0) i++;
        else if (ei[i] == 0) {
            /* real eigenvector */
            printf("eigenvalue: %12.4f\n", er[i]);
            printf("eigenvector:");
        }
    }
}
```

```
        for (j=0;j<n;j++)
            printf("%7.4f ", ev[k][j]);
        printf("\n");
        i++;
        k++;
    }
    else {
        /* complex eigenvector pair */
        printf("eigenvalue: %7.4f+i*%7.4f\n", er[i], ei[i]);
        printf("eigenvector: ");
        for (j=0;j<n;j++)
            printf("%7.4f+i*%7.4f ", ev[k][j], ev[k+1][j]);
        printf("\n");
        printf("eigenvalue: %7.4f+i*%7.4f\n", er[i+1], ei[i+1]);
        printf("eigenvector: ");
        for (j=0;j<n;j++)
            printf("%7.4f+i*%7.4f ", ev[k][j], -ev[k+1][j]);
        printf("\n");
        i = i+2;
        k = k+2;
    }
}
return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の NRML の項目を参照のこと。

c_dodam

連立 1 階常微分方程式 (アダムス法)

<pre>ierr = c_dodam(&x, y, fun, n, xend, &isw, &epsa, &epsr, vw, ivw, &icon);</pre>

1. 機能

連立 1 階常微分方程式の初期値問題:

$$\mathbf{y}' = \mathbf{f}(x, \mathbf{y}), \mathbf{y}(x_0) = \mathbf{y}_0$$

ベクトルを要素ごとに表わすと,

$$\begin{aligned} y_1' &= f_1(x, y_1, y_2, \dots, y_n), & y_1(x_0) &= y_{10} \\ \vdots & & \vdots & \\ y_n' &= f_n(x, y_1, y_2, \dots, y_n), & y_n(x_0) &= y_{n0} \end{aligned}$$

この問題において,

- 関数系 $\mathbf{f}$.
- 初期値 x_0 及び $\mathbf{y}(x_0) = \mathbf{y}_0$.
- x の最終値 x_e .

が与えられたとき, アダムス法により

$$x_m = x_0 + \sum_{j=1}^m h_j, \quad m = 1, 2, \dots, e$$

における解 $(y_{1m}, y_{2m}, \dots, y_{nm})^T$ を求める. ただし, キザミ h_j は, 求めるべき解が要求精度を満たすように制御される.

本関数は解の出力方法, すなわち利用者プログラムに戻るタイミングとして次の 2 通りを用意しており, 利用者は目的に応じて選ぶことができる.

- 最終値出力: 最終値 x_e における解が求まったときはじめて戻る.
- ステップ出力: 途中の解, すなわち, $x_1, x_2, \dots$ における解が求まるたびにに戻る.

2. 引数

呼出し形式:

```
ierr = c_dodam(&x, y, fun, n, xend, &isw, &epsa, &epsr, vw, ivw, &icon);
```

引数の説明:

x	double	入力	初期値 x_0 .
		出力	最終値 x_e . ただし, ステップ出力が指定されている場合は, 1 ステップ前進したときの独立変数の値 x_m .
y	double y[n]	入力	初期値 $y_{10}, y_{20}, \dots, y_{n0}$ を $y[0], y[1], \dots, y[n-1]$ の順に与える.

		出力	最終値 x_e における解ベクトル. ただし, ステップ出力が指定されている場合は, 1 ステップ前進したときの $x = x_m$ における解ベクトル.												
fun	function	入力	(1)における $f_i; i = 1, 2, \dots, n$ を計算するユーザ関数名. プロトタイプ宣言: <pre>void fun(double x, double y[], double yp[]);</pre> 引数の説明: <table> <tr> <td>x</td><td>double</td><td>入力</td><td>独立変数 x.</td></tr> <tr> <td>y</td><td>double y[n]</td><td>入力</td><td>$y[0] = y_1, y[1] = y_2, \dots,$ $y[n-1] = y_n$ なる対応を持つ 大きさ n の 1 次元配列.</td></tr> <tr> <td>yp</td><td>double yp[n]</td><td>出力</td><td>$yp[0] = f_1(x, y_1, y_2, \dots, y_n),$ $yp[1] = f_2(x, y_1, y_2, \dots, y_n),$ $\dots, yp[n-1] = f_n(x, y_1, y_2,$ $\dots, y_n)$ なる対応を持つ大 きさ n の 1 次元配列.</td></tr> </table>	x	double	入力	独立変数 x .	y	double y[n]	入力	$y[0] = y_1, y[1] = y_2, \dots,$ $y[n-1] = y_n$ なる対応を持つ 大きさ n の 1 次元配列.	yp	double yp[n]	出力	$yp[0] = f_1(x, y_1, y_2, \dots, y_n),$ $yp[1] = f_2(x, y_1, y_2, \dots, y_n),$ $\dots, yp[n-1] = f_n(x, y_1, y_2,$ $\dots, y_n)$ なる対応を持つ大 きさ n の 1 次元配列.
x	double	入力	独立変数 x .												
y	double y[n]	入力	$y[0] = y_1, y[1] = y_2, \dots,$ $y[n-1] = y_n$ なる対応を持つ 大きさ n の 1 次元配列.												
yp	double yp[n]	出力	$yp[0] = f_1(x, y_1, y_2, \dots, y_n),$ $yp[1] = f_2(x, y_1, y_2, \dots, y_n),$ $\dots, yp[n-1] = f_n(x, y_1, y_2,$ $\dots, y_n)$ なる対応を持つ大 きさ n の 1 次元配列.												
n	int	入力	連立 1 階常微分方程式の元数.												
xend	double	入力	独立変数の最終値 x_e . (“使用上の注意” d)参照)												
isw	int	入力	積分する際の条件を指定する. isw は 10 進 3 桁の非負の整数 であり, いまそれを, $isw = 100d_3 + 10d_2 + d_1$ と表すと, 各 d_i は次のように指定する. d_1 初回呼出しであるか否かを指定する. 0 初回呼出しである. 1 継続呼出しである. 初回呼出しとは, 与えられた微分方程式に対して初めて 本関数を呼び出すときを言う. d_2 解の出力方法を指定する. 0 最終値出力 1 ステップ出力 d_3 最終値 x_e を超えた点で, 関数系 $f_1, f_2, \dots, f_n$ が計算で きるか否かを指定する. 0 計算できる. 1 計算できない. $d_3 = 1$ と指定する状況は, 微係数が x_e を超えた点では 定義されないか, あるいは x_e の近くに不連続点を持つ 場合である. このような状況でないとき, むやみに $d_3 =$ 1 とすると計算効率を著しく低下させるので注意する こと.												
		出力	x_e における解, 若しくはステップごとの解を求めて利用者プ ログラムに戻るとき, d_1, d_3 は次のように変更される. d_1 $d_1 = 1$ とする. 本関数を繰り返し呼び出すときは d_1 を 1 のままにしておくこと. 利用者が再び $d_1 = 0$ と設定し直すときは, 別の方程式 を解き始めるときである. d_3 入力時に $d_3 = 1$ であった場合は, x_e における解が求まっ た時点で $d_3 = 0$ とする.												

epsa	double	入力 出力	絶対誤差の上限 (≥ 0.0). 入力時に epsa が小さすぎた場合は, 適当な大きさに変更された値. (“使用上の注意” c)参照).
epsr	double	入力 出力	相対誤差の上限 (≥ 0.0). 入力時に epsr が小さすぎた場合は, 適当な大きさに変更された値. (“使用上の注意” c)参照).
vw	double vw[RelLen]	作業領域	RelLen = 21n+110. 繰り返し本関数を呼び出すときは, 内容を変えてはならない.
ivw	int ivw[IntLen]	作業領域	IntLen = 11. 繰り返し本関数を呼び出すときは, 内容を変えてはならない.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	(ステップ出力の場合だけ) 1 ステップ積分した.	続けて呼び出すことができる.
10	xend における解を求めた.	xend を変えて続けて呼び出すことができる.
100	1 ステップ積分した. しかし xend に到達するのに 500 回を超えるステップが必要となることが判明した.	続けて呼び出せば, ステップ回数のカウンタを 0 にして, 新たに回数を数えながら処理を進める.
200	1 ステップ積分した. しかし解いている方程式が強いスティフ性を持つことが判明した.	続けて呼び出すこともできるが, 効率の点を考えると, スティフな方程式を解く関数を利用した方がよい.
10000	epsr 及び epsa が演算精度に比べ小さすぎた.	epsr 及び epsa を適当に大きくした. 続けて呼び出すことができる. (大きくされた epsr 及び epsa は確認しておくのがよい.)
30000	次のいずれかであった. $n \leq 0$. $x = xend$. - isw の指定に誤りがある. $epsr < 0$, 又は $epsa < 0$. - ivw の内容が変えられている (ただし, 2 回目以降の呼出し時)	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

本関数は, 非スティフ又は弱スティフな微分方程式を解くための標準的なプログラムである. 特に, 以下に示すいずれかの状況にある場合は効率的に解を計算する.

- 関数系 $f_1, f_2, \dots, f_n$ の計算に非常な時間がかかる.
- 高精度な解が要求される.
- 解の数表を作るときなどのように, 独立変数の多くの値に対して解を求める.
- 解の微係数, すなわち $f_1, f_2, \dots, f_n$ が不連続点を持つ.

方程式が強スティフであることがあらかじめわかっているならば, 関数 c_dodge を利用した方がよい.

b) icon の判定

利用者が isw の 10 の位を 0 とすることにより最終値出力を指定した場合は, icon が 10 と出力されたときに限り x_e における解を受け取ることができる. しかし, それ以前に icon が 100, 200, あるいは 10000 として利用者プログラムに戻ることもあるので注意すること.

一方, isw の 10 の位を 1 とすることにより, ステップ出力を指定した場合は, icon が 0 と出力されたときは勿論のこと, 100 あるいは 200 と出力されたときでもステップごとの解を受け取ることができる. そのあといずれ icon = 10 と出力されるが, このときは x_e における解が求まったことを意味する.

c) epsa と epsr

微分方程式の解の成分を $y[L]$, 誤差 (正確には局所誤差) を $l_e[L]$ とすると, 本関数は $L = 0, 1, 2, \dots, n-1$ に対して,

$$|l_e[L]| \leq \text{epsr} \cdot |y[L]| + \text{epsa}$$

を満たすように, 誤差を制御する. この式において, $\text{epsa} = 0.0$ と指定すると, 相対誤差判定となり, $\text{epsr} = 0.0$ と指定すると, 絶対誤差判定となる.

利用者はこれらの判定値を指定するに当たり, 以下のことに注意する必要がある.

- 解の大きさが非常に変化する問題では相対誤差判定が望ましい.
- 解の大きさがさほど変化しないか, あるいは, 小さな解に対しては関心が無い場合には, 絶対誤差判定としてもよい.
- しかし, 安全で無駄のない判定は, $\text{epsa} \neq 0$ かつ $\text{epsr} \neq 0$ として指定する混合判定である. この場合, 大きい解に対しては実質的に相対判定となり, 小さい解に対しては実質的に絶対判定となる.

また, 本関数で達成できる最大の精度を望むときは, epsa , epsr を必要以上に小さく指定すればよい. このとき, 関数内部で, それらの値は適当に大きくされ, 利用者はこのことを, icon = 10000 の出力で知らされる. そのあと, 続けて本関数を呼び出せばよい. なお, $\text{epsa} = \text{epsr} = 0.0$ と入力されたときは, $\text{epsr} = 16\mu$ と変更する. ただし μ は丸め誤差の単位である.

d) 引数 xend

独立変数の連続したいくつかの値に対して解を求めたい場合は, 順番に xend を変えながら本関数を呼び出せばよい. 本関数は, このように繰り返し呼ばれることを想定して, 利用者プログラムに戻るときはすべての引数を次の呼出し時に必要な値に設定している. したがって利用者は, 単に xend を変えて呼び出すだけでよい. あるいは必要ならば epsa , epsr を変えてもよい. それ以外は変えてはならない.

e) 微係数の不連続点及び非定義域の処置

解及び微係数が不連続点を持つ場合には, その点を検出し, それに適応した計算をしない限り, 満足のゆく精度の解は得られない. 本関数は不連続点を自動的に検出し, それに適応した計算を行う能力を持っている. したがって, 利用者は不連続点の存在を意識する必要はない. しかし, もし利用者が不連続点の位置を以下に示す要領で指定するならば, 検出に要する手間を省くことができ, しかもより正確に計算することができる.

- xend を不連続点の位置に置き, 同時に isw の 100 の位を 1 と設定して本関数を呼び出す. isw の下位 2 桁を変えないで, 100 の位を 1 にするには,
$$\text{isw} = \text{isw} \% 100 + 100;$$
により行える.
- 不連続点における解が求まると, 本関数は isw の 100 の位を 0 と設定して利用者プログラムに戻る. xend を適当に前進させて, 続けて本関数を呼び出すときは, isw の 1 の位を 0 に設定しておく. これは例えば,

```
isw = (isw/10)*10;
```

により行える. この isw の設定は, 不連続点における解を新たな初期値として, あたかも別の微分方程式を解き始めることを本関数に指示する意味を持つ.

4. 使用例

次の常微分方程式:

$$\begin{aligned} y_1' &= y_3, & y_1(0) &= 1 \\ y_2' &= y_4, & y_2(0) &= 0 \\ y_3' &= -\frac{y_1}{(y_1^2 + y_2^2)^{3/2}}, & y_3(0) &= 0 \\ y_4' &= -\frac{y_2}{(y_1^2 + y_2^2)^{3/2}}, & y_4(0) &= 1 \end{aligned}$$

を区間 $[0, 2\pi]$ で解く. 解は,

$$x = \frac{2\pi}{64}i, \quad i = 1, 2, \dots, 64$$

の各点で出力するものとする.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define N 4 /* order of system */

/* user function prototypes */
void fun(double x, double y[], double yp[]);

MAIN__()
{
    int ierr, icon;
    int i, n, isw, ivw[11];
    double x, y[N], pi, dx, xend, epsa, epsr, vw[21*N+110];

    x = 0;
    y[0] = 1;
    y[1] = 0;
    y[2] = 0;
    y[3] = 1;
    n = N;
    epsa = 1e-8;
    epsr = 1e-5;
    isw = 0;
    pi = 4*atan(1);
    dx = pi/32;
    printf("    x          y[0]          y[1]          y[2]          y[3]\n");
    for (i=1; i<65; i++) {
        xend = dx*(double)i;
        while(1) {
            /* solve system */
            ierr = c_dodam(&x, y, fun, n, xend, &isw, &epsa, &epsr,
                          vw, ivw, &icon);
            if (icon == 10) break;
            if (icon == 100) printf("too many steps\n");
            if (icon == 200) printf("the equations appear to be stiff\n");
            if (icon == 10000)
                printf("tolerance reset; epsa = %12.4e epsr = %12.4e\n", epsa, epsr);
            if (icon == 30000) {
                printf("invalid input\n");
                exit(1);
            }
        }
        printf("%12.4e %12.4e %12.4e %12.4e %12.4e\n",
              x, y[0], y[1], y[2], y[3]);
    }
}
```

```
    return(0);
}

/* user function */
void fun(double x, double y[], double yp[])
{
    double r3;
    r3 = pow((y[0]*y[0]+y[1]*y[1]),1.5);
    yp[0] = y[2];
    yp[1] = y[3];
    yp[2] = -y[0]/r3;
    yp[3] = -y[1]/r3;
    return;
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の ODAM の項目及び[94]を参照のこと.

c_dodge

スティフ連立 1 階常微分方程式(ギア法)

<pre>ierr = c_dodge(&x, y, fun, n, xend, &isw, epsv, &epsr, mf, &h, jac, vw, ivw, &icon);</pre>

1. 機能

連立 1 階常微分方程式の初期値問題:

$$\mathbf{y}' = \mathbf{f}(x, \mathbf{y}), \quad \mathbf{y}(x_0) = \mathbf{y}_0$$

ベクトルを要素ごとに表わすと,

$$\begin{aligned} y_1' &= f_1(x, y_1, y_2, \dots, y_n), & y_1(x_0) &= y_{10} \\ \vdots & & \vdots & \\ y_n' &= f_n(x, y_1, y_2, \dots, y_n), & y_n(x_0) &= y_{n0} \end{aligned}$$

において,

- 関数系 $\mathbf{f}$.
- 初期値 x_0 及び $\mathbf{y}(x_0) = \mathbf{y}_0$.
- x の最終値 x_e .

が与えられたとき, ギア法又はアダムス法により,

$$x_m = x_0 + \sum_{j=1}^m h_j, \quad m = 1, 2, \dots, e$$

における解 $(y_{1m}, y_{2m}, \dots, y_{nm})^T$ を求める. ただし, キザミ h_j は, 求めるべき解が要求精度を満たすように制御される.

ギア(Gear)法はスティフな方程式に適し, アダムス法は非スティフな方程式に適している. 利用者は, スティフ性の有無に応じて二つの手法を使い分けることができる.

本関数は解の出力方法, すなわち利用者プログラムに戻るタイミングとして以下の 2 通りを用意しており, 利用者は目的に応じて選ぶことができる.

- 最終値出力: 最終値 x_e における解が求まったときはじめて戻る.
- ステップ出力: 途中の解, すなわち, $x_1, x_2, \dots$ における解が求まるたびにに戻る.

2. 引数

呼出し形式:

```
ierr = c_dodge(&x, y, fun, n, xend, &isw, epsv, &epsr, mf, &h, jac, vw, ivw,
               &icon);
```

引数の説明:

x double 入力 初期値 x_0 .

		出力	最終値 x_e . ただし、ステップ出力が指定されている場合は、1 ステップ前進したときの独立変数の値 x_m .
y	double y[n]	入力	初期値 $y_{10}, y_{20}, \dots, y_{n0}$ を $y[0] = y_{10}, y[1] = y_{20}, \dots, y[n-1] = y_{n0}$ の順に与える.
		出力	最終値 x_e における解ベクトル. ただし、ステップ出力が指定されている場合は、1 ステップ前進したときの $x = x_m$ における解ベクトル.
fun	function	入力	(1)における f_i ; $i = 1, 2, \dots, n$ を計算するユーザ関数名. プロトタイプ宣言: void fun(double x, double y[], double yp[]); 引数の説明: x double 入力 独立変数 x . y double 入力 $y[0] = y_1, y[1] = y_2, \dots,$ y[n] $y[n-1] = y_n$ なる対応を持つ 大きさ n の 1 次元配列. yp double 出力 $yp[0] = f_1(x, y_1, y_2, \dots, y_n),$ yp[n] $yp[1] = f_2(x, y_1, y_2, \dots, y_n),$ $\dots, yp[n-1] = f_n(x, y_1, y_2,$ $\dots, y_n)$ なる対応を持つ大き さ n の 1 次元配列.
n	int	入力	連立 1 階常微分方程式の元数.
xend	double	入力	独立変数の最終値 x_e . (“使用上の注意” d) 参照)
isw	int	入力	積分する際の条件を指定する. isw は 10 進 4 桁の非負の整数 であり、いまそれを, $isw = 1000d_4 + 100d_3 + 10d_2 + d_1$ と表すと、各 d_i は次のように指定する. d_1 初回呼出しであるか否かを指定する. 0 初回呼出しである. 1 継続呼出しである. 初回呼出しとは、与えられた微分方程式に対して初めて 本関数を呼び出すときを言う. d_2 解の出力方法を指定する. 0 最終値出力 1 ステップ出力 d_3 最終値 x_e を超えた点で、関数系 $f_1, f_2, \dots, f_n$ が計算で きるか否かを指定する. 0 計算できる. 1 計算できない. $d_3 = 1$ と指定する状況は、微係数が x_e を超えた点では 定義されないか、あるいは、 x_e を超えた近くに不連続 点を持つ場合である. このような状況でないとき、む やみに $d_3 = 1$ と指定すると計算効率を著しく低下さ せるので注意すること. d_4 解法の途中で、手法を切り換えたり (引数 mf), 要求精 度 (引数 epsv, epsr), 若しくは n を変えたりする とき意味を持つ. 0 変えなかった.

			1 変えた. (“使用上の注意” e), f) 参照.)
		出力	x_e における解, 若しくはステップごとの解を求めて利用者プログラムに戻るとき, d_1, d_3, d_4 は次のように変更される. d_1 $d_1 = 1$ とする. 本関数を繰り返し呼び出すときは d_1 を 1 のままにしておくこと. 利用者が再び $d_1 = 0$ と設定し直すときは, 別の方程式を解き始めるときである. d_3 入力時に $d_3 = 1$ であった場合は, x_e における解が求まった時点で $d_3 = 0$ とする. d_4 入力時に $d_4 = 1$ であった場合は, $d_4 = 0$ とする.
epsv	double epsv[n]	入力 出力	絶対誤差の上限. 解の成分ごとに与える. 入力時に epsv が小さすぎた場合は適当な大きさに変更された値. (“使用上の注意” c) 参照)
epsr	double	入力 出力	絶対誤差の上限. 入力時に epsr が小さすぎた場合は適当な大きさに変更された値. (“使用上の注意” c) 参照)
mf	int	入力	ギア法とアダムス法のいずれを使うか, 及び, 修正子の反復法をいずれにするかを指定する. mf は 10 進 2 桁の非負の整数であり, いま, それを $mf = 10meth + iter$ と表わすと, $meth, iter$ は次のように指定する. <i>meth</i> 手法を指定する. 1 ギア法を使用する. 方程式がスティフな場合に適している. 2 アダムス法を使用する. 方程式が非スティフな場合に適している. <i>iter</i> 修正子の反復法を指定する. 0 解析的ヤコビアン行列 $J = (\partial f_i / \partial y_i)$ を使ったニュートン法. このとき利用者は, ヤコビアン行列を計算するための関数を用意する必要がある. (引数“jac”参照) 1 差分法により内部的に近似したヤコビアン行列を使ったニュートン法. 2 $iter = 1$ と同じであるが, この場合, ヤコビアン行列を対角行列で近似する. 3 ヤコビアン行列を使わない反復法 (関数反復). 方程式がスティフなら $iter = 0, 1$, 又は 2 とする. このうち $iter = 0$ が最も望ましいが, 解析的ヤコビアン行列が用意できないなら, $iter = 1$ とすればよい. $iter = 2$ は, ヤコビアン行列が優対角であることが分っている場合に有用である. 方程式が非スティフなら $iter = 3$ とする.
h	double	入力 出力	初回呼出しのとき, 試行すべき初期キザミ幅を指定する ($h \neq 0$). 符号は $x_e - x_0$ と同じであること. h の標準的な値は, $h = \min(10^{-5}, \max(10^{-4} x_0 , x_e - x_0))$ である. h は要求精度を満たすべく制御される. 最後に使われたキザミ幅.

jac	function	入力	次に示すヤコビアン行列を計算するユーザ関数名.
			$J = \begin{bmatrix} \frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial y_2} & \dots & \frac{\partial f_1}{\partial y_n} \\ \frac{\partial f_2}{\partial y_1} & \frac{\partial f_2}{\partial y_2} & \dots & \frac{\partial f_2}{\partial y_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial y_1} & \frac{\partial f_n}{\partial y_2} & \dots & \frac{\partial f_n}{\partial y_n} \end{bmatrix}$
			プロトタイプ宣言:
			void jac(double x, double y[], double pd[], int k);
			引数の説明:
x	double	入力	独立変数 x .
y	double y[n]	入力	$y[0]=y_1, y[1]=y_2, \dots,$ $y[n-1]=y_n$ なる対応を持つ 大きさ n の 1 次元配列.
pd	double pd[k*k]	出力	ヤコビアン行列. pd [(i-1)*k+(j-1)] $= \partial f_i / \partial y_j$ $1 \leq i \leq n$ かつ $1 \leq j \leq n$ である こと.
		k	int 入力 pd を 2 次元配列と考えた ときの整合寸法.
vw	double vw[RelLen]	作業領域	RelLen = n*(n+17)+70. 繰り返し本関数を呼び出すときは内容を変えてはならない.
ivw	int ivw[IntLen]	作業領域	IntLen = n+25. 繰り返し本関数を呼び出すときは内容を変えてはならない.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	(ステップ出力の場合だけ) 1 ステップ積分した.	続けて呼び出すことができる. (“使用上の注意” b) 参照)
10	xend における解を求めた.	xend を変えて続けて呼び出すことができる. (“使用上の注意” d) 参照)
10000	epsr 及び epsv[l] : l=0, 1, 2, ... n-1 が演算精度に比べて小さすぎた.	epsr 及び epsv[l] : l=0, 1, 2, ... n-1 を適当に大きくした. (大きくされた epsr 及び epsv[l] は確認しておくのがよい.)
15000	初期キザミ幅の 10^{-10} 倍のキザミ幅を使っても要求精度が得られなかった.	
16000	初期キザミ幅の 10^{-10} 倍のキザミ幅を使っても修正子を求めるための反復計算が収束しなかった.	引数 mf により指定されている手法, 若しくは修正子反復法が与えられた方程式にとって適していない可能性が強い. mf を変えて呼び出すとよい.
30000	次のいずれかであった. - n ≤ 0. - x = xend. - isw の指定に誤りがある. - epsr < 0, 又は epsv[l] < 0 なる l がある.	処理を打ち切る.

コード	意味	処理内容
	• $(x_{\text{end}} - x) \cdot h \leq 0$. • <code>ivw</code> の内容が変えられている(ただし 2 回目以降の呼出し時).	

3. 使用上の注意

a) 本関数の使用方法

`c_dodge` は、方程式がスティフであるか、あるいは、解こうとしている区間で非スティフな状態からスティフな状態に変化する場合に有効に利用することができる。しかし、もし方程式が非スティフならば、`c_dodam` を使用した方が効率の面で良い。

b) `icon` 判定

利用者が `isw` の 10 の位を 0 にすることにより最終値出力を指定した場合は、`icon` が 10 と出力されたときに限り x_e における解を受け取ることができる。

一方、`isw` の 10 の位を 1 とすることによりステップ出力を指定した場合は、`icon` が 0 と出力されたときに解を受け取ることができる。この過程を続けて `icon` が 10 と出力されたときは、 x_e における解が求まったことを意味する。

c) 引数 `epsv` と `epsr`

微分方程式の解の成分を $y[L]$ 、誤差(正確には局所誤差)を $l_e[L]$ とすると、`c_dodge` は、 $L = 0, 1, 2, \dots, n-1$ に対して、

$$|l_e[L]| \leq \text{epsr} \cdot |y[L]| + \text{epsv}[L]$$

を満たすように、誤差を制御する。この式において、 $\text{epsv}[L] = 0$ とすれば、相対誤差判定となり、 $\text{epsr} = 0$ とすれば、解の成分ごとに判定値が異なる絶対誤差判定となる。利用者は、 $\text{epsv}[L]$ 及び epsr を指定するに当り、以下のことに注意する必要がある。

- 解の大きさが非常に変化する問題では、相対誤差判定が望ましい。
- 解の大きさがさほど変化しないか、あるいは、小さな解に対しては関心が無い場合には、絶対誤差判定としてもよい。
- 安全で無駄のない判定は、 $\text{epsv}[L] \neq 0$ 、 $L = 0, 1, 2, \dots, n-1$ 及び $\text{epsr} \neq 0$ とする混合判定である。この場合、大きい解に対しては実質的に相対判定となり、小さい解に対しては実質的に絶対判定となる。方程式がスティフなときは一般に、解の各成分の間で大きさがまちまちになるので、 $\text{epsv}[L]$ もそれに応じて成分ごとに異なる値を与えた方がよい。

なお、入力時に、 $\text{epsr} = 0$ でかつ $\text{epsv}[L]$ のある要素が 0 であったときは、 $\text{epsr} = 16\mu$ と変更する。ただし μ は丸め誤差の単位である。

d) `xend`

独立変数の連続したいくつかの値に対して解を求めたい場合は、順番に `xend` を変えながら本関数を呼び出せばよい。本関数は、このように繰り返すことを想定して、利用者プログラムに戻るときはすべての引数を次の呼出し時に必要な値に設定している。したがって利用者は、単に `xend` を変えて呼び出すだけでよい。

e) 解法の途中における引数 `mf` の切換え

方程式が、解こうとしている区間で、非スティフな状態からスティフな状態へ変化する場合は、スティフ性が強くなってきた時点で、`mf` を 23 から 10 (あるいは 11,12) へ切り換えるのがよい。そのときの手順は以下のとおりである。

- ① isw の 1000 の位, つまり d_4 を 1 とする. これは, $\text{isw} = \text{isw} + 1000$ なる代入文で行える.
- ② mf を切り換える.
- ③ xend を次に望む出力点に進めて, 本関数を呼び出す.

本関数は, ③のあと利用者プログラムに戻るとき, 利用者の要求どおりに手法を切り換えたことを通知する意味で,

```
isw = isw%1000;
```

により, isw の 1000 の位をクリアする.

ただし, xend における解を手法を切り換えるまでもなく少ない手間で計算できるときは, その方法で計算し, 利用者プログラムに戻る. (この場合, isw の 1000 の位はクリアされず, mf の値も呼び出す前と同じである). 本関数は xend が, 手法を切り換えるのに十分な位置まで進められたとき, はじめて手法を切り換える.

f) 解法の途中における引数 n, epsv 及び epsr の変更

利用者は解法の途中で, 引数 n, epsv 及び epsr の値を変更することができる. ただし, n の変更には, 以下に述べるように, 解の挙動に関する十分な知識を必要とする. ここでは, n の変更の意義とその方法について述べる.

方程式がスティフである場合, 積分が進むにつれ, 解の成分のいくつかは他の成分に比べてさほど変化しないか, あるいは, 無視できるほど小さくなることもある. このとき, もし利用者が, それらの成分について興味がないならば, 以降それらの成分を定数とみなし, 残りの成分だけを積分することにすれば, 多少厳密さを失うが, 少ない計算量で必要な解を得ることができる. 引数 n の変更は, このように, 積分する解の成分を減らすことを意味する. ただし, 本関数は, 成分を減らす場合, 方程式系(1)の最後の成分のいくつかを減らす. したがって利用者は, 必要に応じて成分の順序を並べ替えておく必要がある.

以上を前提として, n の縮小は次のように行う.

いま, 本関数の初回呼出しのときの n の値を n_0 とし, これを解法途中で $n_c (< n_0)$ に変更したとすると n_0 元の連立一次方程式の最後の $(n_0 - n_c)$ 個の方程式を削除し, 縮小された n_c 個の方程式からなる系を解く. その際, 除かれた方程式に対応する解の成分 $y[L-1]; L = n_c + 1, n_c + 2, \dots, n_0$ は, 縮小された系において一定に保たれる.

利用者の用意する関数 fun, jac においては, n_c 個の方程式についての微分系又はヤコビアン行列を計算するだけでよい. これらの関数内で n の変更を確認するには, 引数 n を外部変数として宣言することにより, 利用者のプログラムと連絡を図ればよい.

以上が n の変更についてのあらましであるが, epsv 及び epsr も必要なら任意な時点で変更することができる.

これらの引数の変更は前項で述べた mf の切り換えと同じ手順で行うこと.

4. 使用例

連立一階常微分方程式:

$$\begin{aligned} y_1' &= y_2, & y_1(0) &= 1 \\ y_2' &= -11y_2 - 10y_1, & y_2(0) &= -1 \end{aligned}$$

を区間 [0,100] で解く. この方程式はスティフである. 解は $x = 10^{-3+i}$, $i = 1, 2, \dots, 5$ の各点で出力する.

```
#include <stdlib.h>
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

#define N 2 /* order of system */
```

```

/* user function prototypes */
void fun(double x, double y[], double yp[]);
void jac(double x, double y[], double *pd, int k);

MAIN__()
{
    int ierr, icon;
    int i, n, isw, mf, ivw[N+25];
    double x, y[N], xend, epsv[N], epsr, h, vw[N*(N+17)+70];

    mf = 10;
    n = N;
    x = 0;
    h = 1.0e-5;
    y[0] = 1;
    y[1] = -1;
    isw = 0;
    epsv[0] = 0;
    epsv[1] = 0;
    epsr = 1.0e-6;
    xend = 1.0e-3;
    printf("      x          y[0]          y[1]\n");
    for (i=0;i<5;i++) {
        xend = xend*10;
        while(1) {
            /* solve system */
            ierr = c_dodge(&x, y, fun, n, xend, &isw, epsv, &epsr,
                          mf, &h, jac, vw, ivw, &icon);
            if (icon == 10) break;
            if (icon == 16000){
                printf("ERROR: no convergence\n");
                exit(1);
            }
            if (icon == 10000 || icon == 15000){
                /* repeat with new tolerances */
                printf("WARNING: tolerance reset\n");
                printf("epsr = %12.4e epsv[0] = %12.4e epsv[1] = %12.4e\n",
                      epsr, epsv[0], epsv[1]);
            }
        }
        printf("%12.4e %12.4e %12.4e\n", x, y[0], y[1]);
    }
    return(0);
}

/* user function */
void fun(double x, double y[], double yp[])
{
    yp[0] = y[1];
    yp[1] = -11*y[1]-10*y[0];
    return;
}

/* user Jacobian function */
void jac(double x, double y[], double *pd, int k)
{
    pd[0] = 0; /* [0][0] */
    pd[1] = 1; /* [0][1] */
    pd[2] = -10; /* [1][0] */
    pd[3] = -11; /* [1][1] */
    return;
}

```

5. 手法概要

本関数は、キザミ幅制御及び次数制御付きのギア法とアダムス法を適用している。計算方法の詳細については“富士通 SSL II 使用手引書”の ODGE の項及び[19], [39], [55]を参照のこと。

c_dodrk1

連立 1 階常微分方程式(ルンゲ・クッタ・ヴァーナー法)

ierr = c_dodrk1(&x, y, fun, n, xend, &isw, epsa, &epsr, vw, ivw, &icon);

1. 機能

連立 1 階常微分方程式の初期値問題:

$$\mathbf{y}' = \mathbf{f}(x, \mathbf{y}), \quad \mathbf{y}(x_0) = \mathbf{y}_0, \quad (1)$$

ベクトルを要素ごとにあらわすと,

$$\begin{aligned} y'_1 &= f_1(x, y_1, y_2, \dots, y_n), & y_1(x_0) &= y_{10} \\ \vdots & & \vdots & \\ y'_n &= f_n(x, y_1, y_2, \dots, y_n), & y_n(x_0) &= y_{n0} \end{aligned}$$

において,

- 関数系 $\mathbf{f}$
- 初期値 x_0 及び $\mathbf{y}(x_0) = \mathbf{y}_0$
- 最終値 x_e

が与えられたとき, ルンゲ・クッタ・ヴァーナー (Runge-Kutta-Verner) 法により,

$$x_m = x_0 + \sum_{j=1}^m h_j, \quad m = 1, 2, \dots, e$$

における解 $(y_{1m}, y_{2m}, \dots, y_{nm})^T$ を求める. ただし, キザミ h_j は, 求めるべき解が要求精度を満たすように制御される.

本関数は解の出力方法, すなわち利用者プログラムに戻るタイミングとして以下の 2 通りを用意しており, 利用者は目的に応じて選ぶことができる.

1. 最終値出力: 最終値 x_e における解が求まったときはじめて戻る..
2. ステップ出力: 途中の解, すなわち, $x_1, x_2, \dots, x_e$ における解が求まるたびにに戻る.

2. 引数

呼出し形式:

```
ierr = c_dodrk1(&x, y, fun, n, xend, &isw, epsa, &epsr, vw, ivw, &icon);
```

引数の説明:

x	double	入力	初期値 x_0 .
		出力	最終値 x_e . ただし, ステップ出力が指定されている場合は, 1 ステップ前進したときの独立変数の値 x_m .
y	double y[n]	入力	初期値 $y_{10}, y_{20}, \dots, y_{n0}$ を $y[0] = y_{10}, y[1] = y_{20}, \dots, y[n-1] = y_{n0}$ の順に与える.
		出力	最終値 x_e における解ベクトル. ただし, ステップ出力が指定されている場合は, 1 ステップ前進したときの $x = x_m$ における解ベクトル.
fun	function	入力	(1)における $\mathbf{f}$ を計算する関数. プロトタイプ宣言: <code>void fun(double x, double y[], double yp[]);</code> 引数の説明:

			x	double	入力	独立変数 x .
			y	double	入力	$y[i-1] = y_i, i=1,2,\dots,n$ なる対応を持つ 1 次元配列.
			yp	double	出力	$yp[i-1] = f_i(x, y_1, y_2, \dots, y_n)$ $i=1,2,\dots,n$ なる対応を持つ 1 次元配列.
n	int	入力				連立 1 階常微分方程式の元数.
xend	double	入力				独立変数の最終値 x_e .
isw	int	入力				積分する際の条件を指定する. isw は 10 進 2 桁の非負の整数であり, いまそれを $isw=10d_2 + d_1$ と表すと, 各 d_i は次のように指定する. d_1 初回呼出しであるか否かを指定する. 0 初回呼出しである. 1 継続呼出しである. 初回呼出しとは, 与えられた微分方程式に対して初めて本関数を呼び出すときを言う. d_2 解の出力方法を指定する. 0 最終値出力. 1 ステップ出力.
		出力				x_e における解, もしくはステップごとの解を求めて利用者プログラムに戻るとき, $d_1 = 1$ と変更される. 本関数を繰返し呼び出すときは, d_1 を 1 のままにしておくこと. 利用者が再び $d_1 = 0$ と設定し直すときは, 別の方程式を解き始めるときである.
epsa	double	入力				絶対誤差の上限. (“使用上の注意”c)参照)
epsr	double	入力				相対誤差の上限. (“使用上の注意”c)参照)
		出力				入力時に epsr が小さすぎた場合は, 適当な大きさに変更された値.
vw	double	作業領域				繰返し本関数を呼び出すときは内容を変えてはならない.
	vw[9n+40]					
ivw	int	作業領域				繰返し本関数を呼び出すときは内容を変えてはならない.
icon	int	出力				コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	(ステップ出力の場合) 1 ステップ積分した.	正常. 続けて呼び出すことができる.
10	xend における解を求めた.	正常. xend を変えて続けて呼び出すことができる.
10000	epsr が使用計算機の演算精度に比べて小さ過ぎたので, 適当に大きくした. (“使用上の注意”c)参照)	1 ステップ進める前に利用者プログラムに戻る. 続けて呼び出すことができる.
11000	微係数の計算回数が 4000 回を越えても xend に到達できないことが判明した.	1 ステップ進める前に利用者プログラムに戻る. 続けて呼び出せば, 微係数の計算回数カウンタを 0 にして, 新たに回数を数えながら処理を進める.

コード	意 味	処 理 内 容
15000	本関数で定める最小キザミを用いても、要求精度が得られないことが判明した。(“使用上の注意”d)参照)	1 ステップ進める前に利用者プログラムに戻る。epsa 又は epsr を増せば続けて呼び出すことができる。
16000	(epsa = 0 のとき) 解のある成分が 0 になり、相対誤差判定が不可能になった。	1 ステップ進める前に利用者プログラムに戻る。epsa を 0 以外の値に変更すれば、続けて呼び出すことができる。
30000	次のいずれかであった。 • $n \leq 0$ • $x = x_{\text{end}}$ • isw の与え方に誤りがあった。 • $\text{epsa} < 0$ 又は $\text{epsr} < 0$ • $\text{icon} = 15000$ 又は 16000、と出力された後、epsa 又は epsr を何ら変更せずに呼び出した。	処理を打ち切る。

3. 使用上の注意

a) 本関数の使用方法

本関数は、関数評価が比較的安価な非スティフ (non-stiff) 又は弱スティフな微分方程式を解くために利用できる。しかし、高精度が要求される場合には利用すべきではない。

b) icon の判定

利用者は icon が 0 又は 10 と出力されたときだけ解を得ることができる。

icon = 10000 ~ 11000 は、その事象が判明したとき 1 ステップ進める前に利用者プログラムに戻る。利用者は、この事象を確認した上で、続けて本関数を呼び出すことができる。

icon = 15000 ~ 16000 は、その事象が判明したとき、1 ステップ進める前に利用者プログラムに戻るが、このとき利用者は、epsa 又は epsr を変更した上で、続けて本関数を呼び出すことができる。

c) epsr

相対誤差に対する上限 epsr は、

$$\text{epsr} \geq \varepsilon_{r\min} = 10^{-12} + 2\mu, \quad (2)$$

を満足するように与えなければならない。ここで μ は丸め誤差の単位である。epsr が上式を満足しない場合には、本関数は icon = 10000 と出力すると共に、 $\text{epsr} = \varepsilon_{r\min}$ と置いて利用者プログラムに戻る。利用者がさらに計算の続行を希望する場合には続けて本関数を呼び出せばよい。

d) 最小キザミ

本関数では、最小キザミ $h_{\min}$ を

$$h_{\min} = 26\mu \cdot \max(|x|, |d|),$$

を満足するように定めている。ここで x は独立変数、また、 $d = (x_e - x_0)/100$ である。もし、最小キザミを用いても、なお要求精度が得られない場合、本関数は icon = 15000 として利用者プログラムに戻る。利用者がさらに計算の続行を希望する場合には、epsa 又は epsr を適当に増した後、再び本関数を呼び出せばよい。

4. 使用例

連立 1 次階常微分方程式

$$\begin{cases} y_1' = y_1^2 y_2 & , y_1(0) = 1.0 \\ y_2' = -1/y_1 & , y_2(0) = 1.0 \end{cases}$$

を $x_0 = 0.0$ から $x_e = 4.0$. まで積分する. 解は各ステップ毎に印刷するものとする.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define N 2 /* order of system */

/* user function prototypes */
void fun(double x, double y[], double yp[]);

MAIN__()
{
    int ierr, icon;
    int n, isw, ivw[5];
    double x, y[N], xend, epsa, epsr, vw[9*N+40];

    x = 0;
    y[0] = 1;
    y[1] = 1;
    n = N;
    xend = 4;
    epsa = 0;
    epsr = 1e-5;
    isw = 10;
    while(1) {
        /* solve system */
        ierr = c_dodrk1(&x, y, fun, n, xend, &isw, epsa, &epsr,
                       vw, ivw, &icon);
        if (icon == 0 || icon == 10) {
            printf("x = %12.4e y[0] = %12.4e y[1] = %12.4e \n",
                   x, y[0], y[1]);
            if (icon == 10) break;
        }
        else if (icon == 10000)
            printf("relative error tolerance too small\n");
        else if (icon == 11000)
            printf("too many steps\n");
        else if (icon == 15000) {
            printf("tolerance reset\n");
            epsr = 10*epsr;
        }
        else if (icon == 16000) {
            printf("tolerance reset\n");
            epsa = 1e-5;
        }
        else if (icon == 30000) {
            printf("invalid input\n");
            exit(1);
        }
    }
    return(0);
}

/* user function */
void fun(double x, double y[], double yp[])
{
    yp[0] = y[0]*y[0]*y[1];
    yp[1] = -1/y[0];
    return;
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の ODRK1 の項目及び[57], [114]を参照のこと.

c_drjetr

実係数高次代数方程式 (ジェンキンス・トラウプの方法)

ierr = c_drjetr(a, &n, z, vw, &icon);

1. 機能

実係数高次代数方程式

$$a_0x^n + a_1x^{n-1} + \cdots + a_n = 0 \quad (a_i: \text{実数}, a_0 \neq 0, n \geq 1)$$

の根をジェンキンス・トラウプの3段アルゴリズムにより求める。

2. 引数

呼出し形式:

```
ierr = c_drjetr(a, &n, z, vw, &icon);
```

引数の説明:

a	double a[n+1]	入力	代数方程式の係数. $a[i]=a_i$ ($i=0, \dots, n$). 演算後内容は保存されない.
n	int	入力	代数方程式の次数 n .
z	dcomplex z[n]	出力	求まった根の個数. (“使用上の注意” a)参照) n 個の根. 根は求まった順に $z[0], \dots, z[n-1]$ に出力される.
vw	double vw[Vwlen]	作業領域	$Vwlen = 6 * (n+1)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	n 根を, すべては求めつくすことができなかった.	求まった根の個数を引数 n に, そして, 根を配列 z に出力する.
30000	次のいずれかであった. $n < 1$ $a_0 = 0$	処理を打ち切る.

3. 使用上の注意

a) 求まった根について

n 次代数方程式は n 個の根を持つが, ごくまれに, すべての根を求めることができないことがある. 利用者はこのことに注意し, 引数 icon あるいは n の演算後の値を確認すること.

4. 使用例

3 次代数方程式 $x^3 - 6x^2 + 11x - 6 = 0$ の根を求める.


```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

#define N 3

MAIN__()
{
    int ierr, icon;
    dcomplex z[N];
    double a[N+1], vw[6*(N+1)];
    int n, i;

    /* initialize data */
    n = N;
    a[0] = 1;
    a[1] = -6;
    a[2] = 11;
    a[3] = -6;
    /* find roots of polynomial */
    ierr = c_drjetr(a, &n, z, vw, &icon);
    printf("icon = %i    n = %i\n", icon, n);
    for (i=0; i<n; i++)
        printf("z[%i] = {%12.4e, %12.4e}\n", i, z[i].re, z[i].im);
    printf("exact roots are: {1, 0}, {2, 0} and {3, 0}\n");
    return(0);
}
```

5. 手法概要

本関数は、ジェンキンス (M.A.Jenkins) とトラウブ (J.F.Traub) による 3 段アルゴリズムを用いている。

計算方法の詳細については“富士通 SSL II 使用手引書”の RJETR の項目及び[59], [60]を参照のこと。

c_drqdr

実係数 2 次方程式
ierr = c_drqdr(a0, a1, a2, z, &icon);

1. 機能

実係数 2 次方程式の根を求める。

$$a_0x^2 + a_1x + a_2 = 0 \quad (a_0 \neq 0) \tag{1}$$

2. 引数

呼出し形式:

```
ierr = c_drqdr(a0, a1, a2, z, &icon);
```

引数の説明:

a0	double	入力	2 次方程式の係数 a_0 .
a1	double	入力	2 次方程式の係数 a_1 .
a2	double	入力	2 次方程式の係数 a_2 .
z	dcomplex	出力	2 次方程式の根.
	z[2]		
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
10000	$a_0 = 0$ であった.	$z[0].re = -a_2/a_1, z[0].im = 0.0$ とする. $z[1]$ は, 保証しない.
30000	$a_0 = 0$ かつ $a_1 = 0$ であった.	処理を打ち切る.

3. 使用例

2 次方程式 $x^2 - 5x + 6 = 0$ の根を求める.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
    dcomplex z[2];
    double a0, a1, a2;

    /* initialize data */
    a0 = 1;
    a1 = -5;
    a2 = 6;
    /* find roots of quadratic */
    ierr = c_drqdr(a0, a1, a2, z, &icon);
    printf("icon = %i   z[0] = {%12.4e, %12.4e}   z[1] = {%12.4e, %12.4e}\n",
           icon, z[0].re, z[0].im, z[1].re, z[1].im);
    printf("exact roots are: {3, 0} and {2, 0}\n");
}
```

```
    return(0);  
}
```

4. 手法概要

2 次方程式 (1) の根は, 根の公式を用いて求める.

計算方法の詳細については “富士通 SSL II 使用手引書” の RQDR の項目及び [16] を参照のこと.

c_dsbmdm

実対称バンド行列の MDM^T 分解(ブロック対角ピボッティング手法)

```
ierr = c_dsbmdm(a, n, &nh, mh, epsz, ip, ivw,
                &icon);
```

1. 機能

$n \times n$, バンド幅 h の正則な実対称バンド行列 $\mathbf{A}$ をブロック対角ピボッティング手法（同名手法にはクラウト法とガウス消去法の関係に似た二つの考え方があり, ここでは後者）により MDM^T 分解する.

$$\mathbf{PAP}^T = \mathbf{MDM}^T \quad (1)$$

ただし, $\mathbf{P}$ はピボッティングによる行の入換えを行う置換行列, $\mathbf{M}$ はバンド幅 $\tilde{h}$ の単位下バンド行列, $\mathbf{D}$ は高々次数 2 の対称なブロックから成る対称ブロック対角行列で $d_{k+1, k} \neq 0$ なら $m_{k+1, k}=0$ である. ただし, $\mathbf{M} = (m_{ij})$, $\mathbf{D} = (d_{ij})$ であること.

2. 引数

呼出し形式:

```
ierr = c_dsbmdm(a, n, &nh, mh, epsz, ip, ivw, &icon);
```

引数の説明:

a	double a[Alen]	入力	行列 $\mathbf{A}$. $\mathbf{A}$ をバンド幅 h_m の行列とみなし対称バンド行列用圧縮格納法で与える. 格納方法の詳細については“1.3.4 バンド行列の格納方法”を参照のこと. (“使用上の注意”a)参照) $Alen = n(h_m + 1) - h_m(h_m + 1) / 2$.
		出力	行列 $\mathbf{D} + (\mathbf{M} - \mathbf{I})$.
n	int	入力	行列 $\mathbf{A}$ の次数 n .
nh	int	入力	行列 $\mathbf{A}$ のバンド幅 h .
		出力	行列 $\mathbf{M}$ のバンド幅 $\tilde{h}$. (“使用上の注意”a)参照)
mh	int	入力	最大許容バンド幅 h_m ($n > mh \geq nh$). (“使用上の注意”a)参照)
epsz	double	入力	ピボットの相対零判定値 (≥ 0). 0.0 のときは標準値が採用される. (“使用上の注意”b)参照)
ip	int ip[n]	出力	ピボッティングによる行の入換えの履歴を示すトランスポジションベクトル. (“使用上の注意”c)参照)
ivw	int ivw[n]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
20000	ピボットが相対的に零となった. 係数行列は非正則の可能性が強い.	処理を打ち切る.
25000	演算中にバンド幅が許容最大値を超えた.	処理を打ち切る.
30000	次のいずれかであった. • $nh < 0$ • $mh < nh$ • $mh \geq n$ • $epsz < 0$	処理を打ち切る.

3. 使用上の注意

a) バンド幅

ピボッティングのための行と列の交換により, 行列のバンド幅は一般に増大する. よって, 行列 **A** は適当に零要素を補って, 実際のバンド幅 h よりも大きなバンド幅 h_m を持ったバンド行列として入力すること. もしも, 分解の途中でバンド幅が h_m を超える場合, $icon = 25000$ として処理が打ち切られる. nh の出力の値は正にこの h_m としての必要十分な値に外ならない.

b) epsz

ピボットの相対零判定値 $epsz$ に 10^{-5} を設定したとすると, この値は次の意味を持っている. すなわち, ブロック対角ピボッティング手法による MDM^T 分解の過程でピボットの値 (1×1 若しくは, 2×2 ピボットが作る小行列式) が, 10 進数 s 桁以上の桁落ちを生じた場合に, そのピボットの値を相対的に零とみなし, $icon = 20000$ として処理を打ち切る. $epsz$ の標準値は丸め誤差の単位を μ としたとき 16μ である. なお, ピボットが小さくなくても計算を続行させる場合には, $epsz$ に極小の値を与えればよいが, その結果は, 保証されない.

c) ip

トランスポジションベクトルとは, ブロック対角ピボッティング手法による MDM^T 分解

$$PAP^T = MDM^T$$

における置換行列 **P** に相当するものである.

本関数では, ピボッティングに伴い配列 **a** の内容を実際に交換しており, その履歴を **ip** に格納している. しかし, 1×1 ピボットと 2×2 ピボットの場合では, **ip** の格納法が異なるので注意すること. 分解の k 段階目における格納方法は次のとおりである.

1×1 ピボットのときは, 交換は行われず, k を $ip[k-1]$ に格納する. 2×2 ピボットのときは, $-k$ を $ip[k-1]$ に, 第 $k+1$ 行 (及び列) と交換される行 (及び列) の番号に負号をつけて $ip[k-1]$ に格納する.

d) 連立 1 次方程式の解

本関数に続けて, 関数 `c_dbmdmx` を呼び出すことにより連立 1 次方程式を解くことができる. しかし, 通常は関数 `c_dlsbix` を呼び出せば, 一度に解が求められる.

e) 固有値の数

行列 **A** の正, 負固有値の数を各々求めることができる. (“使用例”参照)

f) 行列式

行列 **A** の行列式は, 演算後の **D** の行列式に等しい. (“使用例”参照)

4. 使用例

$n \times n$, バンド幅 h の実対称バンド行列の正と負の固有値の数及び行列式の値を求める。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define max(a,b) ((a) > (b) ? (a) : (b))
#define min(a,b) ((a) < (b) ? (a) : (b))

#define NMAX 100
#define NHMAX 50

MAIN__()
{
    int ierr, icon;
    int n, nh, mh, i, j, ij, jj, jmin, peig, neig;
    double epsz, det;
    double a[NMAX*(NHMAX+1)-NHMAX*(NHMAX+1)/2];
    int ivw[NMAX], ip[NMAX];

    /* initialize matrix */
    n = NMAX;
    nh = 2;
    mh = NHMAX;
    ij = 0;
    for (i=0; i<n; i++) {
        jmin = max(i-mh, 0);
        for (j=jmin; j<=i; j++)
            if (i-j == 0)
                a[ij++] = 10;
            else if (i-j == 1)
                a[ij++] = -3;
            else if (i-j == 2)
                a[ij++] = -6;
            else
                a[ij++] = 0;
    }
    epsz = 1e-6;
    /* MDM decomposition of system */
    ierr = c_dsbdm(a, n, &nh, mh, epsz, ip, ivw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dsbdm failed with icon = %d\n", icon);
        exit(1);
    }
    /* find number of positive and negative eigenvalues */
    peig = 0;
    neig = 0;
    i = 1;
    j = 1;
    while (j<=n) {
        if (ip[j-1] != j) {
            peig++;
            neig++;
            i = min(mh, j)+min(mh, j+1)+2+i;
            j = j+2;
        }
        else {
            if (a[i-1] > 0) peig++;
            else if (a[i-1] < 0) neig++;
            i = min(mh, j)+1+i;
            j++;
        }
    }
    printf("Positive e-values: %i\n", peig);
    printf("Negative e-values: %i\n", neig);
    /* calculate determinant */
    det = 1;
    i = 1;
    j = 1;
    while (i<=n) {
        if (ivw[i-1] == i) {
            det = det*a[j-1];
            j = min(mh, i)+1+j;
        }
    }
}
```

```
        i++;
    }
    else {
        jj = min(mh, i)+1+j;
        det = det*(a[j-1]*a[jj-1]-a[jj-2]*a[jj-2]);
        j = min(mh, i+1)+1+jj;
        i = i+2;
    }
}
printf("Determinant: %12.5e\n", det);
return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の SBMDM の項目及び[15]を参照のこと。

c_dseig1

実対称行列の固有値及び固有ベクトル(QL 法)
<pre>ierr = c_dseig1(a, n, e, ev, k, &m, vw, &icon);</pre>

1. 機能

n 次の実対称行列 A の固有値及び固有ベクトルを, QL 法により求める. 固有ベクトルは, $\|x\|_2 = 1$ となるように正規化される. $n \geq 1$ なること.

2. 引数

呼出し形式:

```
ierr = c_dseig1(a, n, e, (double *)ev, k, &m, vw, &icon);
```

引数の説明:

a	double a[<i>Alen</i>]	入力	実対称行列 A . 対称行列用圧縮格納法. (“1.3.2 対称行列の格納方法”参照) $Alen = n(n+1)/2$ 演算後, 内容は保存されない.
n	int	入力	行列 A の次数 n .
e	double e[n]	出力	固有値.
ev	double ev[n][k]	出力	固有ベクトル. 固有ベクトルは行方向に格納される.
k	int	入力	配列 ev の整合寸法 ($k \geq n$).
m	int	出力	求まった固有値・固有ベクトルの数.
vw	double vw[2n]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	$n=1$ であった.	$e[0] = a[0][0]$ $ev[0][0] = 1$ とする.
15000	固有値・固有ベクトルのすべては求めつく すことができなかった.	m に求まっている固有値・固有ベクトルの数 をセットして処理を打ち切る.
20000	固有値・固有ベクトルは 1 つも求められな かった.	$m=0$ とする.
30000	次のいずれかであった. $n < 1$ $k < n$	処理を打ち切る.

3. 使用上の注意

a) 固有値及び固有ベクトルの格納順序

固有値及び固有ベクトルは固有値の求まった順に格納される。

b) 引数 m

引数 m には, icon = 0 のとき n が, また, icon = 15000 のとき, 求まった固有値及び固有ベクトルの数がセットされている。

4. 使用例

対称行列用圧縮格納法で格納された 5×5 行列の固有値及び固有ベクトルを求める。

```
#include <stdlib.h>
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 5

MAIN__()
{
    int ierr, icon;
    int n, m, i, j, k;
    double a[NMAX*(NMAX+1)/2], e[NMAX], ev[NMAX][NMAX], vw[2*NMAX];

    /* initialize matrix */
    n = NMAX;
    k = 0;
    for (i=0;i<n;i++)
        for (j=0;j<=i;j++) {
            a[k] = n-i;
            k = k+1;
        }
    k = NMAX;
    /* find eigenvalues and eigenvectors */
    ierr = c_dseig1(a, n, e, (double*)ev, k, &m, vw, &icon);
    if (icon == 10000 || icon == 30000) {
        printf("ERROR: c_dseig1 failed with icon = %d\n", icon);
        exit(1);
    }
    /* print eigenvalues and eigenvectors */
    for (i=0;i<m;i++) {
        printf("e-value %d: %10.4f\n", i+1, e[i]);
        printf("e-vector:");
        for (j=0;j<n;j++)
            printf("%7.4f ", ev[i][j]);
        printf("\n");
    }
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の SEIG1 の項目及び[118], [119], [127]を参照のこと。

c_dsfri

正弦フレネル積分 $S(x)$

<code>ierr = c_dsfri(x, &sf, &icon);</code>

1. 機能

正弦フレネル積分 $S(x)$ の値,

$$S(x) = \frac{1}{\sqrt{2\pi}} \int_0^x \frac{\sin(t)}{\sqrt{t}} dt = \int_0^{\sqrt{\frac{2}{\pi}}x} \sin\left(\frac{\pi}{2}t^2\right) dt,$$

を多項式及び漸近形式の近似式を用いて計算する. ただし, $x \geq 0$ であること.

2. 引数

呼出し形式:

```
ierr = c_dsfri(x, &sf, &icon);
```

引数の説明:

x	double	入力	独立変数 x . (“使用上の注意”参照)
sf	double	出力	関数値 $S(x)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
20000	$x \geq t_{\max}$ であった.	sf = 0.5 とする.
30000	$x < 0$ であった.	sf = 0.0 とする.

3. 使用上の注意

a) x の範囲

引数 x の範囲は $0 \leq x < t_{\max}$ であること. x が大きくなると $\sin(x)$, $\cos(x)$ の値が正確に計算できなくなることに対する処理である. $t_{\max}$ については“1.5 計算機定数”を参照のこと.

4. 使用例

$S(x)$ の値を x が 0 から 100 まで, 1 おきに計算し数表を作る.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
    double x, sf;
    int i;

    for (i=0; i<=100; i++) {
        x = i;
        /* calculate Sine Fresnel integral */
        ierr = c_dsfri(x, &sf, &icon);
        if (icon == 0)
            printf("x = %5.2f    sf = %f\n", x, sf);
        else
            printf("ERROR: x = %5.2f    sf = %f    icon = %i\n", x, sf, icon);
    }
}
```

```
    }  
    return(0);  
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の SFRI の項目を参照のこと。

c_dsggm

行列の差 (実行列)

<pre>ierr = c_dsggm(a, ka, b, kb, c, kc, m, n, &icon);</pre>

1. 機能

$m \times n$ の実行列 **A** と実行列 **B** の差 **C** を求める.

$$\mathbf{C} = \mathbf{A} - \mathbf{B}$$

(1)

ここで **C** は, $m \times n$ の実行列である. $m, n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dsggm((double*)a, ka, (double*)b, kb, (double*)c, kc, m, n, &icon);
```

引数の説明:

a	double	入力	行列 A .
	a[m][ka]		
ka	int	入力	配列 a の整合寸法 ($\geq n$).
b	double	入力	行列 B .
	b[m][kb]		
kb	int	入力	配列 b の整合寸法 ($\geq n$).
c	double	出力	行列 C (“使用上の注意”参照).
	c[m][kc]		
kc	int	入力	配列 c の整合寸法 ($\geq n$).
m	int	入力	行列 A, B, C の行数 m .
n	int	入力	行列 A, B, C の列数 n .
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
30000	次のいずれかであった. • $m < 1$ • $n < 1$ • $ka < n$ • $kb < n$ • $kc < n$	処理を打ち切る.

3. 使用上の注意

a) 領域の節約について

配列 a あるいは, b 上の内容を保存する必要のない場合は, 次のように呼び出すことにより領域が節約できる.

配列 a の内容が不要の場合:

```
ierr = c_dsggm((double *)a, ka, (double *)b, kb, (double *)a, ka, m, n, &icon);
```

配列 b の内容が不要の場合:

```
ierr = c_dsggm((double *)a, ka, (double *)b, kb, (double *)b, kb, m, n, &icon);
```

この場合, 行列 C は配列 a あるいは, 配列 b 上に得られる.

4. 使用例

行列の差を計算し, 結果をチェックする. 行列のサイズは 100×100 である.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 100

MAIN__()
{
    int ierr, icon;
    int n, m, ka, kb, kc, i, j;
    double eps;
    double a[NMAX][NMAX], b[NMAX][NMAX], c[NMAX][NMAX];

    /* initialize matrices */
    m = NMAX;
    n = NMAX;
    ka = NMAX;
    kb = NMAX;
    kc = NMAX;
    for (i=0; i<n; i++)
        for (j=0; j<n; j++) {
            a[i][j] = n+i+j;
            b[i][j] = i+j;
        }
    /* subtract matrices */
    ierr = c_dsggm((double*)a, ka, (double*)b, kb, (double*)c, kc, m, n, &icon);
    if (icon != 0) {
        printf("ERROR: c_dsggm failed with icon = %d\n", icon);
        exit(1);
    }
    /* check matrix */
    eps = 1e-6;
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            if (fabs((c[i][j]-n)/n) > eps) {
                printf("WARNING: result inaccurate\n");
                exit(1);
            }
    printf("Result OK\n");
    return(0);
}
```

c_dsimp1

1次元有限区間積分 (等間隔離散点入力, シンプソン則)

ierr = c_dsimp1(y, n, h, &s, &icon);

1. 機能

等間隔離散点 $x_i = x_1 + (i-1)h$, $i = 1, 2, \dots, n$ における関数値 $y_i = f(x_i)$ が与えられたときシンプソン則により,

$$S = \int_{x_1}^{x_n} f(x) dx, \quad n > 2 \quad h > 0$$

を求める. ここで h はキザミ値である.

2. 引数

呼出し形式:

```
ierr = c_dsimp1(y, n, h, &s, &icon);
```

引数の説明:

y	double y[n]	入力	関数値 y_i .
n	int	入力	離散点の個数 n .
h	double	入力	キザミ値 h .
s	double	出力	積分値 S .
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	$n = 2$ であった.	台形則により計算する. (手法概要参照)
30000	$n < 2$ 又は $h \leq 0.0$ であった.	$s = 0.0$ として処理を打ち切る.

3. 使用例

100 個の等間隔離散点から積分値を計算し, 結果を真の値と比較する.

```
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 100

MAIN__()
{
    int ierr, icon;
    int i, n;
    double x, h, y[NMAX], s, eps, exact;

    /* initialize data */
    n = NMAX;
    x = 0;
    h = 1.0/(n-1);
    for (i=0; i<n; i++) {
```

```

    y[i] = x*x;
    x = x + h;
}
/* calculate integral */
ierr = c_dsimp1(y, n, h, &s, &icon);
printf("icon = %i integral = %12.4e\n", icon, s);
/* check result */
eps = 1e-6;
exact = 1.0/3.0;
if (fabs((s-exact)/exact) > eps)
    printf("Inaccurate result\n");
else
    printf("Result OK\n");
return(0);
}

```

4. 手法概要

区間 $[x_1, x_n]$ について, 等間隔に取られた離散点に対する関数値 y_i を用い, シンプソン則により積分する.

まず最初の 3 分点について, 2 次補間式で近似し, 区間 $[x_1, x_3]$ にわたり積分をする.

$$\int_{x_1}^{x_3} f(x) dx \cong \frac{h}{3}(y_1 + 4y_2 + y_3)$$

次に続く 3 分点についても同様に計算は進められる.

$$\int_{x_1}^{x_n} f(x) dx \cong \frac{h}{3}(y_1 + 4y_2 + 2y_3 + 4y_4 + 2y_5 + \dots + 4y_{n-1} + y_n)$$

離散点の個数が奇数の場合は, 上述のように, 3 分点によって最後まで計算するが, 離散点の個数が偶数の場合は, 区間 $[x_1, x_{n-3}]$ までは上述の方法で計算し, 最後の区間 $[x_{n-3}, x_n]$ にわたっては, ニュートン・コーツ 3/8 則を適用して計算する.

$$\int_{x_{n-3}}^{x_n} f(x) dx \cong \frac{3h}{8}(y_{n-3} + 3y_{n-2} + 3y_{n-1} + y_n)$$

ただし, $n=2$ の場合は, シンプソン則は適用できないので特に台形則を適用する.

$$\int_{x_1}^{x_2} f(x) dx \cong \frac{h}{2}(y_1 + y_2)$$

計算方法の詳細については[89]を参照のこと.

c_dsini

正弦積分 $S_i(x)$

<code>ierr = c_dsini(x, &si, &icon);</code>

1. 機能

正弦積分 $S_i(x)$ の値,

$$S_i(x) = \int_0^x \frac{\sin(t)}{t} dt$$

を多項式及び漸近形式の近似式を用いて計算する.

2. 引数

呼出し形式:

```
ierr = c_dsini(x, &si, &icon);
```

引数の説明:

x	double	入力	独立変数 x .
si	double	出力	関数値 $S_i(x)$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	$ x \geq t_{\max}$ であった.	$si = \text{sign}(x) \cdot \pi/2$ とする.

3. 使用上の注意

a) 引数 x

引数 x の範囲は,

$$|x| < t_{\max}$$

であること.

$|x|$ が大きくなると $\sin(x)$, $\cos(x)$ の値が正確に計算できなくなることに對する処理である.

4. 使用例

$S_i(x)$ の値を x が 0 から 10 まで 0.1 おきに計算し, 数表を作る.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

MAIN__()
{
    int ierr, icon;
    double x, si;
    int i;
```



```
for (i=0;i<100;i++) {
    x = (double)i/10;
    /* calculate complete elliptic integral */
    ierr = c_dsini(x, &si, &icon);
    if (icon == 0)
        printf("x = %5.2f    si = %f\n", x, si);
    else
        printf("ERROR: x = %5.2f    si = %f    icon = %i\n", x, si, icon);
    }
return(0);
}
```

5. 手法概要

正弦積分 $S_i(x)$ は, x の値によって近似式の形が異なる.

- $0 \leq |x| < 4$ のとき, ベキ級数展開の近似式を用いて計算する.
- $|x| \geq 4$ のとき, 漸近展開の近似式を用いて計算する.

計算方法の詳細については“富士通 SSL II 使用手引書”の SINI の項目を参照のこと.

c_dsmdm

実対称行列の MDM^T 分解 (ブロック対角ピボッティング手法)

```
ierr = c_dsmdm(a, n, epsz, ip, vw, ivw,
               &icon);
```

1. 機能

$n \times n$ の正則な実対称行列 $\mathbf{A}$ をブロック対角ピボッティング手法 (同名手法にはクラウト法とガウス消去法の関係に似た二つの考え方があり, ここでは前者) により MDM^T 分解する.

$$\mathbf{PAP}^T = \mathbf{MDM}^T \quad (1)$$

ただし, $\mathbf{P}$ はピボッティングによる行の入換えを行う置換行列, $\mathbf{M}$ は単位下三角行列, $\mathbf{D}$ は高々次数 2 の対称なブロックから成る対称ブロック対角行列で $d_{k+1,k} \neq 0$ なら $m_{k+1,k} = 0$ である. ただし, $\mathbf{M} = (m_{ij})$, $\mathbf{D} = (d_{ij})$, $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dsmdm(a, n, epsz, ip, vw, ivw, &icon);
```

引数の説明:

a	double	入力	行列 $\mathbf{A}$. 対称行列用圧縮格納法. 格納方法の詳細については“1.3.2 対称行列の格納方法”を参照のこと. $Alen = n(n+1)/2$.
	a[Alen]	出力	行列 $\mathbf{D} + (\mathbf{M} - \mathbf{I})$. 対称行列用圧縮格納法.
n	int	入力	行列 $\mathbf{A}$ の次数 n .
epsz	double	入力	ピボットの相対零判定値 (≥ 0.0). 0.0 のときは標準値が採用される. (“使用上の注意”a)参照)
ip	int ip[n]	出力	ピボッティングによる行の入換えの履歴を示すトランスポジションベクトル. (“使用上の注意”b)参照)
vw	double vw[2n]	作業領域	
ivw	int ivw[n]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	ピボットが相対的に零となった. 行列は非正則の可能性が強い.	処理を打ち切る.
30000	次のいずれかであった. • $n < 1$ • $epsz < 0$	処理を打ち切る.

3. 使用上の注意

a) epsz

ピボットの相対零判定値 $epsz$ に 10^{-8} を設定したとすると, この値は次の意味を持っている. すなわち, ブロック対角ピボッティング手法による MDM^T 分解の過程でピボットの値 (1×1 若しくは, 2×2 ピボットが成す小行列の行列式)が, 10^{-8} 桁以上の桁落ちを生じた場合に, そのピボットの値を相対的に零と見なし, $icon = 20000$ として処理を打ち切る. $epsz$ の標準値は丸め誤差の単位を μ としたとき $16 \cdot \mu$ である. なお, ピ

ポットが小さくなくても計算を続行させる場合には, `epsz` へ極小の値を与えればよいが, その結果は保証されない.

b) ip

トランスポジションベクトルとは, ブロック対角ピボッティング手法による MDM^T 分解 (1)における置換行列 **P** に相当する. 本関数では, ピボッティングに伴い配列 `a` の内容を実際に交換しており, その履歴を `ip` に格納している. しかし, 1×1 ピボットと 2×2 ピボットの場合では, `ip` の格納法が多少異なるので注意すること. 分解の k 段階目における格納方法は次のとおりである.

1×1 ピボットのときは, 第 k 行 (及び列) と交換する行 (及び列) の番号 r ($r \geq k$) を `ip[k-1]` に格納する. 2×2 ピボットのときは, 更に第 $k+1$ 行 (及び列) と交換する行 (及び列) の番号 s ($s \geq k+1$) の負値 ($-s$) を `ip[k]` に格納する.

c) 連立 1 次方程式の解

本関数に続けて, `c_dmdmx` を呼び出すことにより連立 1 次方程式を解くことができる. しかし, 通常は `c_dlsix` を呼び出せば, 一度に解が求められる.

d) 固有値の数

行列 **A** の正, 負固有値の数を各々求めることができる. (“使用例”参照)

e) 行列式

行列 **A** の行列式は, 演算後の行列 **D** の行列式に等しい. (“使用例”参照)

4. 使用例

$n \times n$ の実対称行列の性質が調べられる正と負の固有値の数と行列式の値を求める.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 100

MAIN__()
{
    int ierr, icon;
    int n, i, j, ij, cnt, peig, neig;
    double epsz, eps, pi, an, ar, det;
    double a[NMAX*(NMAX+1)/2], vw[2*NMAX];
    int ip[NMAX], ivw[NMAX];

    /* initialize matrix */
    n = NMAX;
    ij = 0;
    pi = 2*asin(1);
    an = 1.0/(n+1);
    ar = pi*an;
    an = sqrt(2*an);
    for (i=1;i<=n;i++)
        for (j=1;j<=i;j++) {
            a[ij++] = an*sin(i*j*ar);
        }
    epsz = 1e-6;
    /* MDM decomposition of system */
    ierr = c_dsmdm(a, n, epsz, ip, vw, ivw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dsmdm failed with icon = %d\n", icon);
        exit(1);
    }
    /* find number of positive and negative eigenvalues */
    peig = 0;
    neig = 0;
    i = 1;
```

```
j = 1;
while (j<n) {
    if (ip[j] <= 0) {
        peig++;
        neig++;
        j = j+2;
        i = i+j-1+j;
    }
    else {
        if (a[i-1] > 0) peig++;
        else if (a[i-1] < 0) neig++;
        j++;
        i = i+j;
    }
}
if (j == n) {
    if (a[i-1] > 0) peig++;
    else if (a[i-1] < 0) neig++;
}
printf("Positive e-values: %i\n", peig);
printf("Negative e-values: %i\n", neig);
/* calculate determinant */
det = 1;
i = 1;
j = 1;
while (j<n) {
    if (ip[j] <= 0) {
        det = det*(a[i-1]*a[i+j]-a[i+j-1]*a[i+j-1]);
        j = j+2;
        i = i+j-1+j;
    }
    else {
        det = det*a[i-1];
        j++;
        i = i+j;
    }
}
printf("Determinant: %12.5e\n", det);
return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の SMDM の項目及び[15]を参照のこと。

c_dsmle1

最小二乗近似多項式による平滑化 (等間隔離散点)
ierr = c_dsmle1(y, n, m, l, f, &icon);

1. 機能

独立変数の等間隔な点における観測値 y_i , $i = 1, \dots, n$ が与えられたとき, 局所的な最小二乗近似多項式を用いてこれらの観測値を平滑化する.

すなわち, ある 1 つの観測値を平滑化するとき, 全体の観測値を用いた最小二乗近似多項式を適用する代わりに, その隣接する指定された個数 l (ただし, $l \leq n$) の観測値に関する指定された次数 m の局所的最小二乗近似多項式を適用して平滑化する. この手続きを全ての観測値について行うものである.

ただし, 本関数では, m と l の組合せを, 次のように制限する.

- $m = 1$ のとき, 観測点の個数 l は 3 又は 5.
- $m = 3$ のとき, 観測点の個数 l は 5 又は 7.

2. 引数

呼出し形式:

```
ierr = c_dsmle1(y, n, m, l, f, &icon);
```

引数の説明:

y	double y[n]	入力	観測値 y_i .
n	int	入力	観測値の個数 n .
m	int	入力	局所的最小二乗近似多項式の次数 m .
l	int	入力	局所的最小二乗近似多項式に用いる観測値の個数 l .
f	double f[n]	出力	平滑値.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	次のいずれかであった. • $m \neq 1$ かつ $m \neq 3$ • $m = 1$ のとき $l \neq 3$ かつ $l \neq 5$ • $m = 3$ のとき $l \neq 5$ かつ $l \neq 7$ • $n < l$	処理を打ち切る.

3. 使用上の注意

a) 本関数の利用方法

本関数は, 真の関数が, 全体にわたり唯一の多項式で近似できないが局所的にはある次数の多項式で近似できることを前提としている.

b) m と l の選び方

m と l の選択にあたっては、観測値に対する物理的情報と利用者の経験に基づいてなされるべきである。しかし、平滑化の程度は l の値に伴い増加し、 m の値に伴い減少することを考慮する必要がある。

c) 本関数の繰り返し適用について

本関数を繰り返し呼ぶこと、つまり、 l 個に関する m 次の最小二乗近似多項式を、求まった平滑値に繰り返し適用することも可能であるが、むやみに反復させると、全体の観測値に関する m 次の最小二乗近似多項式を適用した結果に近づく傾向がある。もし反復する場合、いつ反復を止めるべきかは利用者の判断による。

4. 使用例

[0,1] 区間で等間隔な 10 個の点での関数 $f(x) = \sin(x)\sqrt{x}$ の観測値が与えられたとき、これらの観測値を平滑化する。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 10

MAIN__()
{
    int ierr, icon;
    int i, n, m, l;
    double y[NMAX], f[NMAX];
    double h, p;

    /* initialize data */
    n = NMAX;
    p = 0;
    h = 1.0/n;
    for (i=0;i<n;i++) {
        y[i] = sin(p)*sqrt(p);
        p = p + h;
    }
    m = 1;
    l = 5;
    /* smooth data */
    ierr = c_dsmle1(y, n, m, l, f, &icon);
    if (icon != 0) {
        printf("ERROR: c_dsmle1 failed with icon = %d\n", icon);
        exit(1);
    }
    for (i=0;i<n;i++)
        printf("%12.4e %12.4e \n", y[i], f[i]);
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の SMLE1 の項目及び[54], [89]を参照のこと。

c_dsmle2

最小二乗近似多項式による平滑化 (不等間隔離散点)

ierr = c_dsmle2(x, y, n, m, l, w, f, vw, &icon);

1. 機能

離散点 $x_1, x_2, \dots, x_n$ ($x_1 < x_2 < \dots < x_n$) に対して, 観測値 y_i , 重み関数値 $w(x_i)$, $i = 1, 2, \dots, n$ が与えられたとき, 局所的な最小二乗近似多項式を用いて, これらの観測値を平滑化する.

すなわち, ある一つの観測値を平滑化するとき, 全体の観測値を用いた最小二乗近似多項式を適用する代わりに, その隣接する指定された個数 λ の観測値に関する指定された次数 m の局所的な最小二乗近似多項式を適用して平滑化する. この手続きをすべての観測値について行うものである.

ただし, $\lambda \leq n$, $w(x_i) \geq 0$ ($i = 1, \dots, n$), $\lambda \geq m + 2$, λ は奇数, $m \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dsmle2(x, y, n, m, l, w, f, vw, &icon);
```

引数の説明:

x	double x[n]	入力	離散点 x_i .
y	double y[n]	入力	観測値 y_i .
n	int	入力	観測値の個数 n .
m	int	入力	局所的な最小二乗近似多項式の次数 m .
l	int	入力	局所的な最小二乗近似多項式に用いる観測値の個数 λ .
w	double w[n]	入力	重み関数値 $w(x_i)$. 通常は $w(x_i) = 1$ として用いる.
f	double f[n]	出力	平滑値.
vw	double vw[2l]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	次のいずれかであった. $x[0] < x[1] < \dots < x[n-1]$ を満足しない $l > n$ 又は l が偶数である $m < 1$ 又は $l < m + 2$ $w[i]$ に負のものがある	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

本関数は, 真の関数が, 全体にわたり唯一の多項式で近似できないが局所的にはある次数の多項式で近似できることを前提としている.

本関数を繰り返し呼ぶこと、つまり、 λ 個に関する m 次の最小二乗近似多項式を、求まった平滑値に繰り返し適用することも可能であるが、むやみに反復させると、全体の観測値に関する m 次の最小二乗近似多項式を適用した結果に近づく傾向がある。もし反復する場合、いつ反復を止めるべきかは利用者の判断による。

本関数は、等間隔な独立変数における場合も使用できるが、関数 c_dsmle1 に比べ処理時間が長い。

b) m と l の選択

m と λ の選択にあたっては、観測値に対する物理的情報と利用者の経験に基づいてなされるべきである。しかし、平滑化の程度は λ の値に伴い増加し、 m の値に伴い減少することを考慮する必要がある。

4. 使用例

[0, 1]区間で等間隔な 10 個の離散点での関数 $f(x) = \sin(x)\sqrt{x}$ の値が与えられたとき、各々観測値を平滑化する。

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define N 10
#define M 3

MAIN__()
{
    int ierr, icon;
    int i, n, m, l;
    double x[N], y[N], w[N], f[N], vw[21];
    double p, h;

    /* initialize data */
    n = N;
    p = 0;
    h = 1.0/(n-1);
    for (i=0; i<n; i++) {
        w[i] = 1;
        x[i] = p+i*h;
        y[i] = sin(x[i])*sqrt(x[i]);
    }
    l = 5;
    m = 2;

    /* smooth data */
    ierr = c_dsmle2(x, y, n, m, l, w, f, vw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dsmle2 failed with icon = %d\n", icon);
        exit(1);
    }
    for (i=0; i<n; i++)
        printf("%12.4e %12.4e \n", y[i], f[i]);
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の SMLE2 の項目及び[54], [89]を参照のこと。

c_dsssm

行列の差 (実対称行列)

<code>ierr = c_dsssm(a, b, c, n, &icon);</code>

1. 機能

$n \times n$ の実対称行列 **A** と実対称行列 **B** の差 **C** を求める.

$$\mathbf{C} = \mathbf{A} - \mathbf{B}$$

ここで, **C** は $n \times n$ の実対称行列である. $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dsssm(a, b, c, n, &icon);
```

引数の説明:

a	double a[Alen]	入力	行列 A . 対称行列用圧縮格納法. 格納方法の詳細については“1.3.2 対称行列の格納方法”を参照のこと. $Alen = n(n+1)/2$.
b	double b[Blen]	入力	行列 B . 対称行列用圧縮格納法. 格納方法の詳細については“1.3.2 対称行列の格納方法”を参照のこと. $Blen = n(n+1)/2$.
c	double c[Clen]	入力	行列 C . 対称行列用圧縮格納法. 格納方法の詳細については“1.3.2 対称行列の格納方法”を参照のこと. $Clen = n(n+1)/2$. (“使用上の注意”参照)
n	int	入力	行列 A , B 及び C の次数 n .
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
30000	$n < 1$ であった.	処理を打ち切る.

3. 使用上の注意

a) 領域の節約

配列 **a** あるいは, **b** 上の内容を保存する必要のない場合は, 次のように呼び出すことにより領域が節約できる. この場合, 行列 **C** は配列 **a** あるいは, **b** 上に得られる.

配列 **a** の内容が不要の場合:

```
ierr = c_dsssm(a, b, a, n, &icon);
```

配列 **b** の内容が不要の場合:

```
ierr = c_dsssm(a, b, b, n, &icon);
```

4. 使用例

実対称行列の差を計算する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */
```

```
#define NMAX 100

MAIN__()
{
    int ierr, icon;
    int n, i, j, ij;
    double eps, err;
    double a[NMAX*(NMAX+1)/2], b[NMAX*(NMAX+1)/2], c[NMAX*(NMAX+1)/2];

    /* initialize matrices */
    n = NMAX;
    ij = 0;
    for (i=0; i<n; i++)
        for (j=0; j<=i; j++) {
            a[ij] = n+i-j+1;
            b[ij++] = i-j+1;
        }
    /* add matrices */
    ierr = c_dsssm(a, b, c, n, &icon);
    if (icon != 0) {
        printf("ERROR: c_dsssm failed with icon = %d\n", icon);
        exit(1);
    }
    /* check matrix */
    eps = 1e-6;
    ij = 0;
    for (i=0; i<n; i++)
        for (j=0; j<=i; j++) {
            err = fabs((c[ij++] - n) / n);
            if (err > eps) {
                printf("WARNING: result inaccurate\n");
                exit(1);
            }
        }
    printf("Result OK\n");
    return(0);
}
```

c_dteig1

実対称 3 重対角行列の固有値及び固有ベクトル(QL 法)

<code>ierr = c_dteig1(d, sd, n, e, ev, k, &m, &icon);</code>

1. 機能

n 次の実対称 3 重対角行列 $\mathbf{T}$ の固有値及び固有ベクトルを QL 法により求める. 固有ベクトルは $\|\mathbf{x}\|_2 = 1$ となるように正規化する. $n \geq 1$ なること.

2. 引数

呼出し形式:

```
ierr = c_dteig1(d, sd, n, e, (double *)ev, k, &m, &icon);
```

引数の説明:

d	double d[n]	入力	実対称 3 重対角行列 $\mathbf{T}$ の主対角要素.
		出力	演算後, 内容は保存されない.
sd	double sd[n]	入力	3 重対角行列 $\mathbf{T}$ の副対角要素. $\text{sd}[i-1]$, $i = 2, \dots, n$ に副対角要素を格納すること.
		出力	演算後, 内容は保存されない.
n	int	入力	3 重対角行列 $\mathbf{T}$ の次数 n .
e	double e[n]	出力	固有値. 求まった順に格納される.
ev	double ev[n][k]	出力	固有ベクトル. 固有ベクトルは行方向に格納される.
k	int	入力	配列 ev の整合寸法 ($\geq n$).
m	int	出力	求まった固有値・固有ベクトルの数. (“使用上の注意”a)参照)
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	$n = 1$	$e[0] = d[0]$, $ev[0][0] = 1$ とする.
15000	固有値固有ベクトルのすべてを求めつくすことはできなかった.	m に求まっている固有値固有ベクトルの数をセットする.
20000	固有値固有ベクトルは 1 つも求められなかった.	$m = 0$ とする.
30000	次のいずれかであった. • $n < 1$ • $k < n$	処理を打ち切る.

3. 使用上の注意

a) m

引数 m には, $\text{icon} = 0$ のとき n が, また, $\text{icon} = 15000$ のとき, 求まった固有値及び固有ベクトルの数がセットされている.

b) 本関数の使用方法

本関数は、実対称 3 重対角行列用である。実対称行列の固有値及び固有ベクトルを求める場合は、関数 c_dseig1 を用いること。実対称 3 重対角行列 **T** の固有値だけを求める場合は、関数 c_dtrql を用いること。

4. 使用例

n 次の実対称 3 重対角行列 **T** の固有値及び固有ベクトルを求める。

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 5

MAIN__()
{
    int ierr, icon;
    int n, m, i, j, k;
    double d[NMAX], sd[NMAX], e[NMAX], ev[NMAX][NMAX];

    /* initialize matrix */
    n = NMAX;
    k = NMAX;
    for (i=0;i<n;i++) {
        d[i] = n-i;
    }
    for (i=1;i<n;i++) {
        sd[i] = (double)(n-i)/2;
    }
    /* find eigenvalues and eigenvectors */
    ierr = c_dteig1(d, sd, n, e, (double*)ev, k, &m, &icon);
    printf("icon = %i\n", icon);
    /* print eigenvalues and eigenvectors */
    for (i=0;i<m;i++) {
        printf("eigenvalue: %7.4f\n", e[i]);
        printf("eigenvector: ");
        for (j=0;j<n;j++)
            printf("%7.4f ", ev[i][j]);
        printf("\n");
    }
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の TEIG1 の項目及び[118], [119], [127]を参照のこと。

c_dteig2

実対称 3 重対角行列の固有値及び固有ベクトル
(バイセクション法, 逆反復法)

```
ierr = c_dteig2(d, sd, n, m, e, ev, k, vw,
                &icon);
```

1. 機能

n 次の実対称 3 重対角行列 $\mathbf{T}$ の大きい方から又は小さい方から m 個の固有値と対応する固有ベクトルを求める。固有値はバイセクション法により、固有ベクトルは逆反復法により求める。固有ベクトルは $\|x\|_2 = 1$ となるように正規化される。 $1 \leq m \leq n$ なること。

2. 引数

呼出し形式:

```
ierr = c_dteig2(d, sd, n, m, e, (double *)ev, k, vw, &icon);
```

引数の説明:

d	double d[n]	入力	実対称 3 重対角行列 $\mathbf{T}$ の主対角要素.
sd	double sd[n]	入力	実対称 3 重対角行列 $\mathbf{T}$ の副対角要素. ただし, sd[1] ~ sd[n-1] に副対角要素を格納しておくこと.
n	int	入力	実対称 3 重対角行列 $\mathbf{T}$ の次数 n .
m	int	入力	求める固有値の個数 m . $m = +m$ のときは大きい方から求める. $m = -m$ のときは小さい方から求める.
e	double e[m]	出力	固有値.
ev	double ev[m][k]	出力	固有ベクトル. 固有ベクトルは行方向に格納される.
k	int	入力	配列 ev の整合寸法 ($k \geq n$).
vw	double vw[5n]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
10000	$n=1$ であった.	$e[0] = d[0]$ $ev[0][0] = 1$ とする.
15000	固有ベクトルのすべてを求めることはできなかった.	求められなかった固有ベクトルを 0 ベクトルとする.
20000	固有ベクトルが一つも求められなかった.	すべての固有ベクトルを 0 ベクトルとする.
30000	次のいずれかであった. $n < m$ $k < n$ $m = 0$	処理を打ち切る.

3. 使用例

5×5の実対称三重対角行列の固有値及び固有ベクトルを求める.

```
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 5

MAIN__()
{
    int ierr, icon;
    int n, m, i, j, k;
    double d[NMAX], sd[NMAX], e[NMAX], ev[NMAX][NMAX], vw[5*NMAX];

    /* initialize matrix */
    n = NMAX;
    k = NMAX;
    for (i=0;i<n;i++) {
        d[i] = n-i;
    }
    for (i=1;i<n;i++) {
        sd[i] = (double)(n-i)/2;
    }
    m = n;
    /* find eigenvalues and eigenvectors */
    ierr = c_dteig2(d, sd, n, m, e, (double*)ev, k, vw, &icon);
    printf("icon = %i\n", icon);
    /* print eigenvalues and eigenvectors */
    for (i=0;i<m;i++) {
        printf("eigenvalue: %7.4f\n", e[i]);
        printf("eigenvector: ");
        for (j=0;j<n;j++)
            printf("%7.4f ", ev[i][j]);
        printf("\n");
    }
    return(0);
}
```

4. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”のTEIG2の項目及び[118], [119]を参照のこと.

c_dtrap

1次元有限区間積分 (不等間隔離散点入力, 台形則)

ierr = c_dtrap(x, y, n, &s, &icon);

1. 機能

不等間隔離散点 $x_1, x_2, \dots, x_n$ ($x_1 < x_2 < \dots < x_n$) に対して, 関数値 $y_i = f(x_i), i = 1, 2, \dots, n$ が与えられたとき, 台形則により,

$$S = \int_{x_1}^{x_n} f(x) dx$$

を求める.

2. 引数

呼出し形式:

```
ierr = c_dtrap(x, y, n, &s, &icon);
```

引数の説明:

x	double x[n]	入力	離散点 x_i .
y	double y[n]	入力	関数値 y_i .
n	int	入力	離散点の個数 n .
s	double	出力	積分値 S .
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	$n < 2$ 又は $x_i \geq x_{i+1}$ であった.	$s = 0.0$ として処理を打ち切る.

3. 使用上の注意

a) 関数の使用条件

離散点 x_i が等間隔の場合でも本関数は適用できるが, シンプソン則による方法(関数:c_dsimp1)を適用した方が一般的に精度が良い.

4. 使用例

100 個の離散点から積分値を求め, 結果を真の値と比較する.

```
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 100

MAIN__()
{
```

```
int ierr, icon;
int i, n;
double h, p, x[NMAX], y[NMAX], s, eps, exact;

/* initialize data */
n = NMAX;
p = 0;
h = 1.0/(n-1);
for (i=0;i<n;i++) {
    x[i] = p;
    y[i] = p*p;
    p = p + h;
}
/* calculate integral */
ierr = c_dtrap(x, y, n, &s, &icon);
printf("icon = %i integral = %12.4e\n", icon, s);
/* check result */
eps = 1e-4;
exact = 1.0/3.0;
if (fabs((s-exact)/exact) > eps)
    printf("Inaccurate result\n");
else
    printf("Result OK\n");
return(0);
}
```

5. 手法概要

区間 $[x_1, x_n]$ について, 不等間隔に取られた離散点を用い, 台形則により次のように積分する.

$$\begin{aligned}\int_{x_1}^{x_n} f(x)dx &\cong \frac{1}{2} \left((x_2 - x_1)(f(x_1) + f(x_2)) + (x_3 - x_2)(f(x_2) + f(x_3)) + \dots + (x_n - x_{n-1})(f(x_{n-1}) + f(x_n)) \right) \\ &\cong \frac{1}{2} \left((x_2 - x_1)f(x_1) + (x_3 - x_1)f(x_2) + \dots + (x_n - x_{n-2})f(x_{n-1}) + (x_n - x_{n-1})f(x_n) \right)\end{aligned}$$

計算方法の詳細については[89]を参照のこと.

c_dtrbk

実対称行列の固有ベクトルへの逆変換

<code>ierr = c_dtrbk(ev, k, n, m, p, &icon);</code>

1. 機能

n 次の実対称 3 重対角行列 $\mathbf{T}$ の m 個の固有ベクトルを実対称行列 $\mathbf{A}$ の固有ベクトルへ逆変換する。ただし、 $\mathbf{T}$ は $\mathbf{A}$ にハウスホルダー法を適用して得られたものであること。かつ $1 \leq m \leq n$ であること。

2. 引数

呼出し形式:

```
ierr = c_dtrbk((double *) ev, k, n, m, p, &icon);
```

引数の説明:

ev	double	入力	実対称 3 重対角行列 $\mathbf{T}$ の m 個の固有ベクトル。
	ev[m][k]	出力	実対称行列 $\mathbf{A}$ の固有ベクトル..
k	int	入力	配列 ev の整合寸法 ($\geq n$)。
n	int	入力	実対称 3 重対角行列の次数 n 。
m	int	入力	固有ベクトルの個数 m 。 m の内容が負であるときは、その絶対値をとって使う。
p	double	入力	ハウスホルダー法による $\mathbf{A}$ から $\mathbf{T}$ への変換行列 $\mathbf{P}$ 。対称行列用圧縮格納法。格納法の詳細については“1.3.2 対称行列の格納方法”を参照のこと。(“使用上の注意”a)参照)
	p[n(n+1)/2]		
icon	int	出力	コンディションコード。下の表を参照。

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし。	正常終了。
10000	$n = 1$	ev[0][0] = 1.
30000	次のいずれかであった。 • $m = 0$ 又は $ m > n$ • $k < n$	処理を打ち切る。

3. 使用上の注意

a) 本関数の使用方法

本関数は、通常関数 c_dtrid1 に続けて呼び出される。本関数の入力引数 p は関数 c_dtrid1 の出力引数 a が与えられることを想定しているので、これをそのまま用いればよい。

b) 固有ベクトルの正規化

固有ベクトルは、 $\|\mathbf{x}_i\|_2 = 1$ となるように正規化されている。

4. 使用例

n 次の実対称行列を 3 重対角行列へ変換し、 m 個の固有値と固有ベクトルを求め、3 重対角行列の固有ベクトルを実対称行列の固有ベクトルへ逆変換する。

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

```
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 5

MAIN__()
{
    int ierr, icon;
    int n, i, j, k, ij, m;
    double a[NMAX*(NMAX+1)/2], sd[NMAX], d[NMAX];
    double e[NMAX], ev[NMAX][NMAX];

    /* initialize matrix */
    n = NMAX;
    ij = 0;
    for (i=0;i<n;i++)
        for (j=0;j<=i;j++) {
            a[ij++] = n-i;
        }
    /* reduce matrix A to symmetric tridiagonal form */
    ierr = c_dtrid1(a, n, d, sd, &icon);
    if (icon > 10000 ) {
        printf("ERROR: c_dtrid1 failed with icon = %i\n", icon);
        exit (1);
    }
    /* find eigenvalues and eigenvectors */
    k = NMAX;
    ierr = c_dteig1(d, sd, n, e, (double*)ev, k, &m, &icon);
    if (icon >= 20000 ) {
        printf("ERROR: c_dteig1 failed with icon = %i\n", icon);
        exit (1);
    }
    /* back transformation to find e-vectors of A */
    ierr = c_dtrbk((double*)ev, k, n, m, a, &icon);
    if (icon > 10000 ) {
        printf("ERROR: c_dtrbk failed with icon = %i\n", icon);
        exit (1);
    }
    printf("icon = %i\n", icon);
    /* print eigenvalues and eigenvectors */
    for (i=0;i<m;i++) {
        printf("eigenvalue:  %7.4f\n", e[i]);
        printf("eigenvector:  ");
        for (j=0;j<n;j++)
            printf("%7.4f  ", ev[i][j]);
        printf("\n");
    }
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の TRBK の項目及び [119]を参照のこと。

c_dtrbkh

エルミート行列の固有ベクトルへの逆変換

<pre>ierr = c_dtrbkh(evr, evi, k, n, m, p, pv, &icon);</pre>

1. 機能

n 次の実対称 3 重対角行列 $\mathbf{T}$ の m 個の固有ベクトル $\mathbf{y}$ を, エルミート行列 $\mathbf{A}$ の固有ベクトル $\mathbf{x}$ へ逆変換する.

$$\mathbf{x} = \mathbf{P}\mathbf{V}^* \mathbf{y}$$

ただし $\mathbf{P}$ 及び $\mathbf{V}$ はそれぞれ, ハウスホルダー法及び対角ユニタリ変換によって $\mathbf{A}$ から $\mathbf{T}$ へ変換するときの変換行列である. $1 \leq m \leq n$ になること.

2. 引数

呼出し形式:

```
ierr = c_dtrbkh((double *) evr, (double *) evi, k, n, m, (double *) p, pv,
                &icon);
```

引数の説明:

evr	double	入力	実対称 3 重対角行列 $\mathbf{T}$ の m 個の固有ベクトル $\mathbf{y}$.
	evr[m][k]	出力	n 次 of エルミート行列 $\mathbf{A}$ の固有ベクトル $\mathbf{x}$ の実部. (“使用上の注意”b)参照)
evi	double	出力	n 次 of エルミート行列 $\mathbf{A}$ の固有ベクトル $\mathbf{x}$ の虚部. (“使用上の注意”b)参照)
	evi[m][k]		
k	int	入力	配列 evr, evi の整合寸法 ($\geq n$).
n	int	入力	実対称 3 重対角行列の次数 n .
m	int	入力	固有ベクトルの個数 m . m の内容が負であるときは, その絶対値をとって使う.
p	double	入力	$\mathbf{A}$ から $\mathbf{T}$ へハウスホルダー法を適用するときの変換行列 $\mathbf{P}$. エルミート行列用圧縮格納法. 格納法の詳細については“1.3.3 エルミート行列の格納方法”を参照のこと.
	p[n][k]		(“使用上の注意”c)参照)
pv	double pv[2n]	入力	$\mathbf{A}$ から $\mathbf{T}$ へ対角ユニタリ変換を適用するときの変換行列 $\mathbf{V}$. (“使用上の注意”c)参照)
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	$n = 1$	evr[0][0] = 1 とする.
30000	次のいずれかであった. • $m = 0$ 又は $ m > n$ • $k < n$	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

本関数はエルミート行列用であり、一般複素行列については適用することができない。

b) evr 及び evi

入力した固有ベクトル $\mathbf{y}$ が $\|\mathbf{y}\|_2 = 1$ に正規化されていれば、出力する固有ベクトル $\mathbf{x}$ も $\|\mathbf{x}\|_2 = 1$ となる。

第 j 番目の固有値に対応する固有ベクトルの第 l 番目の要素は $\text{evr}[j-1][l-1] + i \cdot \text{evi}[j-1][l-1]$ で表される。ただし $i = \sqrt{-1}$ である。

c) p 及び pv

本関数は関数 `c_dtridh` と通常は組み合わせて使う。したがって、引数 `p`, `pv` の内容は `c_dtridh` の `a`, `pv` で出力される内容を想定しているので、これらをそのまま用いればよい。

配列 `p` は $\mathbf{A}$ から $\mathbf{T}$ への変換行列 $\mathbf{P}$ を直接に表現するものではないので、注意すること。

4. 使用例

n 次のエルミート行列を実対称 3 重対角行列に変換し、 m 個の固有値と固有ベクトルを求め、実対称 3 重対角行列の固有ベクトルをエルミート行列の固有ベクトルへ逆変換する。

```
#include <stdio.h>
#include <stdlib.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 5

MAIN__()
{
    int ierr, icon;
    int n, i, j, k, m;
    double a[NMAX][NMAX], sd[NMAX], d[NMAX], pv[2*NMAX];
    double e[NMAX], evr[NMAX][NMAX], evi[NMAX][NMAX];

    /* initialize matrix */
    n = NMAX;
    k = NMAX;
    for (i=0; i<n; i++) {
        a[i][i] = n-i;
        for (j=0; j<i; j++) {
            a[i][j] = n-i;
            a[j][i] = n-i;
        }
    }
    /* reduce matrix A to symmetric tridiagonal form */
    ierr = c_dtridh((double*)a, k, n, d, sd, pv, &icon);
    if (icon > 10000) {
        printf("ERROR: c_dtridh failed with icon = %i\n", icon);
        exit (1);
    }
    /* find eigenvalues and eigenvectors */
    ierr = c_dteig1(d, sd, n, e, (double*)evr, k, &m, &icon);
    if (icon >= 20000) {
        printf("ERROR: c_dteig1 failed with icon = %i\n", icon);
        exit (1);
    }
    /* back transformation to find e-vectors of A */
    ierr = c_dtrbkh((double*)evr, (double*)evi, k, n, m, (double*)a, pv, &icon);
    if (icon > 10000) {
        printf("ERROR: c_dtrbkh failed with icon = %i\n", icon);
        exit (1);
    }
    printf("icon = %i\n", icon);
    /* print eigenvalues and eigenvectors */
    for (i=0; i<m; i++) {
        printf("eigenvalue: %7.4f\n", e[i]);
        printf("eigenvector: ");
    }
}
```

```
        for (j=0;j<n;j++)
            printf("%7.4f+i*%7.4f  ", evr[i][j], evi[i][j]);
        printf("\n");
    }
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の TRBKH の項目及び[74]を参照のこと。

c_dtrid1

実対称行列の実対称 3 重対角行列への変換 (ハウスホルダー法)

ierr = c_dtrid1(a, n, d, sd, &icon);

1. 機能

n 次の実対称行列 **A** を, ハウスホルダー法 (直交相似変換) により, 実対称 3 重対角行列 **T** へ変換する.

$$\mathbf{T} = \mathbf{P}^T \mathbf{A} \mathbf{P}$$

ただし **P** は変換行列であり, $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dtrid1(a, n, d, sd, &icon);
```

引数の説明:

a	double a[n(n+1)/2]	入力	実対称行列 A . 対称行列用圧縮格納法. 格納法の詳細については “1.3.2 対称行列の格納方法”を参照のこと.
		出力	変換行列 P . 対称行列用圧縮格納法. 格納法の詳細については “1.3.2 対称行列の格納方法”を参照のこと.
n	int	入力	実対称行列 A の次数 n .
d	double d[n]	出力	3 重対角行列 T の主対角要素.
sd	double sd[n]	出力	3 重対角行列 T の副対角要素. sd[i-1], i=2,...,n を使用し, sd[0]には 0 がセットされる.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
10000	$n=1$ 又は $n=2$ であった.	変換しない.
30000	$n < 1$ であった.	処理を打ち切る.

3. 使用上の注意

a) a

出力された配列 a は, 実対称行列 **A** の固有ベクトルを求めるために必要となる.

b) 固有値の精度

固有値の精度は, 3 重対角行列化によってある程度決定される. そのため, できるだけ精度よく, 3 重対角行列を求めることが必要であり, 本関数ではその考慮がはらわれている. しかし, 固有値に非常に大きいものと小さいものがあるとき, 小さい方の固有値は, 大きい固有値に比べて変換の影響を受けやすい. したがって小さい固有値を精度よく求めることがむずかしい場合がある.

4. 使用例

n 次の実対称行列を 3 重対角行列へ変換し, m 個の固有値と固有ベクトルを求め, 3 重対角行列の固有ベクトルを実対称行列の固有ベクトルへ逆変換する.

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

```

#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 5

MAIN__()
{
    int ierr, icon;
    int n, i, j, k, ij, m;
    double a[NMAX*(NMAX+1)/2], sd[NMAX], d[NMAX];
    double e[NMAX], ev[NMAX][NMAX];

    /* initialize matrix */
    n = NMAX;
    ij = 0;
    for (i=0;i<n;i++)
        for (j=0;j<=i;j++) {
            a[ij++] = n-i;
        }
    /* reduce matrix A to symmetric tridiagonal form */
    ierr = c_dtrid1(a, n, d, sd, &icon);
    if (icon > 10000 ) {
        printf("ERROR: c_dtrid1 failed with icon = %i\n", icon);
        exit (1);
    }
    /* find eigenvalues and eigenvectors */
    k = NMAX;
    ierr = c_dteig1(d, sd, n, e, (double*)ev, k, &m, &icon);
    if (icon >= 20000 ) {
        printf("ERROR: c_dteig1 failed with icon = %i\n", icon);
        exit (1);
    }
    /* back transformation to find e-vectors of A */
    ierr = c_dtrbk((double*)ev, k, n, m, a, &icon);
    if (icon > 10000 ) {
        printf("ERROR: c_dtrbk failed with icon = %i\n", icon);
        exit (1);
    }
    printf("icon = %i\n", icon);
    /* print eigenvalues and eigenvectors */
    for (i=0;i<m;i++) {
        printf("eigenvalue:  %7.4f\n", e[i]);
        printf("eigenvector:  ");
        for (j=0;j<n;j++)
            printf("%7.4f  ", ev[i][j]);
        printf("\n");
    }
    return(0);
}

```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の TRID1 の項目及び[119]を参照のこと。

c_dtridh

エルミート行列の実対称 3 重対角行列への変換 (ハウスホルダー法, 対角ユニタリ変換)

```
ierr = c_dtridh(a, k, n, d, sd, pv, &icon);
```

1. 機能

n 次のエルミート行列 $\mathbf{A}$ をハウスホルダー法により, エルミート 3 重対角行列 $\mathbf{H}$ に変換し,

$$\mathbf{H} = \mathbf{P}^* \mathbf{A} \mathbf{P}$$

更に対角ユニタリ変換により, 実対称 3 重対角行列 $\mathbf{T}$ に変換する.

$$\mathbf{T} = \mathbf{V}^* \mathbf{H} \mathbf{V},$$

ただし, $\mathbf{P}$ 及び $\mathbf{V}$ は変換行列である. $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dtridh((double *) a, k, n, d, sd, pv, &icon);
```

引数の説明:

a	double a[n][k]	入力	エルミート行列 $\mathbf{A}$. エルミート行列用圧縮格納法. 格納法の詳細については“1.3.3 エルミート行列の格納方法”を参照のこと.
		出力	変換行列 $\mathbf{P}$. エルミート行列用圧縮格納法. (“使用上の注意”b)参照).
k	int	入力	配列 a の整合寸法 ($\geq n$).
n	int	入力	エルミート行列の次数 n .
d	double d[n]	出力	実対称 3 重対角行列 $\mathbf{T}$ の主対角要素.
sd	double sd[n]	出力	実対称 3 重対角行列 $\mathbf{T}$ の副対角要素. sd[0] は 0 とし, sd[i-1], i = 2,...,n に格納する.
pv	double pv[2n]	出力	変換行列 $\mathbf{V}$. pv[2(i-1)] = Re(v_{ii}), pv[2(i-1)+1] = Im(v_{ii}), i = 1,...,n.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	n = 1 であった.	変換しない.
30000	n < 1 又は k < n であった.	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

本関数はエルミート行列用であり, 一般複素行列については適用することができない.

b) a

出力された配列 a と pv はエルミート行列 $\mathbf{A}$ の固有ベクトルを求めるために必要となる. この a と pv は, エルミート行列の固有ベクトルを求める関数 c_dtrbkh の p と pv にそれぞれ対応している.

c) 固有値の精度

固有値の精度は3重対角化を行った時点で、ある程度決定される。そのため、できるだけ精度よく3重対角行列を求めることが必要である。そのための配慮は、本関数においてなされている。しかし、固有値に非常に大きいものと、小さいものとが混在しているときは、小さい方の固有値は大きい方に比べて変換の影響を受けやすい。したがって、そのような場合は小さい固有値を精度よく求めることは難しいことがある。

4. 使用例

n 次のエルミート行列を実対称3重対角行列に変換し、 m 個の固有値と固有ベクトルを求め、実対称3重対角行列の固有ベクトルをエルミート行列の固有ベクトルへ逆変換する。

```
#include <stdio.h>
#include <stdlib.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 5

MAIN__()
{
    int ierr, icon;
    int n, i, j, k, m;
    double a[NMAX][NMAX], sd[NMAX], d[NMAX], pv[2*NMAX];
    double e[NMAX], evr[NMAX][NMAX], evi[NMAX][NMAX];

    /* initialize matrix */
    n = NMAX;
    k = NMAX;
    for (i=0; i<n; i++) {
        a[i][i] = n-i;
        for (j=0; j<i; j++) {
            a[i][j] = n-i;
            a[j][i] = n-i;
        }
    }
    /* reduce matrix A to symmetric tridiagonal form */
    ierr = c_dtridh((double*)a, k, n, d, sd, pv, &icon);
    if (icon > 10000 ) {
        printf("ERROR: c_dtridh failed with icon = %i\n", icon);
        exit (1);
    }
    /* find eigenvalues and eigenvectors */
    ierr = c_dteig1(d, sd, n, e, (double*)evr, k, &m, &icon);
    if (icon >= 20000 ) {
        printf("ERROR: c_dteig1 failed with icon = %i\n", icon);
        exit (1);
    }
    /* back transformation to find e-vectors of A */
    ierr = c_dtrbkh((double*)evr, (double*)evi, k, n, m, (double*)a, pv, &icon);
    if (icon > 10000 ) {
        printf("ERROR: c_dtrbkh failed with icon = %i\n", icon);
        exit (1);
    }
    printf("icon = %i\n", icon);
    /* print eigenvalues and eigenvectors */
    for (i=0; i<m; i++) {
        printf("eigenvalue:  %7.4f\n", e[i]);
        printf("eigenvector:  ");
        for (j=0; j<n; j++)
            printf("%7.4f+i*%7.4f  ", evr[i][j], evi[i][j]);
        printf("\n");
    }
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の TRIDH の項目及び[74], [119]を参照のこと。

c_dtrql

実対称 3 重対角行列の固有値(QL 法)

<code>ierr = c_dtrql(d, sd, n, e, &m, &icon);</code>

1. 機能

n 次の実対称 3 重対角行列 $\mathbf{T}$ の固有値を QL 法により求める。 $n \geq 1$ になること。

2. 引数

呼出し形式:

```
ierr = c_dtrql(d, sd, n, e, &m, &icon);
```

引数の説明:

d	double d[n]	入力	3 重対角行列 $\mathbf{T}$ の主対角要素.
		出力	演算後内容は保存されない.
sd	double sd[n]	入力	3 重対角行列 $\mathbf{T}$ の副対角要素. $sd[i-1], i = 2, \dots, n$ に格納しておくこと.
		出力	演算後内容は保存されない.
n	int	入力	3 重対角行列 の次数 n .
e	double e[n]	出力	固有値.
m	int	出力	求まった固有値の数. (“使用上の注意”a)参照)
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
10000	$n = 1$ であった.	$e[0] = d[0]$ とする.
15000	固有値のすべてを求めつくすことはできなかった.	m に求まった固有値の数をセットする. $1 \leq m < n$.
20000	固有値が一つも求まらなかった.	$m = 0$ とする.
30000	$n < 1$ であった.	処理を打ち切る.

3. 使用上の注意

a) m

引数 m は, $icon = 0$ のとき n が, また $icon = 15000$ のとき, 求まった固有値の数がセットされている.

b) 本関数の使用方法

本ルーチンは, QL 法を用いているため, 3 重対角行列の左上方の要素の絶対値が, 右下方の要素の絶対値に比べ小さいような行列のとき, 最適な方法である.

本関数は, 3 重対角行列用である. 実対称行列の固有値を求める場合は, まず関数 `c_dtrid1` によって 3 重対角行列に変換した後, 本ルーチンと呼出すこと.

3 重対角行列のおよそ $n/4$ 個以下の固有値を求める場合は, 関数 `c_dbsct1` を用いた方が処理が速い. 3 重対角行列の固有ベクトルまで求める場合は, 関数 `c_dteig1` を用いること.

4. 使用例

n 次の実対称行列を 3 重対角行列に変換し, 固有値を求める..

```
#include <stdio.h>
#include <stdlib.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 15
#define NHMAX 2

MAIN__()
{
    int ierr, icon;
    int n, nh, m, i, k, ij;
    double a[NMAX*(NHMAX+1)-NHMAX*(NHMAX+1)/2], e[NMAX];
    double sd[NMAX], d[NMAX], vw[NMAX+2*NMAX], epst;

    /* initialize matrix */
    n = NMAX;
    k = NMAX;
    nh = NHMAX;
    a[0] = 10;
    a[1] = -3;
    a[2] = 10;
    ij = (nh+1)*nh/2;
    for (i=0;i<n-nh;i++) {
        a[ij] = -6;
        a[ij+1] = -3;
        a[ij+2] = 10;
        ij = ij+nh+1;
    }
    /* reduce to tridiagonal form */
    ierr = c_dbtrid(a, n, nh, d, sd, &icon);
    if (icon > 10000 ) {
        printf("ERROR: c_dbtrid failed with icon = %i\n", icon);
        exit (1);
    }
    /* find eigenvalues using c_dbsct1 */
    m = n;
    epst = 1e-6;
    ierr = c_dbsct1(d, sd, n, m, epst, e, vw, &icon);
    if (icon > 10000 ) {
        printf("ERROR: c_dbsct1 failed with icon = %i\n", icon);
        exit (1);
    }
    printf("icon = %i\n", icon);
    /* print eigenvalues */
    printf("eigenvalues:\n");
    for (i=0;i<m;i++) {
        printf("%7.4f  ", e[i]);
    }
    printf("\n");
    /* find eigenvalues using c_dtrql */
    ierr = c_dtrql(d, sd, n, e, &m, &icon);
    if (icon >= 20000 ) {
        printf("ERROR: c_dtrql failed with icon = %i\n", icon);
        exit (1);
    }
    printf("icon = %i\n", icon);
    /* print eigenvalues */
    printf("eigenvalues:\n");
    for (i=0;i<m;i++) {
        printf("%7.4f  ", e[i]);
    }
    printf("\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の TRQL の項目及び[118], [119], [127]を参照のこと.

c_dtsd1

実超越方程式 $f(x)=0$ (ブレント法)

<code>ierr = c_dtsd1(ai, bi, fun, epst, &x, &icon);</code>

1. 機能

区間 $[a, b]$ で $f(x)$ が連続で, かつ $f(a)f(b) \leq 0$ となる 2 点 a, b が与えられたとき実超越方程式

$$f(x) = 0$$

の根を区間 $[a, b]$ の中で求める. $f(x)$ の導関数を必要とせず, $f(x)$ のふるまいを計算中に調べることでより 2 分法, 線型補間法, 逆 2 次元補間法を併用して求める.

2. 引数

呼出し形式:

```
ierr = c_dtsd1(ai, bi, fun, epst, &x, &icon);
```

引数の説明:

ai	double	入力	区間の下限 a .
bi	double	入力	区間の上限 b .
fun	function	入力	解こうとする方程式の関数 $f(x)$ を計算するユーザ関数名. プロトタイプ宣言: <code>double fun(double x);</code> 引数の説明: x double 入力 独立変数.
epst	double	入力	求めようとする近似根の絶対誤差の上限. (≥ 0) ("使用上の注意" b)参照)
x	double	出力	近似根.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	次のいずれかであった. $f(a)f(b) > 0$ $epst < 0$	処理を打ち切る.

3. 使用上の注意

a) 得られる根について

区間 $[a, b]$ 内に複数個の根を持つ場合は, 出力される根がどの根であることを保証しない.

b) 引数 epst

引数 epst は求めようとする根に対する所要の精度を与えるものである。区間 $[a, b]$ が原点を含まないかぎり $\text{epst} = 0$ とすることができ、この場合本関数は、達成できるかぎりの精度を持つ根を計算する。

4. 使用例

$f(x) = \sin^2(x) - 0.5$ の区間 $[0.0, 1.5]$ 内の 1 根を求める。

```
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

double fun(double x); /* user function prototype */

MAIN__()
{
    int ierr, icon;
    double ai, bi, x, epst, exact;

    /* initialize data */
    ai = 0;
    bi = 1.5;
    epst = 1e-6;
    /* find zero of function */
    ierr = c_dtsd1(ai, bi, fun, epst, &x, &icon);
    printf("icon = %i    x = %12.4e\n", icon, x);
    /* check result */
    exact = asin(sqrt(0.5));
    if (fabs((x-exact)/exact) > epst)
        printf("Inaccurate result\n");
    else
        printf("Result OK\n");
    return(0);
}

/* user function */
double fun(double x)
{
    return(pow(sin(x),2)-0.5);
}
```

5. 手法概要

本関数は広くデッカー (Dekker) のアルゴリズムとして知られている手法に若干の修正を加えた手法を用いている。 $f(x)$ のふるまいを調べることにより、反復における各ステップにおいて 2 分法、線型補間法、逆 2 次補間法のいずれかをある判断に基づいて使い分ける。

計算方法の詳細については“富士通 SSL II 使用手引書”の TSD1 の項目及び[9]を参照のこと。

c_dtsdm

実超越方程式 $f(x)=0$ (マラー法)
<code>ierr = c_dtsdm(&x, fun, isw, eps, eta, &m, &icon);</code>

1. 機能

与えられた初期値をもとに実超越方程式

$$f(x)=0$$

の1根をマラー (Muller) 法により求める.

2. 引数

呼出し形式:

`ierr = c_dtsdm(&x, fun, isw, eps, eta, &m, &icon);`

引数の説明:

x	double	入力	求めようとする根の初期値.
		出力	近似根.
fun	function	入力	解こうとする方程式の関数 $f(x)$ を計算するユーザ関数名. プロトタイプ宣言: <code>double fun(double x);</code> 引数の説明:
		x	double 入力 独立変数.
isw	int	入力	制御情報. 求めようとする根の収束判定法を指定する. isw は, 以下のいずれかの値を設定する. 1 $ f(x_i) \leq \text{eps}$: 収束判定法 I を満たす x_i を根とする. 2 $ x_i - x_{i-1} \leq \text{eta} \cdot x_i $: 収束判定法 II を満たす x_i を根とする. 3 上の二つの収束判定法を同時に採用し少なくとも一方が満たされたとき, x_i を根とする. (使用上の注意 a)参照)
eps	double	入力	収束判定法 I (引数“isw”の項参照)で用いられる収束判定値 (≥ 0).
eta	double	入力	収束判定法 II (引数“isw”の項参照)で用いられる収束判定値 (≥ 0). (“使用上の注意” b)参照)
m	int	入力	根を求めるための反復回数の上限 (≥ 0). (“使用上の注意” c)参照)
		出力	実際に反復した回数.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
1	求まった近似根が収束判定法 I (引数 isw の項参照)を満たした.	
2	求まった近似根が収束判定法 II (引数 isw の項参照)を満たした.	
10	反復の回数は無条件に m 回 ($m = -m$) 繰り返した.	
11	反復回数は無条件に m 回繰り返すと指定されたが ($m = -m$), その回数の反復以前に $f(x_i) = 0$ となったので反復を止めて, x_i を根とした.	
12	反復回数は無条件に m 回繰り返すと指定されたが ($m = -m$), その回数の反復以前に $ x_i - x_{i-1} \leq \mu \cdot x_i $ となったので反復を止めて, x_i を根とした.	x には最後の反復値 x_i が格納される.
10000	与えられた反復回数内で指定された収束条件を満たさなかった.	
20000	$f(x_{i-2}) = f(x_{i-1}) = f(x_i)$ の状態が起こったとき, x_{i-2} , x_{i-1} , x_i に摂動を与えて問題の回避を行なっているが, この摂動が 5 回以上行われた.	処理を打ち切る.
30000	次のいずれかであった. $m > 0$ のとき $\text{isw} = 1$ かつ $\text{eps} < 0$ $\text{isw} = 2$ かつ $\text{eta} < 0$ $\text{isw} = 3$ かつ $\text{eps} < 0$ かつ $\text{eta} < 0$ $m = 0$ $\text{isw} \neq 1, 2, 3.$	処理を打ち切る.

3. 使用上の注意

a) 引数 isw

本関数は, isw = 1 と与えられたときでも, 反復の過程で

$$|x_i - x_{i-1}| \leq \mu \cdot |x_i| \quad (\mu \text{ は丸め誤差の単位})$$

が満たされたときは反復を止め, icon = 2 と出力する. 同様に, isw = 2 と与えられたときでも, 反復の過程で

$$f(x_i) = 0$$

となったときは反復を止め, icon = 1 と出力する.

b) 引数 eta

求めようとする根が多重根であつたり, 近接根である場合は大きめにとる必要がある. なお, $0 \leq \text{eta} < \mu$ (μ は丸め誤差の単位) であつたときは, 関数内部で $\text{eta} = \mu$ と置き換えている.

c) 引数 m

反復回数 m を $m = -m$ ($m > 0$) と与えると反復は無条件に m 回繰り返す.

ただし, a)と同様に, 反復された近似根が

$$f(x_i) = 0 \quad \text{又は} \quad |x_i - x_{i-1}| \leq \mu \cdot |x_i|$$

のとき, 反復を止め, icon = 11 又は icon = 12 と出力する.

4. 使用例

$f(x) = e^x - 1$ の 1 根を初期値 $x_0 = 1$ として求める.

```
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

double fun(double x); /* user function prototype */

MAIN__()
{
    int ierr, icon;
    double x, eps, eta, exact;
    int isw, m;

    /* initialize data */
    x = 1;
    isw = 3;
    eps = 0;
    eta = 1e-6;
    m = 100;
    /* find zero of function */
    ierr = c_dtsdm(&x, fun, isw, eps, eta, &m, &icon);
    printf("icon = %i   m = %i   x = %12.4e\n", icon, m, x);
    /* check result */
    eps = 1e-6;
    exact = 0;
    if (fabs(x-exact) > eps)
        printf("Inaccurate result\n");
    else
        printf("Result OK\n");
    return(0);
}

/* user function */
double fun(double x)
{
    return(exp(x)-1);
}
```

5. 手法概要

本関数は, 実超越方程式の 1 根をマラー法を用いて求めている.

計算方法の詳細については“富士通 SSL II 使用手引書”の TSDM の項目及び[111]を参照のこと.

c_dv1dwt

1次元ウェーブレット変換
ierr = c_dv1dwt(x, n, y, isn, f, k, ls, &icon);

1. 機能

1次元ウェーブレット変換及び逆変換を行う。基底関数は、コンパクトな台を持つ直交ウェーブレットを利用する。

2. 引数

呼出し形式:

```
ierr = c_dv1dwt(x, n, y, isn, f, k, ls, &icon);
```

引数の説明:

x	double x[n]	入力	変換(isn=1)のとき, 変換するベクトルデータを格納する. ("使用上の注意"a)参照)
		出力	逆変換(isn=-1)のとき, 逆変換されたベクトルデータが格納される.
n	int	入力	変換されるデータの長さ. 2 の中で表せる正整数. ("使用上の注意"b)参照)
y	double y[n]	入力	逆変換(isn=-1)のとき, 逆変換するベクトルデータを格納する.
		出力	変換(isn=1)のとき, 変換されたベクトルデータが格納される.
isn	int	入力	変換か逆変換かを指定する. 変換 :isn= 1 逆変換 :isn=-1
f	double f[2k]	入力	変換で使われるウェーブレットフィルター. ("使用上の注意"c)参照)
k	int	入力	ウェーブレットフィルターの係数を表す正の偶数.
ls	int	入力	変換の深さ. $n \geq 2^{ls}$. $n = 2^{ls}$ のとき, 完全なウェーブレット変換が行われる.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	isn $\neq$ 1, -1	処理を打ち切る.
30002	n < 2	処理を打ち切る.
30004	n の中で表せない数である.	処理を打ち切る.
30008	次のいずれかであった. - k が偶数でない - ls < 0 又は ls > log₂ n	処理を打ち切る.

3. 使用上の注意

a) 変換結果の格納について

入力 1 次元データ x ($isn = 1$) 又は y ($isn = -1$) に対して, ウェーブレット変換の各段の High-pass フィルタ一の結果は $y[n \times 2^{-i}]$, ..., $y[n \times 2^{-i+1} - 1]$ 又は $x[n \times 2^{-i}]$, ..., $x[n \times 2^{-i+1} - 1]$, $i = 1, \dots, ls$ に格納される.

b) n

変換するデータの大きさが 2 の巾で表せないときは, 残りをゼロ詰めしてそれより大きな 2 の巾で表せる大きさ n のデータとして利用できる.

c) f

本ルーチンに先立って `c_dvwflt` を呼び出し, 1 次元ウェーブレット変換で使われるフィルタの係数を設定する必要がある. 本関数の引数 k 及び f は `c_dvwflt` の n 及び f にそれぞれ対応している.

本機能で使われる直交フィルタ(orthogonal filter)は, 長さ $2 \times k$ のベクトルで, $f[0], f[1], \dots, f[k-1]$ は Low-pass フィルタを, $f[k], f[k+1], \dots, f[2k-1]$ は High-pass フィルタを含んでいる. これらの係数には, 以下の関係がある.

$$\sum_{i=0}^{k-1} f[i]^2 = 1, \quad f[2k-1-i] = (-1)^{i+1} f[i], \quad i = 0, 1, \dots, k-1.$$

4. 使用例

ウェーブレットフィルタを生成し, 1 次元ウェーブレット変換及び逆変換を行う.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 1024
#define KMAX 6

MAIN__()
{
    int ierr, icon;
    double phai, ran, eps;
    double x[NMAX], y[NMAX], f[2*KMAX], xx[NMAX];
    int isn, i, k, ls, n;

    /* generate initial data */
    n = NMAX;
    ls = 10;
    k = KMAX;
    phai = (sqrt(5.0)-1.0)/2;
    for (i=0; i<n; i++) {
        ran = (i+1)*phai;
        x[i] = ran - (int)ran;
    }
    for (i=0; i<n; i++)
        xx[i] = x[i];
    /* generate wavelet filter */
    ierr = c_dvwflt(f, k, &icon);
    if (icon != 0) {
        printf("ERROR: c_dvwflt failed with icon = %i\n", icon);
        exit (1);
    }
    /* perform normal wavelet transform */
    isn = 1;
    ierr = c_dv1dwt(x, n, y, isn, f, k, ls, &icon);
    if (icon != 0) {
        printf("ERROR: c_dv1dwt failed with icon = %i\n", icon);
        exit (1);
    }
    /* perform inverse wavelet transform */
    isn = -1;
    ierr = c_dv1dwt(x, n, y, isn, f, k, ls, &icon);
```

```
if (icon != 0 ) {
    printf("ERROR: c_dv1dwt failed with icon = %i\n", icon);
    exit (1);
}
/* check results */
eps = 1e-6;
for (i=0;i<n;i++)
    if (fabs((x[i]-xx[i])/xx[i]) > eps) {
        printf("Inaccurate result\n");
        exit(1);
    }
printf("Result OK\n");
return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書 II”の V1DWT の項目及び[20], [27], [43], [93], [105]を参照のこと.

c_dv2dwt

2次元ウェーブレット変換

<pre>ierr = c_dv2dwt(x, m, n, y, isn, f, k, lsx, lsy, &icon);</pre>

1. 機能

2次元ウェーブレット変換及び逆変換を行う。基底関数は、コンパクトな台を持つ直交ウェーブレットを利用する。

2. 引数

呼出し形式:

```
ierr = c_dv2dwt((double *) x, m, n, (double *) y, isn, f, k, lsx, lsy,
                &icon);
```

引数の説明:

x	double x[n][m]	入力	変換(isn = 1)のとき, 変換する2次元行列データを格納する. (“使用上の注意”a)参照)
		出力	逆変換(isn = -1)のとき, 逆変換された2次元行列データが格納される.
m	int	入力	変換されるデータの列の数. 2の中で表せる正整数. (“使用上の注意”b)参照)
n	int	入力	変換されるデータの行の数. 2の中で表せる正整数. (“使用上の注意”b)参照)
y	double y[m][n]	入力	逆変換(isn = -1)のとき, 逆変換する2次元行列データを転置した形で格納する.
		出力	変換(isn = 1)のとき, 変換された2次元行列データが転置した形で格納される.
isn	int	入力	変換か逆変換かを指定する. 変換 : isn = 1 逆変換 : isn = -1
f	double f[2k]	入力	変換で使われるウェーブレットフィルター. (“使用上の注意”c)参照)
k	int	入力	ウェーブレットフィルターの係数を表す正の偶数.
lsx	int	入力	各行に対する変換の深さを表す. $m \geq 2^{lsx}$. $m = 2^{lsx}$ のとき, 完全なウェーブレット変換が行われる.
lsy	int	入力	各列に対する変換の深さを表す正整数. $n \geq 2^{lsy}$. $n = 2^{lsy}$ のとき, 完全なウェーブレット変換が行われる.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	isn ≠ 1, -1	処理を打ち切る.
30002	m < 2 又は n < 2	処理を打ち切る.
30004	m 又は n が 2 の中で表せない数である.	処理を打ち切る.

コード	意味	処理内容
30008	次のいずれかであった. • k が偶数でない • $lsx < 0$ 又は $lsx > \log_2 m$ • $lsy < 0$ 又は $lsy > \log_2 n$	処理を打ち切る.

3. 使用上の注意

a) 変換結果の格納について

入力 2 次元データの列ベクトル c_j 及び行ベクトル r_j に対して, ウェーブレット変換の各段の High-pass フィルターの結果は, それぞれ

$$c_j[n \times 2^{-i}], \dots, c_j[n \times 2^{-i+1} - 1], i = 1, \dots, lsy$$

及び

$$r_k[m \times 2^{-i}], \dots, r_k[m \times 2^{-i+1} - 1], i = 1, \dots, lsx.$$

に格納される. 2 次元ウェーブレット変換の結果は, 転置されて配列 y に格納される. 第 1 段の部分的なウェーブレット変換の High-pass フィルターの出力結果は, $y[k][j]$, $k = m/2, \dots, m-1$ and $j = n/2, \dots, n-1$ に格納される.

b) m 及び n

変換するデータの各次元の大きさが 2 の巾で表せないときは, 残りの列と行にゼロを詰めてそれより大きな 2 の巾で表せる次元の大きさ $[n][m]$ を持つデータとして利用できる.

c) f

本ルーチンに先立って `c_dvwflt` を呼び出し, 1 次元ウェーブレット変換で使われるフィルターの係数を設定する必要がある. 本関数の引数 k 及び f は `c_dvwflt` の n 及び f にそれぞれ対応している.

本機能で使われる直交フィルター(orthogonal filter)は, 長さ $2 \times k$ のベクトルで, $f[0], f[1], \dots, f[k-1]$ は Low-pass フィルターを, $f[k], f[k+1], \dots, f[2k-1]$ は High-pass フィルターを含んでいる. これらの係数には, 以下の関係がある.

$$\sum_{i=0}^{k-1} f[i]^2 = 1, \quad f[2k-1-i] = (-1)^{i+1} f[i], \quad i = 0, 1, \dots, k-1.$$

4. 使用例

ウェーブレットフィルターを生成し, 1 次元ウェーブレット変換及び逆変換を行う.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define MMAX 512
#define NMAX 256
#define KMAX 6

MAIN__()
{
    int ierr, icon;
    double phai, ran, eps;
    double x[NMAX][MMAX], y[MMAX][NMAX], f[2*KMAX], xx[NMAX][MMAX];
    int isn, i, j, k, lsx, lsy, m, n;

    /* generate initial data */
    m = MMAX;
    n = NMAX;
    lsx = 3;
    lsy = 4;
    k = KMAX;
    phai = (sqrt(5.0)-1.0)/2;
```

```
for (j=0;j<n;j++) {
    for (i=0;i<m;i++) {
        ran = ((i*n+1)+j+1)*phai;
        x[j][i] = ran - (int)ran;
    }
}
for (j=0;j<n;j++)
    for (i=0;i<m;i++)
        xx[j][i] = x[j][i];
/* generate wavelet filter */
ierr = c_dvwflt(f, k, &icon);
if (icon != 0 ) {
    printf("ERROR: c_dvwflt failed with icon = %i\n", icon);
    exit (1);
}
/* perform normal wavelet transform */
isn = 1;
ierr = c_dv2dwt((double*)x, m, n, (double*)y, isn, f, k, lsx, lsy, &icon);
if (icon != 0 ) {
    printf("ERROR: c_dv2dwt failed with icon = %i\n", icon);
    exit (1);
}
/* perform inverse wavelet transform */
isn = -1;
ierr = c_dv2dwt((double*)x, m, n, (double*)y, isn, f, k, lsx, lsy, &icon);
if (icon != 0 ) {
    printf("ERROR: c_dv2dwt failed with icon = %i\n", icon);
    exit (1);
}
/* check results */
eps = 1e-6;
for (j=0;j<n;j++)
    for (i=0;i<m;i++)
        if (fabs((x[j][i]-xx[j][i])/xx[j][i]) > eps) {
            printf("Inaccurate result\n");
            exit(1);
        }
printf("Result OK\n");
return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書 II”の V2DWT の項目及び[20], [27], [43], [93], [105]を参照のこと。

c_dvalu

実行列の LU 分解 (ブロッキング LU 分解法)
ierr = c_dvalu(a, k, n, epsz, ip, &is, vw, &icon);

1. 機能

$n \times n$ の正則な実行列 **A** をブロッキング LU 分解法 (ガウスの消去法) により LU 分解する.

$$\mathbf{PA} = \mathbf{LU}$$

ただし, **P** は部分ピボットによる行の入換えを行う置換行列, **L** は下三角行列, **U** は単位上三角行列である. $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dvalu((double*)a, k, n, epsz, ip, &is, vw, &icon);
```

引数の説明:

a	double a[n][k]	入力	行列 A .
		出力	行列 L と行列 U . (“使用上の注意” c)参照).
k	int	入力	配列 a の整合寸法 ($\geq n$).
n	int	入力	行列 A の次数 n .
epsz	double	入力	ピボットの相対零判定値 (≥ 0.0). 0.0 のときは標準値が採用される. (“使用上の注意” a)参照).
ip	int ip[n]	出力	部分ピボットによる行の入換えの履歴を示すトランス ポジションベクトル. (“使用上の注意” b)参照).
is	int	出力	行列 A の行列式を求めるための情報. 演算後の配列 a の n 個 の対角要素と is の値を掛け合わせると行列式が得られる.
vw	double vw[n]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処理 内容
0	エラーなし.	正常終了.
20000	行列 A のある行の要素がすべて零であ ったか, 又はピボットが相対的に零とな った. 行列 A は非正則の可能性が強い.	処理を打ち切る.
30000	次のいずれかであった. • $k < n$ • $n < 1$ • $epsz < 0$	処理を打ち切る.

3. 使用上の注意

a) epsz について

ピボットの相対零判定値 epsz に値を設定したとすると、この値は次の意味を持っている。すなわち、選択されたピボット要素が、実行列 $\mathbf{A} = (a_{ij})$ の絶対値最大要素 $\max |a_{ij}|$ と epsz の積以下なら

$$|a_{kk}^k| \leq \max |a_{ij}| \text{epsz}$$

そのピボットを相対的に零とみなし、 $\text{icon} = 20000$ として処理を打ち切る。 epsz の標準値は丸め誤差の単位を μ としたとき、 $\text{epsz} = 16\mu$ である。

なお、ピボットが小さくなくても計算を続行させる場合には、 epsz へ極小の値を与えればよいが、その結果は保証されない。

b) ip について

トランスポジションベクトルとは、部分ピボットを行う LU 分解

$$\mathbf{PA} = \mathbf{LU}$$

における置換行列 $\mathbf{P}$ に相当する。

本関数では、部分ピボットに伴い、配列 $\mathbf{a}$ の内容を実際に交換している。すなわち、分解の J 段階目 ($J = 1, \dots, n$) において第 I 行 ($I \geq J$) がピボット行として選択された場合には、配列 $\mathbf{a}$ の第 I 行と第 J 行の内容が交換される。そして、その履歴を示すために $\text{ip}[J-1]$ に I が格納される。

c) 逆行列

本関数に続けて、関数 `c_dvluiv` を呼び出すことにより、逆行列を計算することができる。

4. 使用例

行列 $\mathbf{A}$ とベクトル $\mathbf{x}$ の積 $\mathbf{b}$ を求める。行列 $\mathbf{A}$ は LU 分解した後、逆行列 $\mathbf{A}^{-1}$ を求める。 $\mathbf{A}^{-1}\mathbf{b}$ を計算して $\mathbf{x}$ と比較することにより、 $\mathbf{A}^{-1}$ が正しく求まっていることを確認する。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 100

MAIN__()
{
    int ierr, icon;
    int n, i, j, k, is;
    double epsz, eps;
    double a[NMAX][NMAX], ai[NMAX][NMAX];
    double b[NMAX], x[NMAX], y[NMAX], vw[NMAX];
    int ip[NMAX];

    /* initialize matrix and vector */
    n = NMAX;
    for (i=0; i<n; i++)
        for (j=i; j<n; j++) {
            a[i][j] = n-j;
            a[j][i] = n-j;
        }
    for (i=0; i<n; i++)
        x[i] = i+1;
    k = NMAX;
    /* initialize constant vector b = a*x */
    ierr = c_dmav((double*)a, k, n, n, x, b, &icon);
    epsz = 1e-6;
    /* perform LU decomposition */
    ierr = c_dvalu((double*)a, k, n, epsz, ip, &is, vw, &icon);
```



```
if (icon != 0) {
    printf("ERROR: c_dvalu failed with icon = %d\n", icon);
    exit(1);
}
/* find matrix inverse from LU factors */
ierr = c_dvluiv((double*)a, k, n, ip, (double*)ai, &icon);
if (icon != 0) {
    printf("ERROR: c_dvluiv failed with icon = %d\n", icon);
    exit(1);
}
/* calculate y = ai*b */
ierr = c_dmav((double*)ai, k, n, n, b, y, &icon);
/* compare x and y */
eps = 1e-6;
for (i=0;i<n;i++)
    if (fabs((x[i]-y[i])/y[i]) > eps) {
        printf("WARNING: result inaccurate\n");
        exit(1);
    }
printf("Result OK\n");
return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書”の VALU の項目及び[5], [7], [34], [83], [126]を参照のこと。

c_dvbcsd

非対称又は不定値のスパース行列の連立 1 次方程式
(BICGSTAB(*l*)法, 対角形式格納法)

```
ierr = c_dvbcsd(a, k, ndiag, n, nofst, b,
                itmax, eps, iguss, l, x, &iter,
                vw, &icon);
```

1. 機能

$n \times n$ の非対称/不定値なスパース行列を係数行列とする連立 1 次方程式(1)を l 次安定化双対共役勾配法 (BICGSTAB(*l*):Bi-Conjugate Gradient Stabilized(*l*) method)で解く.

$$\mathbf{Ax} = \mathbf{b} \quad (1)$$

係数行列は, 対角形式の格納法で格納する. ベクトル $\mathbf{b}$ 及び $\mathbf{x}$ は n 次元ベクトル.

2. 引数

呼出し形式:

```
ierr = c_dvbcsd((double*)a, k, ndiag, n, nofst, b, itmax, eps, iguss, l, x,
                &iter, vw, &icon);
```

引数の説明:

a	double a[ndiag][k]	入力	係数行列の非零要素を対角形式格納法で格納する. (“使用上の注意” a)参照).
k	int	入力	配列 a の整合寸法 ($\geq n$).
ndiag	int	入力	係数行列 A の非零要素を含む対角ベクトル列の総数. 配列 a の 1 次元目の大きさ.
n	int	入力	行列 A の次数 n .
nofst	int nofst[ndiag]	入力	a に格納される対角ベクトルに対応した主対角ベクトルからの距離を格納する. 上対角ベクトル列は正, 下対角ベクトル列は負の値で表す. (“使用上の注意” a)参照).
b	double b[n]	入力	連立 1 次方程式の右辺の定数ベクトルを格納する.
itmax	int	入力	BICGSTAB(<i>l</i>)法の反復回数の上限值. (> 0). 2000 程度で十分である.
eps	double	入力	収束判定に用いられる判定値. eps が 0.0 以下のとき, eps は 10^{-6} が設定される. (“使用上の注意” b)参照).
iguss	int	入力	配列 x に指定された解ベクトルの近似値から反復計算を開始するかどうかを示す制御情報. 0 のとき 解ベクトルの近似値を指定しない. 0 以外のとき 配列 x に指定された解ベクトルの近似値から反復計算を開始する.
l	int	入力	BICGSTAB(<i>l</i>)法で安定化を行うときの安定化のステップ数 l ($1 \leq l \leq 8$). ほとんどの場合 1 か 2 で十分である.
x	double x[n]	入力 出力	連立 1 次方程式の解ベクトルの近似値を指定することができる. (引数“iguss”の項を参照) 解ベクトル $\mathbf{x}$ が格納される.

iter	int	出力	BICGSTAB(l)法での実際の反復回数.
vw	double	作業領域	$Vwlen = k * (4 + 2 * l) + n + NBANDL + NBANDR$
	vw[Vwlen]		NBANDL は下バンド幅, NBANDR は上バンド幅の大きさ.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	break down を起こした.	処理を打ち切る.
20001	反復回数の上限に達した.	処理を打ち切る. 配列 x には, そのときまでに得られている近似値を出力するが, 精度は保証できない.
30000	次のいずれかであった. • $n < 1$ • $k < 1$ • $n > k$ • $l < 1$ • $l > 8$ • $ndiag < 1$ • $ndiag > k$ • $itmax \leq 0$	処理を打ち切る.
32001	$ nofst[i] > n-1$ ($0 \leq i < ndiag$)	

3. 使用上の注意

a) 収束判定について

BICGSTAB(l)法は, 残差のユーグリッドノルムが最初の残差のユーグリッドノルムと EPS の積以下になったとき収束したと見なす. 正確な解と求められた近似解の誤差はほぼ行列 **A** の条件数と EPS の積に等しくなる.

収束判定に使われる残差は, 反復法におけるベクトルとして順次計算されるもので, 真の残差の値とは多少異なる.

b) l の値について

l の値は, 1 から 8 までの整数値を与える. $l=1$ のとき, BICGSTAB 法と同じアルゴリズムになる. l の値が大きいと, 反復一回あたりのコストは増加するが, 反復回数が減り, よい収束が得られる場合がある.

c) 対角形式を使う上での注意

係数行列 **A** の外側の対角ベクトルの要素は零を設定する必要がある. 対角ベクトル列を配列 A に格納する順序に制限は特でない.

d) 対角要素によるスケーリング

本ルーチン呼び出し前に, 係数行列の対角要素が 1 になるように方程式をスケーリングしておくことで, 収束が良くなる場合がある.

e) break down について

本方法では, 反復計算が係数行列や初期ベクトルの特性のため続けられない場合(break down)がある. この場合, 出力解を初期値として再呼び出しすることで収束する場合がある. 又は, 他の手法による解法ルーチンを利用すること.

4. 使用例

$Ax = b$ を解き、得られた結果を正しい値と比較し確認する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX      100
#define UBANDW    2
#define LBANDW    1
#define L         2

MAIN__()
{
    double one=1.0, bcoef=10.0, eps=1.e-6;
    int ierr, icon, ndiag, nub, nlb, n, i, j, k;
    int itmax, iguss, l, iter;
    int nofst[UBANDW + LBANDW + 1];
    double a[UBANDW + LBANDW + 1][NMAX], b[NMAX], x[NMAX];
    double vw[NMAX * (4 + 2 * L) + NMAX + UBANDW + LBANDW];

    nub = UBANDW;
    nlb = LBANDW;
    ndiag = nub + nlb + 1;
    n = NMAX;
    k = NMAX;

    /* Set A-mat & b */
    for (i=1; i<=nub; i++) {
        for (j=0 ; j<n-i; j++) a[i][j] = -1.0;
        for (j=n-i; j<n ; j++) a[i][j] = 0.0;
        nofst[i] = i;
    }
    for (i=1; i<=nlb; i++) {
        for (j=0 ; j<i+1; j++) a[nub + i][j] = 0.0;
        for (j=i+1; j<n ; j++) a[nub + i][j] = -2.0;
        nofst[nub + i] = -i;
    }
    nofst[0] = 0;
    for (j=0; j<n; j++) {
        b[j] = bcoef;
        a[0][j] = bcoef;
        for (i=1; i<ndiag; i++) b[j] += a[i][j];
    }
    /* solve the nonsymmetric system of linear equations */
    itmax = n;
    iguss = 0;
    l = L;
    ierr = c_dvbcsd ((double*)a, k, ndiag, n, nofst, b, itmax, eps,
                    iguss, l, x, &iter, vw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dvbcsd failed with icon = %d\n", icon);
        exit(1);
    }
    /* check result */
    for (i=0; i<n; i++)
        if (fabs(x[i]-one) > eps*10.0) {
            printf("WARNING: result maybe inaccurate\n");
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書 II”の VBCSD の項目及び[101], [112]を参照のこと.

c_dvbcse

非対称又は不定値のスパース行列の連立 1 次方程式
(BICGSTAB(*l*)法, ELLPACK 形式格納法)

```
ierr = c_dvbcse(a, k, iwid, n, icol, b,
                itmax, eps, iguss, l, x, &iter,
                vw, &icon);
```

1. 機能

$n \times n$ の非対称/不定値なスパース行列を係数行列とする連立 1 次方程式(1)を l 次安定化双対共役勾配法 (BICGSTAB(*l*):Bi-Conjugate Gradient Stabilized(*l*) method)で解く.

$$\mathbf{Ax} = \mathbf{b} \quad (1)$$

係数行列は, ELLPACK 形式の格納法で格納する. ベクトル $\mathbf{b}$ 及び $\mathbf{x}$ は n 次元ベクトル.

2. 引数

呼出し形式:

```
ierr = c_dvbcse((double*)a, k, iwid, n, (int*)icol, b, itmax, eps, iguss, l,
                x, &iter, vw, &icon);
```

引数の説明:

a	double a[iwid][k]	入力	係数行列の非零要素を ELLPACK 形式格納法で格納する. ("使用上の注意" a)参照).
k	int	入力	a 及び icol の整合寸法 ($\geq n$).
iwid	int	入力	係数行列 A の非零要素の行ベクトル方向の最大個数. a 及び icol の 1 次元目の大きさ.
n	int	入力	行列 A の次数 n .
icol	int icol[iwid][k]	入力	ELLPACK 形式で使用される列指標で, a の対応する要素が いずれの列ベクトルに属するかを示す. ("使用上の注意" a)参 照).
b	double b[n]	入力	連立 1 次方程式の右辺の定数ベクトルを格納する.
itmax	int	入力	BICGSTAB(<i>l</i>)法の反復回数の上限值. (> 0). 2000 程度で十分 である.
eps	double	入力	収束判定に用いられる判定値. eps が 0.0 以下のとき, eps は 10^{-6} が設定される. ("使用上の注意" b)参照).
iguss	int	入力	配列 x に指定された解ベクトルの近似値から反復 計算を開 始するかどうかを示す制御情報. 0 のとき 解ベクトルの近似値を指定しない. 0 以外のとき 配列 x に指定された解ベクトルの近似値か ら反復計算を開始する.
l	int	入力	BICGSTAB(<i>l</i>)法で安定化を行うときの安定化のステップ数 l ($1 \leq l \leq 8$). ほとんどの場合, 1 か 2 で十分である.
x	double x[n]	入力 出力	連立 1 次方程式の解ベクトルの近似値を指定することがで きる. (引数 iguss の項を参照) 解ベクトルが格納される.

iter	int	出力	BICGSTAB(l)法での実際の反復回数.
vw	double	作業領域	$Vwlen = k * (4 + 2 * l)$
	vw[Vwlen]		
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	break down を起こした.	処理を打ち切る.
20001	最大反復回数に達した.	処理を打ち切る. 配列 x には, そのときまでに得られている近似値を出力するが, 精度は保証できない.
30000	次のいずれかであった. • $n < 1$ • $k < 1$ • $n > k$ • $l < 1$ • $l > 8$ • $iwidth < 1$ • $iwidth > k$ • $itmax \leq 0$	処理を打ち切る.

3. 使用上の注意

a) 収束判定について

BICGSTAB(l)法は, 残差のユーグリッドノルムが最初の残差のユーグリッドノルムと EPS の積以下になったとき収束したと見なす. 正確な解と求められた近似解の誤差はほぼ行列 **A** の条件数と EPS の積に等しくなる.

収束判定に使われる残差は, 反復法におけるベクトルとして順次計算されるもので, 真の残差の値とは多少異なる.

b) l の値について

l の値は, 1 から 8 までの整数値を与える. $l=1$ のとき, BICGSTAB 法と同じアルゴリズムになる. l の値が大きいと, 反復一回あたりのコストは増加するが, 反復回数が減り, よい収束が得られる場合がある.

c) 対角要素によるスケーリング

本ルーチン呼び出し前に, 係数行列の対角要素が 1 になるように方程式をスケーリングしておくことで, 収束が良くなる場合がある.

d) break down について

本方法では, 反復計算が係数行列や初期ベクトルの特性のため続けられない場合(break down)がある. この場合, 出力解を初期値として再呼び出しすることで収束する場合がある. 又は, 他の手法による解法ルーチンを利用すること.

4. 使用例

$Ax = b$ を解き, 得られた結果を正しい値と比較し確認する.

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

```

#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX      100
#define UBANDW    2
#define LBANDW    1
#define L         2

MAIN__()
{
    double lcf=-2.0, ucf=-1.0, bcoef=10.0, one=1.0, eps=1.e-6;
    int ierr, icon, nlb, nub, iwidt, n, k, itmax, iguss, l, iter, i, j, ix;
    int icol[UBANDW + LBANDW + 1][NMAX];
    double a[UBANDW + LBANDW + 1][NMAX], b[NMAX], x[NMAX];
    double vw[NMAX * (4 + 2 * L)];

    nub = UBANDW;
    nlb = LBANDW;
    iwidt = UBANDW + LBANDW + 1;
    n = NMAX;
    k = NMAX;
    for (i=0; i<iwidt; i++)
        for (j=0; j<n; j++) {
            a[i][j] = 0.0;
            icol[i][j] = j+1;
        }
    /* Set A-mat & b */
    for (j=0; j<nlb; j++) {
        for (i=0; i<j; i++) a[i][j] = lcf;
        a[j][j] = bcoef;
        b[j] = bcoef + (double) j * lcf + (double) nub * ucf;
        for (i=j+1; i<j+1+nub; i++) a[i][j] = ucf;
        for (i=0; i<=nub+j; i++) icol[i][j] = i+1;
    }
    for (j=nlb; j<n-nub; j++) {
        for (i=0; i<nlb; i++) a[i][j] = lcf;
        a[nlb][j] = bcoef;
        b[j] = bcoef + (double) nlb * lcf + (double) nub * ucf;
        for (i=nlb+1; i<iwidt; i++) a[i][j] = ucf;
        for (i=0; i<iwidt; i++) icol[i][j] = i+1+j-nlb;
    }
    for (j=n-nub; j<n; j++){
        for (i=0; i<nlb; i++) a[i][j] = lcf;
        a[nlb][j] = bcoef;
        b[j] = bcoef + (double) nlb * lcf + (double) (n-j-1) * ucf;
        for (i=1; i<nub-2+n-j; i++) a[i+nlb][j] = ucf;
        ix = n - (j+nub-nlb-1);
        for (i=n; i>=j+nub-nlb-1; i--) icol[ix--][j] = i;
    }
    /* solve the nonsymmetric system of linear equations */
    itmax = 2000;
    iguss = 0;
    l = L;
    ierr = c_dvbcse ((double*)a, k, iwidt, n, (int*)icol, b, itmax,
                    eps, iguss, l, x, &iter, vw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dvbcse failed with icon = %d\n", icon);
        exit(1);
    }
    /* check result */
    for (i=0; i<n; i++)
        if (fabs(x[i]-one) > eps*10.0) {
            printf("WARNING: result maybe inaccurate\n");
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}

```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書 II”の VBCSE の項目及び[101], [112]を参照のこと。

c_dvbldl

正値対称バンド行列の LDL^T 分解 (変形コレスキー法)

<code>ierr = c_dvbldl(a, n, nh, epsz, &icon);</code>

1. 機能

$n \times n$ 上, 下バンド幅 h の正値対称バンド行列 **A** を変形コレスキー法によって LDL^T 分解する.

$$\mathbf{A} = \mathbf{L}\mathbf{D}\mathbf{L}^T$$

ただし, **L** は下バンド幅 h の単位下バンド行列, **D** は対角行列である. $0 \leq h < n$.

2. 引数

呼出し形式:

```
ierr = c_dvbldl(a, n, nh, epsz, &icon);
```

引数の説明:

a	double a[Alen]	入力	係数行列 A . 正値対称バンド行列用圧縮格納法. 格納法の詳細については“1.3.4 バンド行列の格納方法”を参照のこと. $Alen = n(h+1)$.
		出力	行列 D + (L - I). 正値対称バンド行列用圧縮格納法.
n	int	入力	行列 A の次数 n .
nh	int	入力	下バンド幅 h .
epsz	double	入力	ピボットの相対零判定値 (≥ 0). 0 のときは標準値が採用される. (“使用上の注意”a)参照)
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
10000	ピボットが負となった. 行列 A は正値ではない.	処理は続行する.
20000	ピボットが相対的に零となった. 行列 A は非正則の可能性が強い.	処理を打ち切る.
30000	次のいずれかであった. • $nh < 0$ 又は $nh \geq n$ • $epsz < 0$	処理を打ち切る.

3. 使用上の注意

a) epsz

epsz の標準値は丸め誤差の単位を μ としたとき 16μ である.

epsz よりピボットの値が小さいとき相対的に零と見なし, icon=20000 として処理を打ち切る.

b) icon

分解過程でピボットが負となった場合, 係数行列は正値ではない. 本関数では, このとき処理を続行するが icon=10000 とする.

4. 使用例

正値対称バンド行列を LDL^T 分解して, 連立 1 次方程式を解く. また, 行列式を計算する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define min(i,j) (i<j) ? i : j

#define NMAX 100
#define HMAX 2

MAIN__()
{
    int ierr, icon;
    int n, nh, i, j, imax, jmax;
    double epsz, det, eps, sum;
    double a[(HMAX+1)*NMAX], b[NMAX], x[NMAX];

    /* initialize matrix */
    n = NMAX;
    nh = HMAX;
    for (j=0;j<n;j++) {
        imax = min(j+nh,n-1);
        for (i=j;i<=imax;i++)
            a[j*(nh+1)+i-j] = n-(j-i);
    }
    for (i=0;i<n;i++) {
        x[i] = i+1;
        b[i] = 0;
    }
    /* initialize constant vector b = a*x */
    for (i=0;i<n;i++) {
        sum = a[i*(nh+1)]*x[i];
        jmax = min(i+nh,n-1);
        for (j=i+1;j<=jmax;j++) {
            b[j] = b[j] + a[i*nh+j]*x[i];
            sum = sum + a[i*nh+j]*x[j];
        }
        b[i] = b[i]+sum;
    }
    epsz = 1e-6;
    /* LDL decomposition of system of equations */
    ierr = c_dvbldl(a, n, nh, epsz, &icon);
    if (icon > 10000) {
        printf("ERROR: c_dvbldl failed with icon = %d\n", icon);
        exit(1);
    }
    /* calculate determinant */
    det = 1;
    for (i=0;i<n;i++) {
        det = det*a[i*(nh+1)];
    }
    printf("Determinant: %7.4e\n", det);
    /* solve decomposed system of equations */
    ierr = c_dvbldx(b, a, n, nh, &icon);
    if (icon > 10000) {
        printf("ERROR: c_dvbldx failed with icon = %d\n", icon);
        exit(1);
    }
    /* check solution vector */
    eps = 1e-6;
    for (i=0;i<n;i++)
        if (fabs((x[i]-b[i])/b[i]) > eps) {
            printf("WARNING: result inaccurate\n");
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書 II”の VBLDL の項目及び[79]を参照のこと.

c_dvbldx

LDL ^T 分解された正値対称バンド行列の連立 1 次方程式

ierr = c_dvbldx(b, fa, n, nh, &icon);

1. 機能

LDL^T分解された正値対称バンド行列を係数行列に持つ連立 1 次方程式

$$\mathbf{LDL}^T \mathbf{x} = \mathbf{b}$$

を解く。ただし $\mathbf{L}$, $\mathbf{D}$ はそれぞれ $n \times n$, 下バンド幅 h の単位下バンド行列, 対角行列, $\mathbf{b}$ は n 次元の実定数ベクトル, $\mathbf{x}$ は n 次元の解ベクトル. $n > h \geq 0$.

2. 引数

呼出し形式:

```
ierr = c_dvbldx(b, fa, n, nh, &icon);
```

引数の説明:

b	double b[n]	入力	定数ベクトル $\mathbf{b}$.
		出力	解ベクトル $\mathbf{x}$.
fa	double fa[Falen]	入力	行列 $\mathbf{D} + (\mathbf{L} - \mathbf{I})$. 正値対称バンド行列用圧縮格納法. 格納法の詳細については“1.3.4 バンド行列の格納方法”を参照のこと. $Falen = n(h+1)$.
n	int	入力	行列 $\mathbf{L}$ 及び $\mathbf{D}$ の次数 n .
nh	int	入力	下バンド幅 h .
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	係数行列が正値でなかった.	処理は続行する.
30000	次のいずれかであった. • $nh < 0$ 又は $nh \geq n$	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

本関数を c_dvbldl に続けて呼び出すことにより, 連立 1 次方程式を解くことができる. 本関数の引数 fa は c_dvbldl の a に対応している. しかし, 通常は c_dvlsbx を 1 回呼び出せば, 一度に解を求めることができる.

4. 使用例

正値対称バンド行列を LDL^T分解して, 連立 1 次方程式を解く. また, 行列式を計算する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define min(i,j) (i<j) ? i : j

#define NMAX 100
#define HMAX 2
```

```
MAIN__()
{
    int ierr, icon;
    int n, nh, i, j, imax, jmax;
    double epsz, det, eps, sum;
    double a[(HMAX+1)*NMAX], b[NMAX], x[NMAX];

    /* initialize matrix */
    n = NMAX;
    nh = HMAX;
    for (j=0;j<n;j++) {
        imax = min(j+nh,n-1);
        for (i=j;i<=imax;i++)
            a[j*(nh+1)+i-j] = n-(j-i);
    }
    for (i=0;i<n;i++) {
        x[i] = i+1;
        b[i] = 0;
    }
    /* initialize constant vector b = a*x */
    for (i=0;i<n;i++) {
        sum = a[i*(nh+1)]*x[i];
        jmax = min(i+nh,n-1);
        for (j=i+1;j<=jmax;j++) {
            b[j] = b[j] + a[i*nh+j]*x[i];
            sum = sum + a[i*nh+j]*x[j];
        }
        b[i] = b[i]+sum;
    }
    epsz = 1e-6;
    /* LDL decomposition of system of equations */
    ierr = c_dvbldl(a, n, nh, epsz, &icon);
    if (icon > 10000) {
        printf("ERROR: c_dvbldl failed with icon = %d\n", icon);
        exit(1);
    }
    /* calculate determinant */
    det = 1;
    for (i=0;i<n;i++) {
        det = det*a[i*(nh+1)];
    }
    printf("Determinant: %7.4e\n", det);
    /* solve decomposed system of equations */
    ierr = c_dvbldx(b, a, n, nh, &icon);
    if (icon > 10000) {
        printf("ERROR: c_dvbldx failed with icon = %d\n", icon);
        exit(1);
    }
    /* check solution vector */
    eps = 1e-6;
    for (i=0;i<n;i++)
        if (fabs((x[i]-b[i])/b[i]) > eps) {
            printf("WARNING: result inaccurate\n");
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書 II”の VBLDX の項目を参照のこと。

c_dvblu

実バンド行列の LU 分解 (ガウス消去法)
<pre>ierr = c_dvblu(a, n, nh1, nh2, epsz, &is, ip, vw, &icon);</pre>

1. 機能

$n \times n$, 下バンド巾 h_1 , 上バンド巾 h_2 のバンド行列をガウスの消去法により LU 分解する.

$$\mathbf{PA} = \mathbf{LU}$$

$\mathbf{P}$ は部分ピボットによる行の入替えを行う置換行列, $\mathbf{L}$ は単位下バンド行列, $\mathbf{U}$ は上バンド行列である. $0 \leq h_1 < n, 0 \leq h_2 < n$.

2. 引数

呼出し形式:

```
ierr = c_dvblu(a, n, nh1, nh2, epsz, &is, ip, vw, &icon);
```

引数の説明:

a	double a[Alen]	入力	係数行列 $\mathbf{A}$. バンド行列用圧縮格納法. 格納法の詳細については“1.3.4 バンド行列の格納方法”を参照のこと. $Alen = (2h_1 + h_2 + 1)n$.
		出力	行列 $(\mathbf{L} - \mathbf{I}) + \mathbf{U}$. バンド行列用圧縮格納法.
n	int	入力	行列 $\mathbf{A}$ の次数 n .
nh1	int	入力	行列 $\mathbf{A}$ の下バンド巾 h_1 .
nh2	int	入力	行列 $\mathbf{A}$ の上バンド巾 h_2 .
epsz	double	入力	ピボットの相対零判定値 (≥ 0). 0.0 のときは, 標準値が設定される. (“使用上の注意”a)参照)
is	int	出力	行列 $\mathbf{A}$ の行列式を求めるための情報. (“使用上の注意”b)参照)
ip	int ip[n]	出力	部分ピボットによる行の入替えの履歴を示すトランスポジションベクトル. (“使用上の注意”c)参照)
vw	double vw[n]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	行列 $\mathbf{A}$ のある行の要素がすべて零であったか, 又はピボットが相対的に零となった. 行列 $\mathbf{A}$ は非正則の可能性が強い.	処理を打ち切る.
30000	次のいずれかであった. - nh1 < 0 又は nh1 $\geq$ n - nh2 < 0 又は nh2 $\geq$ n - epsz < 0	処理を打ち切る.

3. 使用上の注意

a) epsz

epsz の標準値は丸め誤差の単位を μ としたとき 16μ である。

epsz よりピボットの値が小さいとき相対的に零と見なし, icon=20000 として処理を打ち切る。

b) 行列式の計算

行列式は, 配列 a に格納された行列 **U** の n 個の対角要素の積と is の値を掛け合わせて得られる。

c) ip

本ルーチンでは, 部分ピボットリングに伴い, 行列 **A** の内容を実際に交換している。すなわち, 分解の第 j 段階目 ($j=1, 2, \dots, n-1$) において第 i 行 ($i \geq j$) がピボット行として選択された場合には, 配列 a の第 i 行と第 j 行の内容が交換される。そして, その履歴を示すために ip[j-1] に i が格納される。

d) データ格納領域の節約

本関数では, バンド行列の特性を考慮して, データ格納領域を節約できるようにバンド行列を格納しているが, バンド幅の大きさによっては, c_dvalu よりデータ格納領域を多く必要とする場合がある。そのときは, c_dvalu を用いた方がデータ格納領域を節約できる。本関数の特性を生かせるのは, $n > 2 \times h_1 + h_2 + 1$ のときである。

4. 使用例

実バンド行列を LU 分解し, 連立 1 次方程式を解く。また, 行列式の値を求める。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define min(i,j) (i<j) ? i : j
#define max(i,j) (i>j) ? i : j

#define NMAX 100
#define H1MAX 2
#define H2MAX 2

MAIN__()
{
    int ierr, icon;
    int n, nh1, nh2, i, j, jmin, jmax, is, ip[NMAX];
    double epsz, det, eps, sum;
    double a[(2*H1MAX+H2MAX+1)*NMAX], b[NMAX], x[NMAX], vw[NMAX];

    /* initialize matrix */
    n = NMAX;
    nh1 = H1MAX;
    nh2 = H2MAX;
    for (i=0; i<n*(2*nh1+nh2+1); i++)
        a[i] = 0;
    for (i=0; i<n; i++) {
        jmin = max(i-nh1, 0);
        jmax = min(i+nh2, n-1);
        for (j=jmin; j<=jmax; j++)
            a[i*(2*nh1+1+nh2)+j-i+nh1] = n-fabs(j-i);
    }
    for (i=0; i<n; i++) {
        x[i] = i+1;
    }
    /* initialize constant vector b = a*x */
    for (i=0; i<n; i++) {
        jmin = max(i-nh1, 0);
        jmax = min(i+nh2, n-1);
        sum = 0;
        for (j=jmin; j<=jmax; j++)
            sum = sum + a[i*(2*nh1+1+nh2)+j-i+nh1]*x[j];
        b[i] = sum;
    }
}
```

```

    }
    epsz = 1e-6;
    /* LU decomposition of system of equations */
    ierr = c_dvblu(a, n, nh1, nh2, epsz, &is, ip, vw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dvblu failed with icon = %d\n", icon);
        exit(1);
    }
    /* calculate determinant */
    det = is;
    for (i=0;i<n;i++) {
        det = det*a[i*(2*nh1+1+nh2)+nh1];
    }
    printf("Determinant: %7.4e\n", det);
    /* solve decomposed system of equations */
    ierr = c_dvblux(b, a, n, nh1, nh2, ip, &icon);
    if (icon != 0) {
        printf("ERROR: c_dvblux failed with icon = %d\n", icon);
        exit(1);
    }
    /* check solution vector */
    eps = 1e-6;
    for (i=0;i<n;i++)
        if (fabs((x[i]-b[i])/b[i]) > eps) {
            printf("WARNING: result inaccurate\n");
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}

```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書 II”の VBLU の項目及び[42]を参照のこと。

c_dvblux

LU 分解された実バンド行列の連立 1 次方程式

<pre>ierr = c_dvblux(b, fa, n, nh1, nh2, ip, &icon);</pre>

1. 機能

$n \times n$, 下バンド巾 h_1 , 上バンド巾 h_2 のバンド行列をガウスの消去法により LU 分解した結果,

$$\mathbf{PA} = \mathbf{LU}$$

より, 前進代入及び後退代入で連立 1 次方程式を解く.

$$\mathbf{Ax} = \mathbf{b}$$

$\mathbf{P}$ は部分ピボットリングによる行の入替えを行う置換行列, $\mathbf{L}$ は単位下バンド行列, $\mathbf{U}$ は上バンド行列.
 $0 \leq h_1 < n, 0 \leq h_2 < n$.

2. 引数

呼出し形式:

```
ierr = c_dvblux(b, fa, n, nh1, nh2, ip, &icon);
```

引数の説明:

b	double b[n]	入力	定数ベクトル $\mathbf{b}$.
		出力	解ベクトル $\mathbf{x}$.
fa	double fa[Falen]	入力	行列 $(\mathbf{L} - \mathbf{I}) + \mathbf{U}$. バンド行列用圧縮格納法. 格納法の詳細については“1.3.4 バンド行列の格納方法”を参照のこと. $Falen = (2h_1 + h_2 + 1)n$
n	int	入力	行列 $\mathbf{A}$ の次数 n .
nh1	int	入力	行列 $\mathbf{A}$ の下バンド巾 h_1 .
nh2	int	入力	行列 $\mathbf{A}$ の上バンド巾 h_2 .
ip	int ip[n]	出力	部分ピボットリングによる行の入替えの履歴を示すトランスポジションベクトル. (“使用上の注意”参照)
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	係数行列が非正則であった.	処理を打ち切る.
30000	次のいずれかであった. - nh1 < 0 又は nh1 ≥ n - nh2 < 0 又は nh2 ≥ n - ip に誤りがあった.	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

連立 1 次方程式を解く場合, c_dvblu を呼び出してから, 本関数を呼び出せば方程式を解くことができる. このとき, 本関数の入力引数 (定数ベクトルは除く) は, c_dvblu の出力引数をそのまま指定すること.

4. 使用例

実バンド行列を LU 分解し, 連立 1 次方程式を解く. また, 行列式の値を求める.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define min(i,j) (i<j) ? i : j
#define max(i,j) (i>j) ? i : j

#define NMAX 100
#define H1MAX 2
#define H2MAX 2

MAIN__()
{
    int ierr, icon;
    int n, nh1, nh2, i, j, jmin, jmax, is, ip[NMAX];
    double epsz, det, eps, sum;
    double a[(2*H1MAX+H2MAX+1)*NMAX], b[NMAX], x[NMAX], vw[NMAX];

    /* initialize matrix */
    n = NMAX;
    nh1 = H1MAX;
    nh2 = H2MAX;
    for (i=0;i<n*(2*nh1+nh2+1);i++)
        a[i] = 0;
    for (i=0;i<n;i++) {
        jmin = max(i-nh1,0);
        jmax = min(i+nh2,n-1);
        for (j=jmin;j<=jmax;j++)
            a[i*(2*nh1+1+nh2)+j-i+nh1] = n-fabs(j-i);
    }
    for (i=0;i<n;i++) {
        x[i] = i+1;
    }
    /* initialize constant vector b = a*x */
    for (i=0;i<n;i++) {
        jmin = max(i-nh1,0);
        jmax = min(i+nh2,n-1);
        sum = 0;
        for (j=jmin;j<=jmax;j++)
            sum = sum + a[i*(2*nh1+1+nh2)+j-i+nh1]*x[j];
        b[i] = sum;
    }
    epsz = 1e-6;
    /* LU decomposition of system of equations */
    ierr = c_dvblu(a, n, nh1, nh2, epsz, &is, ip, vw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dvblu failed with icon = %d\n", icon);
        exit(1);
    }
    /* calculate determinant */
    det = is;
    for (i=0;i<n;i++) {
        det = det*a[i*(2*nh1+1+nh2)+nh1];
    }
    printf("Determinant: %7.4e\n", det);
    /* solve decomposed system of equations */
    ierr = c_dvblux(b, a, n, nh1, nh2, ip, &icon);
    if (icon != 0) {
        printf("ERROR: c_dvblux failed with icon = %d\n", icon);
        exit(1);
    }
    /* check solution vector */
    eps = 1e-6;
    for (i=0;i<n;i++)
        if (fabs((x[i]-b[i])/b[i]) > eps) {
            printf("WARNING: result inaccurate\n");
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書 II”の VBLUX の項目を参照のこと.

c_dvccvf

複素データの離散畳み込み及び離散相関

```
ierr = c_dvccvf(zx, k, n, m, zy, ivr, isw,
               tab, &icon);
```

1. 機能

複素数データの 1 次元, 多重の離散畳み込み又は離散相関をフーリエ変換を利用して計算する.

畳み込み

フィルターと, 1 本のデータをそれぞれ $\{y_i\}$ と $\{x_j\}$ としたとき, 以下で定義される畳み込みをデータ配列の各列に対して行い, $\{z_k\}$ を計算する.

$$z_k = \sum_{i=0}^{n-1} x_{k-i} y_i, \quad k = 0, \dots, n-1$$

相関

フィルターと, 1 本のデータをそれぞれ $\{y_i\}$ と $\{x_j\}$ としたとき, 以下で定義される相関をデータ配列の各列に対して行い, $\{z_k\}$ を計算する.

$$z_k = \sum_{i=0}^{n-1} x_{k+i} \overline{y_i}, \quad k = 0, \dots, n-1$$

ここで, x_j は周期 n の周期的データである.

2. 引数

呼出し形式:

```
ierr = c_dvccvf ((dcomplex*)zx, k, n, m, zy, ivr, isw, tab, &icon);
```

引数の説明:

zx	dcomplex zx[m][k]	入力	m 本の $\{x_j\}$ データを $zx[i][j]$, $i = 0, \dots, m-1, j = 0, \dots, n-1$ に格納する.
		出力	m 本の $\{z_k\}$ データが $zx[i][k]$, $i = 0, \dots, m-1, k = 0, \dots, n-1$ に格納される.
k	int	入力	データ配列 zx の 2 次元目の大きさ.
n	int	入力	1 本のデータの要素数 n . (“使用上の注意” b) 参照)
m	int	入力	多重度 m .
zy	dcomplex zy[n]	入力	フィルター $\{y_i\}$ を $zy[i]$, $i = 0, \dots, n-1$ に格納する.
			isw = 0 又は 2 のとき, 値は保存されない. (“使用上の注意” c), d) 参照)
ivr	int	入力	畳み込みと相関の処理を選択する.
			0 畳み込みを計算する.
			1 相関を計算する.
isw	int	入力	制御情報.
			0 1 回の呼出しで処理を完了する.
			以下の isw 指定で順次呼び出すことで, 処理のステップを分けて利用することができる. (“使用上の注意” c) 参照)

			1 tab 設定のみ行う.
			2 zy のフーリエ変換を行う.
			3 設定済みの tab と変換済みの zy を使用して畳み込み又は相関を計算する.
tab	double tab[2×n]	作業領域	三角関数表を格納する. ("使用上の注意" c) 参照)
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード：

コード	意味	処理内容
0	エラーなし	正常終了.
30000	次のいずれかであった. • $n \leq 0$ • $k < n$ • $m \leq 0$ • $isw \neq 0, 1, 2, 3$ • $ivr \neq 0, 1$	処理を打ち切る.

3. 使用上の注意

a) 非周期的データに対する畳み込みおよび相関について

データ x の長さが n_x , y の長さが n_y であるとき, $n \geq n_x + n_y - 1$ として $zx[i][j]$, $i = 0, \dots, m-1$, $j = n_x, \dots, n-1$ および $zy[k]$, $k = n_y, \dots, n-1$ には 0 を設定しておくことで, 非同期データに対する畳み込み及び相関が計算できる. ("使用例" b) 参照)

このとき, 相関の $z_k(k = -n_y + 1, \dots, -1)$ に相当する値は, $zx[i][j]$, $i = 0, \dots, m-1$, $j = n - n_y + 1, \dots, n-1$, に格納される.

b) n の値について

1本のデータの要素 n は任意の数が指定可能であるが, 2, 3, 5 の因子に分解される場合が高速である.

c) tab 及び zy の再利用について

同じ n で本関数を繰り返し呼び出す場合, 初回に $isw = 0$ 又は 1 で呼び出した後, 2 回目以降は $isw = 2$ と 3 で呼び出すことにより tab 作成を省略し, 処理の高速化ができる. さらに, 同じフィルタ y を使用する場合には, $isw = 0$ 又は 2 で変換された zy を $isw = 3$ の呼び出しで利用することで処理が効率化できる.

このように本関数を繰り返し呼び出す場合, zy に対して 2 回以上の変換が行われないように注意が必要である.

d) 自己相関計算について

引数 zx と zy に同一の実引数を指定することで, 自己相関の計算が可能である. このとき, $isw = 2$ の指定は無視される. ("使用例" c) 参照)

e) スタックサイズについて

本関数では, 内部で使用する作業域をスタック域に確保している. このため, スタックが不足すると異常終了する可能性がある. 本関数がスタック域に必要なサイズは, $16 \times n$ バイトである.

この他に, 固定サイズの作業域や, ユーザプログラムで必要とするスタック域もあるので, limit コマンド又は ulimit コマンドなどで十分な大きさのスタックサイズを指定することを勧める.

4. 使用例

a) 周期的データに対する畳み込みを計算する.

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

```

#include <stdlib.h>
#include "cssl.h" /* standard C-SSL header file */

#define K 8
#define M 3

int MAIN__(void)
{
    dcomplex zx[M][K], zy[K];
    double   tab[K*2];
    int      i, j, n;
    int      ivr, isw, icon;

    n=K;

    for (j=0; j<M; j++) {
        for (i=0; i<n; i++) {
            zx[j][i].re = i+j+1;
            zx[j][i].im = i-j;
        }
    }

    for (i=0; i<n; i++) {
        zy[i].re = (i+1)*(i+1);
        zy[i].im = 9-i;
    }

    printf("--INPUT DATA--\n");

    for (j=0; j<M; j++) {
        printf("zx[%d][*] : ", j);
        for (i=0; i<n; i++) {
            if(i%4==0) printf("\n");
            printf("(%8.2f,%8.2f) ", zx[j][i].re, zx[j][i].im);
        }
        printf("\n");
    }

    printf("Filter zy : ");
    for (i=0; i<n; i++) {
        if(i%4==0) printf("\n");
        printf("(%8.2f,%8.2f) ", zy[i].re, zy[i].im);
    }

    ivr = 0;
    isw = 0;
    c_dvccvf((dcomplex*)zx, K, n, M, zy, ivr, isw, tab, &icon);

    printf("\n\n--OUTPUT DATA--\n");
    for (j=0; j<M; j++) {
        printf("zx[%d][*] : ", j);
        for (i=0; i<n; i++) {
            if(i%4==0) printf("\n");
            printf("(%8.2f,%8.2f) ", zx[j][i].re, zx[j][i].im);
        }
        printf("\n");
    }
}

```

b) 非周期的データに対する畳み込みを計算する.

```

#include <stdio.h>
#include <stdlib.h>
#include "cssl.h" /* standard C-SSL header file */

#define K 16
#define M 3

int MAIN__(void)
{
    dcomplex zx[M][K], zy[K];
    double   tab[K*2];
    int      i, j, n, nx, ny;
    int      ivr, isw, icon;

    nx=7, ny=9, n=nx+ny-1;

```

```
    if(n%2) n=n+1;

    for (j=0; j<M; j++) {
        for (i=0; i<nx; i++) {
            zx[j][i].re = i+j+1;
            zx[j][i].im = i-j;
        }
        for (i=nx; i<n; i++) {
            zx[j][i].re = 0.0;
            zx[j][i].im = 0.0;
        }
    }

    for (i=0; i<ny; i++) {
        zy[i].re = (i+1)*(i+1);
        zy[i].im = 9-i;
    }
    for (i=ny; i<n; i++) {
        zy[i].re = 0.0;
        zy[i].im = 0.0;
    }

    printf("--INPUT DATA--\n");

    for (j=0; j<M; j++) {
        printf("zx[%d][*] : ",j);
        for (i=0; i<n; i++) {
            if(i%4==0) printf("\n          ");
            printf("(%8.2f,%8.2f) ",zx[j][i].re, zx[j][i].im);
        }
        printf("\n");
    }

    printf("Filter zy : ");
    for (i=0; i<n ; i++) {
        if(i%4==0) printf("\n          ");
        printf("(%8.2f,%8.2f) ", zy[i].re, zy[i].im);
    }

    ivr = 0;
    isw = 0;
    c_dvccvf((dcomplex*)zx, K, n, M, zy, ivr, isw, tab, &icon);

    printf("\n\n--OUTPUT DATA--\n");
    for (j=0; j<M; j++) {
        printf("zx[%d][*] : ",j);
        for (i=0; i<n; i++) {
            if(i%4==0) printf("\n          ");
            printf("(%8.2f,%8.2f) ",zx[j][i].re, zx[j][i].im);
        }
        printf("\n");
    }
}
```

c) 自己相関を計算する.

```
#include <stdio.h>
#include <stdlib.h>
#include "cssl.h" /* standard C-SSL header file */

#define K 8
#define M 3

int MAIN__(void)
{
    dcomplex zx[M][K];
    double   tab[K*2];
    int      i, j, n, nx;
    int      ivr, isw, icon;

    nx=4, n=nx*2;

    for (j=0; j<M; j++) {
        for (i=0; i<nx; i++) {
            zx[j][i].re = i+j+1;
            zx[j][i].im = i-j;
        }
    }
}
```

```

    }
    for (i=nx; i<n; i++) {
        zx[j][i].re = 0.0;
        zx[j][i].im = 0.0;
    }
}

printf("--INPUT DATA--\n");

for (j=0; j<M; j++) {
    printf("zx[%d][*] : ", j);
    for (i=0; i<n; i++) {
        if(i%4==0) printf("\n          ");
        printf("(%8.2f,%8.2f) ", zx[j][i].re, zx[j][i].im);
    }
    printf("\n");
}

ivr = 1;
isw = 1;
c_dvccvf((dcomplex*)zx, K, n, M, (dcomplex*)zx, ivr, isw, tab, &icon);
isw=3;
c_dvccvf((dcomplex*)zx, K, n, M, (dcomplex*)zx, ivr, isw, tab, &icon);

printf("\n--OUTPUT DATA--\n");
for (j=0; j<M; j++) {
    printf("zx[%d][*] : ", j);
    for (i=0; i<n; i++) {
        if(i%4==0) printf("\n          ");
        printf("(%8.2f,%8.2f) ", zx[j][i].re, zx[j][i].im);
    }
    printf("\n");
}
}
}

```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書”の VCCVF の項目を参照のこと。

c_dvcfm1

1次元離散型複素フーリエ変換 (2, 3, 5 及び 7 の混合基底)

ierr = c_dvcfm1(x, n, &isw, isn, w, &icon);

1. 機能

1次元複素フーリエ変換の正変換又は逆変換を行う。変換データ長 n は、2, 3, 5, 7 の中の積として表される数でなければならない。

1次元フーリエ変換

$\{x_j\}$ を入力し、次の式で定義する変換を行い、 $\{na_k\}$ を求める。

$$na_k = \sum_{j=0}^{n-1} x_j \cdot \omega_n^{-jk}, \quad k = 0, 1, \dots, n-1$$

$$\omega_n = \exp(2\pi i / n)$$

1次元フーリエ逆変換

$\{a_k\}$ を入力し、次の式で定義する変換を行い、 $\{x_j\}$ を求める。

$$x_j = \sum_{k=0}^{n-1} a_k \cdot \omega_n^{jk}, \quad j = 0, 1, \dots, n-1$$

$$\omega_n = \exp(2\pi i / n)$$

2. 引数

呼出し形式:

```
ierr = c_dvcfm1(x, n, &isw, isn, w, &icon);
```

引数の説明:

x	dcomplex x[n]	入力	複素数データ.
n	int	入力	変換データ長 n .
isw	int	入力	制御情報.
			isw = 1 初回呼出し. w に三角関数テーブル及び制御情報を格納しフーリエ変換を行う.
			isw ≠ 1 同じ大きさのデータの変換を初回以降引き続き行う. このとき、初回に w に格納された三角関数テーブル及び制御情報を利用するため、 n, w の内容を初回呼び出し後に変更してはならない.
		出力	isw = 1 を設定して呼び出したとき、isw = 0 が設定される. このため 2 回目以降特に isw に値を設定せずに、配列 x に変換データを格納して呼び出すことで効率よく変換を行うことができる.
isn	int	入力	変換か逆変換かを指定する(≠ 0)
			変換 isn = 1
			逆変換 isn = -1
w	dcomplex w[2×n+70]	作業領域	isw = 1 のとき、データ長 n の変換用の三角関数テーブルが格納される.
			isw ≠ 1 のとき、isw = 1 を指定した初回呼び出しで作成された三角関数テーブルを利用する. (“使用上の注意” b) 参照)

icon int 出力 コンディションコード. 下の表を参照.

コンディションコード :

コード	意 味	処 理 内 容
0	エラーなし	正常終了.
20000	2 回目以降の呼び出しで n の値が初回呼び出しのときの値と異なる.	処理を打ち切る.
30000	isn の値が正しくない.	
30008	n の変換数が 2, 3, 5, 7 を基底としていない.	

3. 使用上の注意

a) 一般的なフーリエ変換の定義

1次元離散型複素フーリエ変換及び, 逆変換は一般的に(1), (2)で定義される.

$$a_k = \frac{1}{n} \cdot \sum_{j=0}^{n-1} x_j \cdot \omega_n^{-jk}, \quad k = 0, 1, \dots, n-1 \quad (1)$$

$$x_j = \sum_{k=0}^{n-1} a_k \cdot \omega_n^{jk}, \quad j = 0, 1, \dots, n-1 \quad (2)$$

ここで $\omega = \exp(2\pi i / n)$.

本関数は(1), (2)の左辺に対応して $\{na_k\}$ 又は $\{x_j\}$ を求める. したがって, 結果の正規化は必要に応じて行うこと.

b) 作業領域 w について

同じ n で複数回本関数を呼び出す場合, 初回に isw=1 で呼び出した後, 2 回目以降は isw ≠ 1 で呼び出すことにより, 配列 w 内に作成した三角関数テーブルが再利用されます. 配列 w は作業領域としても利用されるため, 2 回目以降の呼び出しであっても書き込みが行われることに注意してください.

4. 使用例

1次元 FFT を計算する.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define N 640
#define max(a,b) ((a) > (b) ? (a) : (b))

int MAIN__(void)
{
    dcomplex x[N], w[N*2+70], tmp;
    double error;
    int isw, isn, icon, i;

    for (i=0; i<N; i++) {
        x[i].re=(double)(i+1)/(double)N;
        x[i].im=0.0;
    }

    /* do the forward transform */
    isw=1, isn=1;
    c_dvcfm1(x, N, &isw, isn, w, &icon);

    if (icon != 0) {
        printf("icon = %d", icon);
        exit(1);
    }

    /* do the reverse transform */
    isn=-1;
}
```

```
c_dvcfm1(x, N, &isw, isn, w, &icon);

if (icon != 0) {
    printf("icon = %d", icon);
    exit(1);
}

error = 0.0;
for (i=0; i<N; i++) {
    tmp.re = fabs(x[i].re/(double)N - (double)(i+1)/(double)N);
    tmp.im = fabs(x[i].im/(double)N);
    tmp.re += tmp.im;
    error=max(error,tmp.re);
}

printf("error = %e\n", error);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書”の VCFM1 の項目を参照のこと。

c_dvcft1

離散型複素フーリエ変換(2 基底 FFT)

```
ierr = c_dvcft1(a, b, n, isn, isw, vw, ivw,
               &icon);
```

1. 機能

1 次元 (項数 n) の複素時系列データ $\{x_j\}$ が与えられたとき, 離散型複素フーリエ変換, 又はその逆変換をベクトル計算機向きの高速変換手法 (FFT) により計算する. ただし, $n = 2^\ell$ ($\ell: 0$ 又は正整数) であること.

フーリエ変換

$\{x_j\}$ を入力し, 次の式で定義する変換を行い, $\{na_k\}$ を求める.

$$na_k = \sum_{j=0}^{n-1} x_j \cdot \omega^{-jk}, \quad k = 0, 1, \dots, n-1$$

ここで, $\omega = e^{2\pi i / n}$.

フーリエ逆変換

$\{a_k\}$ を入力し, 次の式で定義する変換を行い, $\{x_j\}$ を求める.

$$x_j = \sum_{k=0}^{n-1} a_k \cdot \omega^{jk}, \quad j = 0, 1, \dots, n-1$$

ここで, $\omega = e^{2\pi i / n}$.

2. 引数

呼出し形式:

```
ierr = c_dvcft1(a, b, n, isn, isw, vw, ivw, &icon);
```

引数の説明:

a	double a[n]	入力	$\{x_j\}$ 又は $\{a_k\}$ の実部.
		出力	$\{na_k\}$ 又は $\{x_j\}$ の実部.
b	double b[n]	入力	$\{x_j\}$ 又は $\{a_k\}$ の虚部.
		出力	$\{na_k\}$ 又は $\{x_j\}$ の虚部.
n	int	入力	変換の項数 n .
isn	int	入力	変換か逆変換かを指定する ($\neq 0$) 変換 isn = 1 逆変換 isn = -1 (“使用上の注意” c) 参照
isw	int	入力	変換の初期設定を制御する情報. 0 初回呼出しのとき 1 継続呼出しのとき (“使用上の注意” b) 参照
vw	double vw[Rlen]	作業領域	$Rlen = \max(n \cdot \ell, 1)$.

ivw int ivw[l_{len}] 作業領域 $l_{len} = n \cdot \max(\ell - 3, 2)$.
 icon int 出力 コンディションコード. 下の表を参照.

コンディションコード :

コード	意味	処理内容
0	エラーなし	正常終了.
30000	次のいずれかであった. • isn = 0 • isw ≠ 0 又は 1 • $n \neq 2^\ell$ ($\ell:0$ 又は正整数)	処理を打ち切る.

3. 使用上の注意

a) 本関数の特徴

本関数は, 複素フーリエ変換をベクトル計算機上で高速処理する.

b) isw による制御

複数組の変換を行う場合は, 2 回目以降の呼出しのとき isw = 1 と指定すると, 変換に必要な三角関数表やリストベクトルなどの生成を省略するので, 効率良く処理される. このとき, 配列 vw, ivw の内容は変更せずに呼び出すこと.

複数組の変換の項数 n が各々異なる場合でも, isw = 1 の指定は有効である. しかし, 可能な限り同一の項数の変換が連続するように呼び出すことが望ましい.

実フーリエ変換の関数 c_dvrft1 と組み合わせて本関数を呼び出す場合でも, isw = 1 の指定は有効である.

c) isn の与え方

isn では, 変換か逆変換かを指定するが, 更に次のように使い分けることができる. すなわち, $\{x_j\}$ 又は $\{a_k\}$ の実部, 虚部が, 共に間隔 i で格納されている場合は, 次のように指定する.

変換 : isn = +i
 逆変換 : isn = -i

この場合, 計算結果も間隔 i で格納される. ただし, $i > 1$ の場合は, 作業領域 vw の大きさは $n \cdot (\ell + 2)$ とすること.

ベクトル計算機で実行する場合, メモリアクセスを効率良くするために, 間隔 i は $i = 2 \cdot p + 1$, $p = 1, 2, 3, \dots$ であることが望ましい.

d) 作業用配列の大きさ早見表

$16 \leq n \leq 4096$ の場合の大きさを以下に示す.

ℓ	n	vw の大きさ	ivw の大きさ
4	16	64 (96)	32
5	32	160 (224)	64
6	64	384 (512)	192
7	128	896 (1152)	512
8	256	2048 (2560)	1280
9	512	4608 (5632)	3072
10	1024	10240 (12288)	7168
11	2048	22528 (26624)	16384
12	4096	49152 (57344)	36864

()内は $|isn| > 1$ のときの大きさ.

e) 一般的なフーリエ変換の定義

離散型複素フーリエ変換及び、逆変換は一般的に(1), (2)で定義される.

$$a_k = \frac{1}{n} \cdot \sum_{j=0}^{n-1} x_j \cdot \omega^{-jk}, \quad k = 0, 1, \dots, n-1 \quad (1)$$

$$x_j = \sum_{k=0}^{n-1} a_k \cdot \omega^{jk}, \quad j = 0, 1, \dots, n-1 \quad (2)$$

ここで $\omega = e^{2\pi i / n}$.

この関数は(1), (2)の左辺に対応して $\{na_k\}$ 又は $\{x_j\}$ を求める. したがって, 結果の正規化は必要に応じて行うこと.

4. 使用例

1次元フーリエ変換を求める. 入力は乱数により与える. 変換後, さらに逆変換して結果が正しいか確認する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 1024

MAIN__()
{
    int ierr, icon;
    double phai, ran, eps;
    double a[NMAX], b[NMAX], aa[NMAX], bb[NMAX], vw[NMAX*10];
    int i, n, isw, isn, ivw[NMAX*(10-3)];

    /* generate initial data */
    n = NMAX;
    phai = (sqrt(5.0)-1.0)/2;
    for (i=0; i<n; i++) {
        ran = (i+1)*phai;
        a[i] = ran - (int)ran;
        ran = (i+n+1)*phai;
        b[i] = ran - (int)ran;
    }
    for (i=0; i<n; i++) {
        aa[i] = a[i];
        bb[i] = b[i];
    }
    /* perform normal transform */
    isw = 0;
    isn = 1;
    ierr = c_dvcft1(a, b, n, isn, isw, vw, ivw, &icon);
    /* perform inverse transform */
    isw = 1;
    isn = -1;
    ierr = c_dvcft1(a, b, n, isn, isw, vw, ivw, &icon);
    /* check results */
    eps = 1e-6;
    for (i=0; i<n; i++)
        if ((fabs((a[i]/n - aa[i])/aa[i]) > eps) ||
            (fabs((b[i]/n - bb[i])/bb[i]) > eps)) {
            printf("Inaccurate result\n");
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書”の VCFT1 の項目及び[110], [135], [136], [137]を参照のこと.

c_dvcft2

離散型複素フーリエ変換(メモリ節約型, 2 基底 FFT)

```
ierr = c_dvcft2(a, b, n, isn, isw, vw, ivw,
                &icon);
```

1. 機能

1 次元(項数 n)の複素時系列データ $\{x_j\}$ が与えられたとき, 離散型複素フーリエ変換, 又はその逆変換をベクトル計算機向きの高速変換手法(FFT)により計算する. ただし, $n=2^l$ ($l:0$ 又は正整数)であること.

a) フーリエ変換

$\{x_j\}$ を入力し, 次の式で定義する変換を行い, $\{na_k\}$ を求める.

$$na_k = \sum_{j=0}^{n-1} x_j \omega^{-jk}, \quad k = 0, 1, \dots, n-1,$$

$$\omega = e^{2\pi i / n}$$

b) フーリエ逆変換

$\{a_k\}$ を入力し, 次の式で定義する変換を行い, $\{x_j\}$ を求める.

$$x_j = \sum_{k=0}^{n-1} a_k \omega^{jk}, \quad j = 0, 1, \dots, n-1,$$

$$\omega = e^{2\pi i / n}$$

2. 引数

呼出し形式:

```
ierr = c_dvcft2(a, b, n, isn, isw, vw, ivw, &icon);
```

引数の説明:

a	double a[n]	入力	$\{x_j\}$ 又は $\{a_k\}$ の実部.
		出力	$\{na_k\}$ 又は $\{x_j\}$ の実部.
b	double b[n]	入力	$\{x_j\}$ 又は $\{a_k\}$ の虚部.
		出力	$\{na_k\}$ 又は $\{x_j\}$ の虚部.
n	int	入力	変換の項数 n .
isn	int	入力	変換か逆変換かを指定する ($isn \neq 0$). isn = 1 なら変換. isn = -1 なら逆変換. ("使用上の注意"b)参照)
isw	int	入力	変換の初期設定を制御する情報. isw = 0 なら初回呼出し. isw = 1 なら継続呼出し. ("使用上の注意"c)参照)
vw	double vw[5n]	作業領域	
ivw	int ivw[3n]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
30000	次のいずれかであった. • isn = 0 • isw ≠ 0 又は 1 • $n \neq 2^l$ ($l:0$ 又は正整数)	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

本関数は, 単独の変換を行う場合に適している. 作業用配列は, 最小限におさえであり, 言わば, 「メモリ節約型」の関数である. 複数组の変換を行う場合で, 作業用配列を十分に確保できるならば, 「性能優先型」の関数 c_dvcft1 が適している.

b) isn

isn では, 変換か逆変換かを指定するが, 更に次のように使い分けることができる. すなわち, $\{x_j\}$ 又は $\{a_k\}$ の実部, 虚部が, 共に間隔 i で格納されている場合は, 次のように指定する.

変換 : isn = +i

逆変換 : isn = -i

この場合, 計算結果も間隔 i で格納される. ただし $i > 1$ の場合は, 作業領域 vw の大きさは $7n$ とすること. ベクトル計算機で実行する場合, メモリアクセスを効率良くするために, 間隔 i は $i = 2p + 1, p = 1, 2, 3, \dots$ を満たす値であることが望ましい.

c) isw

複数组の変換を行う場合, 2 回目以降の呼出しのとき isw = 1 と指定すると, 変換に必要な三角関数表などの生成を省略するので, 多少効率良く処理される. このとき, 配列 vw, ivw の内容は変更せず呼び出すこと.

複数组の変換の項数 n が各々異なる場合でも, isw = 1 の指定は有効である. しかし, 可能な限り同一の項数の変換が連続するように呼び出すことが望ましい. 実フーリエ変換の関数 c_dvrft2 と組み合わせて本関数を呼び出す場合でも isw = 1 の指定は有効である.

d) 作業用配列の大きさ早見表

$16 \leq n \leq 4096$ の場合の大きさを以下に示す. () 内は, $|isn| > 1$ のときの大きさ.

l	n	vw	ivw
4	16	80 (112)	48
5	32	160 (224)	96
6	64	320 (448)	192
7	128	640 (896)	384
8	256	1280 (1792)	768
9	512	2560 (3584)	1536
10	1024	5120 (7168)	3072
11	2048	10240 (14336)	6144
12	4096	20480 (28672)	12288

e) 一般的なフーリエ変換の定義

離散型複素フーリエ変換及び逆変換は一般的に(1), (2)で定義される.

$$a_k = \frac{1}{n} \sum_{j=0}^{n-1} x_j \omega^{-jk}, \quad k = 0, 1, \dots, n-1, \quad (1)$$

$$x_j = \sum_{k=0}^{n-1} a_k \omega^{jk}, \quad j = 0, 1, \dots, n-1, \quad (2)$$

ここで, $\omega = e^{2\pi i / n}$.

本関数では, (1), (2)の左辺に対応して, $\{a_k\}$ 又は $\{x_j\}$ を求める. したがって, 結果の正視化は必要に応じて行うこと.

4. 使用例

1次元のフーリエ変換及びその逆変換を連続して計算する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 1024

MAIN__()
{
    int ierr, icon;
    double phai, ran, eps;
    double a[NMAX], b[NMAX], aa[NMAX], bb[NMAX], vw[NMAX*5];
    int i, n, isw, isn, ivw[NMAX*3];

    /* generate initial data */
    n = NMAX;
    phai = (sqrt(5.0)-1.0)/2;
    for (i=0; i<n; i++) {
        ran = (i+1)*phai;
        a[i] = ran - (int)ran;
        ran = (i+n+1)*phai;
        b[i] = ran - (int)ran;
    }
    for (i=0; i<n; i++) {
        aa[i] = a[i];
        bb[i] = b[i];
    }
    /* perform normal transform */
    isw = 0;
    isn = 1;
    ierr = c_dvcft2(a, b, n, isn, isw, vw, ivw, &icon);
    /* perform inverse transform */
    isw = 1;
    isn = -1;
    ierr = c_dvcft2(a, b, n, isn, isw, vw, ivw, &icon);
    /* check results */
    eps = 1e-6;
    for (i=0; i<n; i++)
        if ((fabs((a[i]/n - aa[i])/aa[i]) > eps) ||
            (fabs((b[i]/n - bb[i])/bb[i]) > eps)) {
            printf("Inaccurate result\n");
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書”の VCFT2 の項目を参照のこと.

c_dvcft3

1次元離散型複素フーリエ変換(2基底, 一定間隔を持つデータ)

ierr = c_dvcft3(x, n, ndist, &isw, isn, w, &icon);

1. 機能

1次元複素フーリエ変換の正変換または逆変換を行う.変換データ長 n は,2 の巾として表される数でなければならない.

a) 1次元フーリエ変換

$\{x_j\}$ を入力し, 次の式で定義する変換を行い, $\{na_k\}$ を求める.

$$na_k = \sum_{j=0}^{n-1} x_j \omega_n^{-jk}, \quad k = 0, 1, \dots, n-1,$$

$$\omega_n = \exp(2\pi i / n)$$

b) 1次元フーリエ逆変換

$\{a_k\}$ を入力し, 次の式で定義する変換を行い, $\{x_j\}$ を求める.

$$x_j = \sum_{k=0}^{n-1} a_k \omega_n^{jk}, \quad j = 0, 1, \dots, n-1,$$

$$\omega_n = \exp(2\pi i / n)$$

2. 引数

呼出し形式:

```
ierr = c_dvcft3(x, n, ndist, &isw, isn, w, &icon);
```

引数の説明:

x	dcomplex x[(n-1)*ndist+1]	入力	複素数データ.変換するデータ $\{x_j\}$ または $\{a_k\}$ を, x[0], x[ndist], ..., x[(n-1)*ndist] に格納する.
		出力	複素数データ.変換された結果 $\{na_k\}$ または $\{x_j\}$ が, x[0], x[ndist], ..., x[(n-1)*ndist] に格納される. 複素数型 1次元配列.
n	int	入力	変換データ長(n).
ndist	int	入力	配列 x にデータを格納する一定の間隔.正の整数.ndist=1 のとき,連続に格納される.
isw	int	入力	制御情報. isw = 1 のとき:初回呼び出し.w に三角関数テーブルおよび制御情報を格納しフーリエ変換を行う. isw ≠ 1 のとき:同じ大きさのデータの変換を初回以降引き続き行う.このとき,初回に w に格納された三角関数テーブルおよび制御情報を利用するため,n, w の内容を初回呼び出し後に変更してはならない.
		出力	isw=1 を設定して呼び出したとき,isw=0 が設定される.このため 2 回目以降特に isw に値を設定せずに,配列 x に変換データを格納して呼び出すことで効率よく変換を行うことができる.

isn	int	入力	変換か逆変換かを指定. 変換 : isn=1 逆変換 : isn=-1
w	dcomplex w[2n+70]	作業領域	isw=1 のとき, データ長 n の変換用の三角関数テーブルが格納される. isw≠1 のとき, isw=1 を指定した初回呼び出しで作成された三角関数テーブルを利用する. (“使用上の注意” b) 参照)
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	2 回目以降の呼び出しで n の値が初回呼び出しのときの値と異なる.	処理を打ち切る.
30000	isn の値が正しくない. ndist が正の整数ではない.	
30008	n の変換数が 2 を基底としていない.	

3. 使用上の注意

a) 一般的なフーリエ変換の定義

1 次元離散型複素フーリエ変換および逆変換は一般的に(1), (2)で定義される.

$$a_k = \frac{1}{n} \sum_{j=0}^{n-1} x_j \omega_n^{-jk}, \quad k = 0, 1, \dots, n-1, \quad (1)$$

$$x_j = \sum_{k=0}^{n-1} a_k \omega_n^{jk}, \quad j = 0, 1, \dots, n-1, \quad (2)$$

ここで, $\omega_n = \exp(2\pi i / n)$.

本関数では, (1), (2)の左辺に対応して, $\{na_k\}$ 又は $\{x_j\}$ を求める. したがって, 結果の正視化は必要に応じて行うこと.

b) 作業領域 w について

同じ n で複数回本関数を呼び出す場合, 初回に isw=1 で呼び出した後, 2 回目以降は isw ≠ 1 で呼び出すことにより, 配列 w 内に作成した三角関数テーブルが再利用されます. 配列 w は作業領域としても利用されるため, 2 回目以降の呼び出しであっても書き込みが行われることに注意してください.

4. 使用例

複数の一定の間隔をもったデータ列に対して, 1 次元 FFT を計算する.

```
#include <stdio.h>
#include <math.h>
#include "cssl.h"

#define N      1024
#define MULT   16
#define NPAD   3
#define NDIST (MULT+NPAD)
```

```
#define max(a,b) ((a) > (b) ? (a) : (b))

MAIN__()
{
    dcomplex x[N][NDIST],w[2*N+70];
    int i,j;
    int isw,icon,ierr;
    double tmp;

    for(j=0;j<MULT;j++)
        for(i=0;i<N;i++) {
            x[i][j].re=i/((double)N)+j;
            x[i][j].im=0.0;
        }
    /*
     * multiple forward transform
     */
    isw=1;
    for(j=0;j<MULT;j++) {
        ierr=c_dvcft3((dcomplex*)&x[0][j],N,NDIST,&isw,1,w,&icon);
        if(icon!=0) printf("icon=%d\n",icon);
    }
    /*
     * multiple reverse transform
     */
    for(j=0;j<MULT;j++) {
        ierr=c_dvcft3((dcomplex*)&x[0][j],N,NDIST,&isw,-1,w,&icon);
        if(icon!=0) printf("icon=%d\n",icon);
    }

    tmp=0.0;
    for(j=0;j<MULT;j++) {
        for(i=0;i<N;i++) {
            tmp=max(tmp,fabs(x[i][j].re/N-(i/((double)N)+j))+fabs(x[i][j].im/N));
        }
    }
    printf("error = %le\n",tmp);
    return 0;
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書”の VCFT3 の項目を参照のこと。

c_dvcgd

正値対称スパース行列の連立 1 次方程式
(前処理付き CG 法, 対角形式格納法)

```
ierr = c_dvcgd(a, k, nw, n, ndlt, b, ipc,
               itmax, isw, omega, eps, iguss, x,
               &iter, &rz, vw, ivw, &icon);
```

1. 機能

$n \times n$ の正規化された正値対称なスパース行列を係数行列する連立 1 次方程式(1)を前処理付き CG 法で解く.

$$\mathbf{Ax} = \mathbf{b} \quad (1)$$

係数行列は, 対角要素を除く非零要素を対角形式の格納法で格納する. ベクトル $\mathbf{b}$ 及び $\mathbf{x}$ は n 次元ベクトル.

2. 引数

呼出し形式:

```
ierr = c_dvcgd((double*)a, k, nw, n, ndlt, b, ipc, itmax, isw, omega, eps,
               iguss, x, &iter, &rz, vw, ivw, &icon);
```

引数の説明:

a	double a[nw][k]	入力	正規化された正値対称スパース行列の係数行列の非零要素を対角形式で格納する. (“使用上の注意” a)参照). ipc = 3 のとき, 演算後, 内容は保存されない.
k	int	入力	配列 a の整合寸法 ($\geq n$).
nw	int	入力	配列 a の 1 次元目の大きさ. 対角形式の格納方法で係数行列 $\mathbf{A}$ を格納する対角方向のベクトルの本数. 偶数.
n	int	入力	行列 $\mathbf{A}$ の次数 n .
ndlt	int ndlt[nw]	入力	主対角ベクトルからの距離を示す. (“使用上の注意” a)参照).
b	double b[n]	入力	連立 1 次方程式の右辺の定数ベクトルを格納する.
ipc	int	入力	前処理の制御情報. (“使用上の注意” d)参照). 1 前処理なし. 2 Neumann の前処理. 3 不完全コレスキー分解による前処理. このとき omega を指定する必要がある.
itmax	int	入力	反復回数の上限 (> 0).
isw	int	入力	制御情報. (“使用上の注意” b)参照). 1 初回の呼出し. 2 2 回目以降の呼出し. ただし a, ndlt, vw, ivw の値は初回呼び出しで設定された値を使用するため変更してはならない.
omega	double	入力	不完全コレスキー分解のための修正値. $0 \leq \omega \leq 1$. ipc = 3 のとき使用される. (“使用上の注意” d)参照).
eps	double	入力	収束判定に用いられる判定値. $\text{rz} < \text{eps}$ を満たしたとき収束したと見なす. 0.0 以下のときは, $\text{eps} = \varepsilon \cdot \ \mathbf{b}\ $ として設定される. $\varepsilon = 10^{-6}$. (“使用上の注意” c)参照).

iguss	int	入力	配列 x に指定された解ベクトルの近似値から反復計算を開始するかどうかを示す制御情報。 0 のとき: 解ベクトルの近似を指定しない。 0 以外のとき: 配列 x に指定された解ベクトルの近似値から反復計算を開始する。
x	double $x[n]$	入力	連立 1 次方程式の解ベクトルの近似値を指定することができる。(引数“iguss”の項を参照)
iter	int	出力	連立 1 次方程式の解ベクトルが格納される。
rz	double	出力	実際行った反復の回数。
vw	double $vw[Vwlen]$	作業領域	収束判定を行った残差 rz の平方根の値。(“使用上の注意” c)参照)。
ivw	int $ivw[Ivwlen]$	作業領域	1) $ipc = 3$ のとき $Vwlen = k * (nw + 6) + 2 * nband$. $nband$ は下バンド幅又は上バンド幅の大きさ。 2) その他のとき $Vwlen = k * 5 + 2 * nband$. $nband$ は下バンド幅又は上バンド幅の大きさ。
icon	int	出力	$Ivwlen = (k + 1) * 4$ コンディションコード. 下の表を参照。

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20001	反復回数の上限に達した.	処理を打ち切る.
20003	ブレークダウンを起こした.	配列 x には, そのときまでに得られている近似値を出力するが精度は保証できない.
30003	$itmax \leq 0$	処理を打ち切る.
30005	$k < n$	
30006	不完全 LL^T 分解ができなかった.	
30007	ピボットが負になった.	
30089	nw が偶数でない.	
30091	$nband = 0$	
30092	$nw \leq 0, n \leq 0$	
30093	$k \leq 0$	
30096	$\omega < 0, \omega > 1$	
30097	$ipc < 1, ipc > 3$	
30102	上三角部分が正しく格納されていない.	
30103	下三角部分が正しく格納されていない.	
30104	上三角の本数が, 下三角の本数と等しくない.	
30105	$isw \neq 1, 2$	
30200	$ ndlt[i] > n - 1$ 又は $ndlt[i] = 0$ ($0 \leq i < nw$)	

3. 使用上の注意

a) **a** と **ndlt** について

$n \times n$ の係数行列は、対角要素が 1 になるように正規化し、対角要素を除いた非零要素を対角形式のスパース行列の格納法で格納する。正値対称なスパース行列を係数行列とする連立 1 次方程式の正規化及び対角形式の格納方法については、“1.3.6 正値対称スパース行列の格納方法”を参照のこと。

対角形式の格納法は、係数行列 **A** の非零要素が主対角ベクトルに平行な幾つかの対角線上のベクトルにのみ存在することを仮定している。偏微分方程式を、特に定義される領域の境界と平行な格子で離散化して得られる連立 1 次方程式の場合、このような構造を持つ。この場合、各要素が係数行列の何列ベクトルにあるかを示す情報が不要で、主対角ベクトルからの距離のみを要し、効率的である。

b) **isw** について

同一係数行列を持ち定数ベクトルの異なる複数組の連立 1 次方程式を $\text{ipc} = 3$ で解く場合、1 回目は $\text{isw} = 1$ で解き、2 回目以降は $\text{isw} = 2$ で解く。2 回目以降では、1 回目の呼び出しで得られた不完全コレスキー分解の結果を再利用して解く。

c) **eps** と **rz** について

m 回目の反復で、残差 $\mathbf{r}_z$ の平方根が判定値 eps より小さいとき(2)、解が収束したと判定する。

$$\mathbf{r}_z = \sqrt{\mathbf{r}^T \mathbf{M}^{-1} \mathbf{r}} < \text{eps} \quad (2)$$

ここで、残差ベクトル $\mathbf{r}$ を(3)、前処理行列を **M** としたとき、 $\mathbf{r}_z$ は(4)で計算される。

$$\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}_m \quad (3)$$

$$\mathbf{r}_z = \mathbf{r}^T \mathbf{M}^{-1} \mathbf{r} \quad (4)$$

d) **ipc** と **omega** について

2 種類の前処理及び前処理なしの機能が提供されている。楕円型の偏微分方程式を離散化して解く場合には、不完全コレスキー分解による前処理が有効である。

$\mathbf{A} = \mathbf{I} - \mathbf{N}$ としたとき、連立 1 次方程式 $(\mathbf{I} - \mathbf{N})\mathbf{x} = \mathbf{b}$ の前処理 **M** は ipc の値により以下ようになる。

- 1 前処理なし, $\mathbf{M} = \mathbf{I}$.
- 2 Neumann, $\mathbf{M}^{-1} = (\mathbf{I} + \mathbf{N})$.
- 3 不完全コレスキー分解, $\mathbf{M} = \mathbf{L}\mathbf{L}^T$.

$\text{ipc} = 2$ のときは、前処理行列も正定値である必要がある。例えば $(\mathbf{I} + \mathbf{N})$ の対角優位性はそのための十分条件になる。また、行列 **N** の固有値の絶対値で最大の値が 1 より大きく前処理行列が **A** の逆行列の良い近似にならない場合などでは、前処理を適用しても収束が改善しない可能性がある。

$\text{ipc} = 3$ のときは、 omega ($0 \leq \text{omega} \leq 1$) に値を設定する必要がある。 omega が 0 のときは、不完全コレスキー分解、 omega が 1 のときは修正不完全コレスキー分解である。

偏微分方程式の離散化から得られる連立 1 次方程式に対しては、 $\text{omega} = 0.92$ から 1.00 が最適であることが経験から分かっている。

$\text{ipc} = 3$ のとき、方程式は前処理の効率化のためにウェーブフロント順に並び換えられる。

4. 使用例

$\mathbf{Ax} = \mathbf{b}$ を解き、得られた結果を正しい値と比較し確認する。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX      100
#define UBANDW     2

MAIN__()
{
    double one=1.0, bcoef=10.0, eps=1.e-6;
    int ierr, icon, nw, nub, n, i, j, k;
    int ipc, itmax, isw, iguss, iter;
    int ndlt[2 * UBANDW], ivw[4 * (NMAX + 1)];
    double sum, omega, rz;
    double a[2 * UBANDW][NMAX], b[NMAX], x[NMAX];
    double vw[NMAX*(2 * UBANDW + 6) + 2 * UBANDW];

    /* initialize normalized symmetric matrix and vector */
    nub = UBANDW;
    nw = nub + nub;
    n = NMAX;
    k = NMAX;
    for (i=0; i<nub; i++) {
        for (j=0 ; j<n-i; j++) a[i][j] = -1.0;
        for (j=n-i; j<n ; j++) a[i][j] = 0.0;
        ndlt[i] = i;
        for (j=0; j<i; j++) a[nub + i][j] = 0.0;
        for (j=i; j<n; j++) a[nub + i][j] = -1.0;
        ndlt[nub + i] = -i;
    }
    for (j=0; j<n; j++) {
        sum = bcoef;
        for (i=0; i<nw; i++) sum -= a[i][j];
        for (i=0; i<nw; i++) a[i][j] /= sum;
        b[j] = bcoef / sum;
    }
    /* solve the system of linear equations */
    ipc = 3;
    itmax = 8 * (int) sqrt ((double) n + 0.1);
    isw = 1;
    omega = 0.98;
    iguss = 0;
    ierr = c_dvcgd ((double*)a, k, nw, n, ndlt, b, ipc, itmax, isw,
                    omega, eps, iguss, x, &iter, &rz, vw, ivw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dvcgd failed with icon = %d\n", icon);
        exit(1);
    }
    /* check vector */
    for (i=0; i<n; i++)
        if (fabs(x[i]-one) > eps) {
            printf("WARNING: result inaccurate\n");
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書 II”の VCGD の項目及び[42], [77]を参照のこと。

c_dvcge

正値対称スパース行列の連立 1 次方程式
(前処理付き CG 法, ELLPACK 形式格納法)

```
ierr = c_dvcge(a, k, nw, n, icol, b, ipc,
               itmax, isw, omega, eps, iguss, x,
               &iter, &rz, vw, ivw, &icon);
```

1. 機能

$n \times n$ の正規化された正値対称なスパース行列を係数行列とする連立 1 次方程式(1)を前処理付き CG 法で解く.

$$\mathbf{Ax} = \mathbf{b} \quad (1)$$

係数行列は, 対角要素が 1 になるように正規化されたもので, 対角要素を除く非零要素を ELLPACK 形式の格納法で格納する. ベクトル $\mathbf{b}$ 及び $\mathbf{x}$ は n 次元ベクトル.

2. 引数

呼出し形式:

```
ierr = c_dvcge((double*)a, k, nw, n, (int*)icol, b, ipc, itmax, isw, omega,
               eps, iguss, x, &iter, &rz, vw, ivw, &icon);
```

引数の説明:

a	double a[nw][k]	入力	正規化された正値対称スパース行列の係数行列の非零要素を ELLPACK 形式で格納する. (“使用上の注意” a)参照). ipc=3 のとき, 演算後, 内容は保存されない.
k	int	入力	配列 a, icol の整合寸法($\geq n$).
nw	int	入力	配列 a の 1 次元目の大きさ. 上三角行列の行ベクトルの非零要素の最大値を <i>NSU</i> とし, 下三角行列の行ベクトルの非零要素の最大値を <i>NSL</i> とするとき, nw = 2·max(<i>NSU</i> , <i>NSL</i>). (“使用上の注意” a)参照).
n	int	入力	行列 $\mathbf{A}$ の次数 n .
icol	int icol[nw][k]	入力	非零要素がどの列ベクトルに属するかを格納する. (“使用上の注意” a)参照).
b	double b[n]	入力	連立 1 次方程式の右辺の定数ベクトルを格納する.
ipc	int	入力	前処理の制御情報. (“使用上の注意” d)参照). 1 前処理なし. 2 Neumann の前処理. 3 不完全コレスキー分解による前処理. このとき omega を指定する必要がある.
itmax	int	入力	反復回数の上限. 正の整数.
isw	int	入力	制御情報. (“使用上の注意” b)参照). 1 初回の呼出し. 2 2 回目以降の呼出し. ただし a, icol, vw, ivw の値は 1 回目に設定された値を使用するため, 変更してはならない.

omega	double	入力	不完全コレスキー分解のための修正値. $0 \leq \text{omega} \leq 1$. ipc = 3 のとき使用される. (“使用上の注意” d)参照).
eps	double	入力	収束判定に用いられる判定値. $\text{rz} < \text{eps}$ を満たしたとき収束したと見なす. eps が 0.0 以下のとき, $\text{eps} = \varepsilon \cdot \ \mathbf{b}\ $ として設定される. $\varepsilon = 10^{-6}$. (“使用上の注意” c)参照).
iguss	int	入力	配列 x に指定された解ベクトルの近似値から反復計算を開始するかどうかを示す制御情報. 0 のとき 解ベクトルの近似値は指定しない. 0 以外のとき 配列 x に指定された解ベクトルの近似値から反復を開始する.
x	double x[n]	入力	連立 1 次方程式の解の近似ベクトルの近似値を指定することができる.(引数“iguss”の項を参照)
		出力	連立 1 次方程式の解ベクトルが格納される.
iter	int	出力	実際行った反復の回数.
rz	double	出力	収束判定を行った残差 rz の平方根の値. (“使用上の注意” c)参照).
vw	double vw[Vwlen]	作業領域	1) ipc = 3 のとき $Vwlen = k * nw + 4 * n$. 2) その他のとき $Vwlen = n * 3$.
ivw	int ivw[Ivwlen]	作業領域	1) ipc = 3 のとき $Ivwlen = k * nw + 4 * n$. 2) その他のとき $Ivwlen = n * 4$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	a, icol の要素を U/L と並べ換えた.	処理は続行する.
20001	反復回数の上限に達した.	処理を打ち切る.
20003	ブレークダウンを起こした.	配列 x には, そのときまでに得られている近似値を出力するが精度は保証できない.
30003	itmax $\leq$ 0	処理を打ち切る.
30005	k < n	
30006	不完全 $\mathbf{LL}^T$ 分解が出来なかった.	
30007	ピボットが負になった.	
30092	nw $\leq$ 0	
30093	k $\leq$ 0, n $\leq$ 0	
30096	omega < 0, omega > 1	
30097	ipc < 1, ipc > 3	
30098	isw $\neq$ 1, 2	
30100	nw $\neq$ 2 * max(NSU, NSL)	
30104	上三角又は下三角部分が正しく格納されていない.	
負の数	第(-icon)行に非零対角要素がある.	処理を打ち切る.

3. 使用上の注意

a) **a**, **nw** と **icol** について

$n \times n$ の係数行列 **A** は、対角要素が 1 になるように正規化し、対角要素を除いた非零要素を ELLPACK 形式のスパース行列の格納法で格納する。正値対称なスパース行列を係数行列とする連立 1 次方程式の正規化及び ELLPACK 形式の格納方法については、“1.3.6 正値対称スパース行列の格納方法”を参照のこと。

$ipc = 3$ 以外(不完全コレスキー分解による前処理以外の前処理を指定するとき)では、正規化された正値対称スパース行列で対角要素を除いたものを、一般スパース行列の ELLPACK 形式の格納方法で格納したものも入力として受け入れる。このとき、 $nw = 2 \cdot \max(NSU, NSL)$ である必要はない。

b) **isw** について

同一係数行列を持ち定数ベクトルの異なる複数組の連立 1 次方程式を $ipc = 3$ で解く場合、1 回目は $isw = 1$ で解き、2 回目以降は $isw = 2$ で解く。2 回目以降では、1 回目の呼び出しで得られた不完全コレスキー分解の結果を再利用する。

c) **eps** と **rz** について

m 回目の反復で、残差 rz の平方根が判定値 eps より小さいとき(2), 解が収束したと判定する。

$$rz = \sqrt{rz} < eps \quad (2)$$

ここで、残差ベクトル **r** を(3)、前処理行列を **M** としたとき、 rz は(4)で計算される。

$$\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}_m \quad (3)$$

$$rz = \mathbf{r}^T \mathbf{M}^{-1} \mathbf{r} \quad (4)$$

d) **ipc** と **omega** について

2 種類の前処理及び前処理なしの機能が提供されている。楕円型の偏微分方程式を離散化して解く場合には、不完全コレスキー分解による前処理が有効である。

$\mathbf{A} = \mathbf{I} - \mathbf{N}$ としたとき、連立 1 次方程式 $(\mathbf{I} - \mathbf{N})\mathbf{x} = \mathbf{b}$ の前処理 **M** は ipc の値により以下ようになる。

- 1 前処理なし, $\mathbf{M} = \mathbf{I}$.
- 2 Neumann, $\mathbf{M}^{-1} = (\mathbf{I} + \mathbf{N})$.
- 3 不完全コレスキー分解, $\mathbf{M} = \mathbf{L}\mathbf{L}^T$.

$ipc = 2$ のときは、前処理行列も正定値である必要がある。例えば $(\mathbf{I} + \mathbf{N})$ の対角優位性はそのための十分条件になる。また、行列 **N** の固有値の絶対値で最大の値が 1 より大きく前処理行列が **A** の逆行列の良い近似にならない場合などでは、前処理を適用しても収束が改善しない可能性がある。

$ipc = 3$ のときは、 $omega$ ($0 \leq omega \leq 1$) に値を設定する必要がある。 $omega$ が 0 のときは、不完全コレスキー分解、 $omega$ が 1 のときは修正不完全コレスキー分解である。

偏微分方程式の離散化から得られる連立 1 次方程式に対しては、 $omega = 0.92$ から 1.00 が最適であることが経験から分かっている。

$ipc = 3$ のとき、方程式は前処理の効率化のためにウェーブフロント順に並び換えられる。

4. 使用例

$\mathbf{Ax} = \mathbf{b}$ を解き、得られた結果を正しい値と比較し確認する。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX      100
#define UBANDW     1

MAIN__()
{
    double cf=-1.0, bcoef=10.0, one=1.0, eps=1.e-6;
    int ierr, icon;
    int nw, n, k, id, ipc, itmax, isw, iter, iguss, i, j;
    int icol[2 * UBANDW][NMAX], ivw[NMAX * (2 * UBANDW + 5)];
    double sum, omega, rz;
    double a[2 * UBANDW][NMAX], b[NMAX], x[NMAX];
    double vw[NMAX * (2 * UBANDW + 5)];

    /* initialize matrix and vector */
    nw = 2 * UBANDW;
    n = NMAX;
    k = NMAX;
    id = 1;
    for (i=0; i<nw; i++)
        for (j=0; j<n; j++) {
            a[i][j] = 0.0;
            icol[i][j] = j+1;
        }
    for (j=0; j<n-id; j++) {
        a[0][j] = cf;
        icol[0][j] = j+id+1;
    }
    for (j=id; j<n; j++) {
        a[1][j] = cf;
        icol[1][j] = j-id+1;
    }
    for (j=0; j<n; j++) {
        sum = bcoef;
        for (i=0; i<nw; i++) sum -= a[i][j];
        for (i=0; i<nw; i++) a[i][j] /= sum;
        b[j] = bcoef / sum;
    }
    /* solve the system of linear equations */
    ipc = 3;
    itmax = 8 * (int) sqrt ((double) n + 0.1);
    isw = 1;
    omega = 0.98;
    iguss = 0;
    ierr = c_dvcge ((double*)a, k, nw, n, (int*)icol, b, ipc, itmax,
                    isw, omega, eps, iguss, x, &iter, &rz, vw, ivw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dvcge failed with icon = %d\n", icon);
        exit(1);
    }
    /* check vector */
    for (i=0; i<n; i++)
        if (fabs(x[i]-one) > eps) {
            printf("WARNING: result inaccurate\n");
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書 II”の VCGE の項目及び[42], [77]を参照のこと。

c_dvcos1

離散型 cosine 変換 (2 基底 FFT)
ierr = c_dvcos1(a, n, tab, vw, ivw, &icon);

1. 機能

周期 2π の偶関数 $x(t)$ の半周期を n 等分した $n+1$ 個の標本 $\{x_j\}$

$$x_j = x(j \cdot \theta), \quad j = 0, 1, \dots, n, \quad \theta = \frac{\pi}{n}$$

が与えられたとき、離散型 cosine 変換、又はその逆変換をベクトル計算機向きの高速変換手法(FFT)により計算する。ただし、 $n = 2^\ell$ ($\ell: 0$ 又は、正整数)であること。

cosine 変換

$\{x_j\}$ を入力し、次の式で定義する変換を行い、フーリエ係数 $\{2na_k\}$ を求める。

$$2na_k = 2x_0 + 2x_n \cdot \cos(k\pi) + 4 \cdot \sum_{j=1}^{n-1} x_j \cdot \cos(kj\theta), \quad k = 0, 1, \dots, n$$

ここで $\theta = \pi/n$ 。

cosine 逆変換

$\{a_k\}$ を入力し、次の式で定義する変換を行い、フーリエ級数の値 $\{4x_j\}$ を求める。

$$4x_j = 2a_0 + 2a_n \cdot \cos(j\pi) + 4 \cdot \sum_{k=1}^{n-1} a_k \cdot \cos(kj\theta), \quad j = 0, 1, \dots, n$$

ここで $\theta = \pi/n$ 。

2. 引数

呼出し形式:

ierr = c_dvcos1(a, n, tab, vw, ivw, &icon);

引数の説明:

a	double a[n+2]	入力	$\{x_j\}$ 又は $\{a_k\}$. $x_0, x_1, \dots, x_n$ を a[0], a[1], ..., a[n] に、又は $a_0, a_1, \dots, a_n$ を a[0], a[1], ..., a[n] に格納する。a[n+1] は任意。
		出力	$\{2na_k\}$ 又は $\{4x_j\}$. a[n+1] には 0.0 が設定される。
n	int	入力	標本の数-1.
tab	double tab[Tlen]	出力	変換で使用された三角関数表が格納される。Tlen = 2n+4.
vw	double vw[Rlen]	作業領域	Rlen = max(n(ℓ+1)/2, 1).
ivw	int ivw[Ilen]	作業領域	Ilen = n · max(ℓ-4, 2)/2.
icon	int	出力	コンディションコード。下の表を参照。

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	$n \neq 2^\ell$ ($\ell: 0$ 又は、正整数)	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

本関数は、離散型 cosine 変換をベクトル計算機上で高速処理する。

b) 複数组の変換について

複数组の変換を行う場合、2 回目以降の呼出しのときは、変換に必要な三角関数表やリストベクトルなどの生成を省略するので、効率良く処理される。このとき、配列 `tab`, `vw`, `ivw` の内容は変更せず呼び出すこと。

複数组の変換の項数 n が各々異なる場合でも、それまでに生成した、配列 `tab`, `vw`, `ivw` の内容は有効である。しかし、可能な限り同一の項数の変換が連続するように呼び出すことが望ましい。

c) 三角関数表及び作業領域の大きさ早見表

$16 \leq n \leq 4096$ の場合の大きさを以下に示す。

ℓ	n	tab	vw	ivw
4	16	36	40	16
5	32	68	96	32
6	64	132	224	64
7	128	260	512	192
8	256	516	1152	512
9	512	1028	2560	1280
10	1024	2052	5632	3072
11	2048	4100	12288	7168
12	4096	8196	26624	16384

d) 一般的な離散型 cosine 変換

離散型 cosine 変換及び、逆変換は一般的に (1), (2) で定義される。

$$a_k = \frac{x_0}{n} + \frac{x_n}{n} \cdot \cos(k\pi) + \frac{2}{n} \cdot \sum_{j=1}^{n-1} x_j \cdot \cos(kj\theta), \quad k = 0, 1, \dots, n, \quad (1)$$

$$x_j = \frac{a_0}{2} + \frac{a_n}{2} \cdot \cos(j\pi) + \sum_{k=1}^{n-1} a_k \cdot \cos(kj\theta), \quad j = 0, 1, \dots, n, \quad (2)$$

ここで $\theta = \pi/n$ 。

本関数では(1), (2)の左辺に対応して $\{2na_k\}$ 又は $\{4x_j\}$ を求める。したがって、結果の正規化は必要に応じて行うこと。

4. 使用例

cosine 変換を求める。入力は一乱数により与える。変換後、さらに逆変換して結果が正しいか確認する。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 1024

MAIN__()
{
    int ierr, icon;
```

```
double phai, ran, scale, eps;
double a[NMAX+2], b[NMAX+2], tab[2*NMAX+4], vw[NMAX*(10+1)/2];
int i, n, ivw[NMAX*(10-4)/2];

/* generate initial data */
n = NMAX;
phai = (sqrt(5.0)-1.0)/2;
for (i=0;i<n+1;i++) {
    ran = (i+1)*phai;
    a[i] = ran - (int)ran;
}
for (i=0;i<n+1;i++)
    b[i] = a[i];
/* perform normal transform */
ierr = c_dvcos1(a, n, tab, vw, ivw, &ierr);
/* perform inverse transform */
ierr = c_dvcos1(a, n, tab, vw, ivw, &ierr);
/* check results */
scale = 1.0/(8*n);
eps = 1e-6;
for (i=0;i<n+1;i++)
    if (fabs((scale*a[i]-b[i])/b[i]) > eps) {
        printf("Inaccurate result\n");
        exit(1);
    }
printf("Result OK\n");
return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書”の VCOS1 の項目及び[108]を参照のこと。

c_dvcpf1

1次元素因子離散型複素フーリエ変換

<pre>ierr = c_dvcpf1(x, n, &isw, isn, &iout, y, w, iw, &icon);</pre>

1. 機能

1次元複素フーリエ変換の正変換又は逆変換を行う。変換データ長 n は、次の条件を満たす数である。

- 次の数の中で互いに素な因子の積で表せること。
因子 p ($p \in \{2, 3, 4, 5, 7, 8, 9, 16, 25\}$)

1次元フーリエ変換

$\{x_j\}$ を入力し、(1)で定義する変換を行い、 $\{n\alpha_k\}$ を求める。

$$n\alpha_k = \sum_{j=0}^{n-1} x_j \omega_n^{-jk}, \quad k = 0, 1, \dots, n-1, \quad (1)$$

$$\omega_n = \exp(2\pi i / n)$$

1次元フーリエ逆変換

$\{\alpha_k\}$ を入力し、(2)で定義する変換を行い、 $\{x_j\}$ を求める。

$$x_j = \sum_{k=0}^{n-1} \alpha_k \omega_n^{jk}, \quad j = 0, 1, \dots, n-1, \quad (2)$$

$$\omega_n = \exp(2\pi i / n)$$

2. 引数

呼出し形式:

```
ierr = c_dvcpf1(x, n, &isw, isn, &iout, y, w, iw, &icon);
```

引数の説明:

x	dcomplex x[n]	入力	複素数データ.
n	int	入力	変換データ長 n .
isw	int	入力	制御情報. isw = 1 初回呼出し. w , iw に三角関数テーブル及び制御情報を格納しフーリエ変換を行う. isw ≠ 1 同じ大きさのデータの変換を初回以降引き続き行う. このとき, 初回に w , iw に格納された三角関数テーブル及び制御情報を利用するため, n , isn , w , iw の内容を初回呼び出し後に変更してはならない.
		出力	isw = 1 を設定して呼び出したとき, isw = 0 が設定される. このため2回目以降特に isw に値を設定せずに, 配列 x に変換データを格納して呼び出すことで効率よく変換を行うことができる.
isn	int	入力	変換か逆変換かを指定する(≠ 0) 変換 isn = 1 逆変換 isn = -1
iout	int	出力	変換するデータの大きさ n によって結果が格納される領域が異なるため, どこに格納されるかを示す情報.

y	dcomplex y[n]	出力	iout = 1 変換した結果は $y[i], i = 0, \dots, n-1$ に格納される. iout $\neq 1$ 変換した結果は $x[i], i = 0, \dots, n-1$ に格納される. iout = 1 のとき, 変換された複素データが格納される. 配列 x とは別の領域でなければなりません.
w	dcomplex w[n]	作業領域	isw = 1 のとき, データ長 n の isn で指定される変換の三角関数テーブルが格納される. isw $\neq 1$ のとき, isw = 1 を指定した初回呼び出しで作成された三角関数テーブルを入力として利用する.
iw	int iw[20]	作業領域	変換のための制御情報. isw = 1 のとき, データ長 n の isn で指定された変換の制御情報が格納される. isw $\neq 1$ のとき, isw = 1 を指定した初回呼び出しで作成された制御情報を入力として利用する.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	n が 2, 3, 4, 5, 7, 8, 9, 16, 25 の中に互いに素な因子の積に分解できなかった.	処理を打ち切る.
20100	2 回目以降の呼び出しで n, isn の値が初回呼び出しのときの値と異なる.	

3. 使用上の注意

a) 一般的なフーリエ変換の定義について

1次元離散型複素フーリエ変換及び, 逆変換は一般的に(3), (4)で定義される.

$$\alpha_k = \frac{1}{n} \sum_{j=0}^{n-1} x_j \omega_n^{-jk}, \quad k = 0, 1, \dots, n-1 \quad (3)$$

$$\omega_n = \exp(2\pi i / n)$$

$$x_j = \sum_{k=0}^{n-1} \alpha_k \omega_n^{jk}, \quad j = 0, 1, \dots, n-1 \quad (4)$$

$$\omega_n = \exp(2\pi i / n)$$

本関数では(3), (4)の左辺に対応して $\{n\alpha_k\}$ 又は $\{x_j\}$ を求める. したがって, 結果の正規化は必要に応じて行うこと.

4. 使用例

1次元FFTを計算する.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define N 560
#define max(a,b) ((a) > (b) ? (a) : (b))

int MAIN__(void)
{
    dcomplex w[N], x[N], y[N], tmp;
    double error;
    int iw[20], isw, isn, iout, icon, i;

    for (i=0; i<N; i++) {
```

```
    x[i].re=(double)(i+1)/(double)N;
    x[i].im=0.0;
}

/* do the forward transform */
isw=1, isn=1;
c_dvcpf1(x, N, &isw, isn, &iout, y, w, iw, &icon);

if (icon != 0) {
    printf("icon = %d", icon);
    exit(1);
}

/* do the reverse transform */
if (iout != 1) {
    isw=1, isn=-1;
    c_dvcpf1(x, N, &isw, isn, &iout, y, w, iw, &icon);
} else {
    isw=1, isn=-1;
    c_dvcpf1(y, N, &isw, isn, &iout, x, w, iw, &icon);
}

if (icon != 0) {
    printf("icon = %d", icon);
    exit(1);
}

error = 0.0;
for (i=0; i<N; i++) {
    tmp.re = fabs(x[i].re/(double)N - (double)(i+1)/(double)N);
    tmp.im = fabs(x[i].im/(double)N);
    tmp.re += tmp.im;
    error=max(error, tmp.re);
}

printf("error = %e\n", error);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書 II”の VCPF1 の項目を参照のこと。

c_dvcpf3

3次元素因子離散型複素フーリエ変換

ierr = c_dvcpf3(a, b, l, m, n, isn, vw1, vw2, &icon);

1. 機能

3次元(各元の項数 n_1, n_2, n_3)の複素時系列データ $\{x_{j_1 j_2 j_3}\}$ が与えられたとき, 離散型複素フーリエ変換, 又はその逆変換を素因子フーリエ変換 (a prime factor FFT) により行う. 各元の項数は, 次の条件を満たす数である.

- 次の数の中で互いに素な因子の積で表せること.
因子 p ($p \in \{2, 3, 4, 5, 7, 8, 9, 16\}$)

3次元複素フーリエ変換

$\{x_{j_1 j_2 j_3}\}$ を入力し, (1)で定義する変換を行い, $\{n_1 n_2 n_3 \alpha_{k_1 k_2 k_3}\}$ を求める.

$$n_1 n_2 n_3 \alpha_{k_1 k_2 k_3} = \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} \sum_{j_3=0}^{n_3-1} x_{j_1 j_2 j_3} \omega_1^{-j_1 k_1} \omega_2^{-j_2 k_2} \omega_3^{-j_3 k_3}, \quad (1)$$

$$k_r = 0, \dots, n_r - 1,$$

$$\omega_r = \exp(2\pi i / n_r), r = 1, 2, 3$$

3次元複素フーリエ逆変換

$\{\alpha_{k_1 k_2 k_3}\}$ を入力し, (2)で定義する変換を行い, $\{x_{j_1 j_2 j_3}\}$ を求める.

$$x_{j_1 j_2 j_3} = \sum_{k_1=0}^{n_1-1} \sum_{k_2=0}^{n_2-1} \sum_{k_3=0}^{n_3-1} \alpha_{k_1 k_2 k_3} \omega_1^{j_1 k_1} \omega_2^{j_2 k_2} \omega_3^{j_3 k_3} \quad (2)$$

$$j_r = 0, \dots, n_r - 1,$$

$$\omega_r = \exp(2\pi i / n_r), r = 1, 2, 3$$

2. 引数

呼出し形式:

```
ierr = c_dvcpf3((double *) a, (double *) b, l, m, n, isn, vw1, vw2, &icon);
```

引数の説明:

a	double a[n][m][l]	入力	$\{x_{j_1 j_2 j_3}\}$ 又は, フーリエ変換された $\{\alpha_{k_1 k_2 k_3}\}$ の実部. ("使用上の注意" a)参照)
		出力	フーリエ変換した $\{n_1 n_2 n_3 \alpha_{k_1 k_2 k_3}\}$ 又は, 逆変換された $\{x_{j_1 j_2 j_3}\}$ の実部. ("使用上の注意" a)参照)
b	double b[n][m][l]	入力	$\{x_{j_1 j_2 j_3}\}$ 又は, フーリエ変換された $\{\alpha_{k_1 k_2 k_3}\}$ の虚部. ("使用上の注意" a)参照)
		出力	フーリエ変換した $\{n_1 n_2 n_3 \alpha_{k_1 k_2 k_3}\}$ 又は, 逆変換された $\{x_{j_1 j_2 j_3}\}$ の虚部. ("使用上の注意" a)参照)
l	int	入力	1次元のデータ数 n_1 . 配列 a, b の3次元のサイズとなる. $l \leq 5040$.
m	int	入力	2次元のデータ数 n_2 . 配列 a, b の2次元のサイズとなる. $m \leq 5040$.
n	int	入力	3次元のデータ数 n_3 . 配列 a, b の1次元のサイズとなる. $n \leq 5040$.

isn	int	入力	変換か逆変換かを指定する. isn ≥ 0 (非負の整数) なら変換. isn < 0 (負の整数) なら逆変換.
vw1	double vw1[l*m*n]	作業領域	
vw2	double vw2[l*m*n]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	次のいずれかがあった. - l, m, n のいずれかが 5040 を超えた. - l, m, n のいずれかが {2,3,4,5,7,8,9,16} の中の互いに素な因子の積に分解できなかった.	処理を打ち切る.
30000	l, m, n のいずれかが 0 又は負であった.	処理を打ち切る.

3. 使用上の注意

a) データの格納方法

$\{x_{j_1 j_2 j_3}\}$, $\{n_1 n_2 n_3 \alpha_{k_1 k_2 k_3}\}$, $\{\alpha_{k_1 k_2 k_3}\}$ の実部は配列 a に以下のように格納される.

$$a[j_3][j_2][j_1] = \text{Re}(x_{j_1 j_2 j_3}), \quad j_i = 0, 1, \dots, n_i - 1, \quad i = 1, 2, 3$$

$$\text{又は} \quad a[k_3][k_2][k_1] = \text{Re}(n_1 n_2 n_3 \alpha_{k_1 k_2 k_3}) \text{ 又は } \text{Re}(\alpha_{k_1 k_2 k_3}), \quad k_i = 0, 1, \dots, n_i - 1, \quad i = 1, 2, 3$$

$\{x_{j_1 j_2 j_3}\}$, $\{n_1 n_2 n_3 \alpha_{k_1 k_2 k_3}\}$, $\{\alpha_{k_1 k_2 k_3}\}$ の虚部は配列 b に以下のように格納される.

$$b[j_3][j_2][j_1] = \text{Im}(x_{j_1 j_2 j_3}), \quad j_i = 0, 1, \dots, n_i - 1, \quad i = 1, 2, 3$$

$$\text{又は} \quad b[k_3][k_2][k_1] = \text{Im}(n_1 n_2 n_3 \alpha_{k_1 k_2 k_3}) \text{ 又は } \text{Im}(\alpha_{k_1 k_2 k_3}), \quad k_i = 0, 1, \dots, n_i - 1, \quad i = 1, 2, 3$$

b) 項数及び次元数について

項数は次の数の中で互いに素な因子 p ($p \in \{2, 3, 4, 5, 7, 8, 9, 16\}$) の積で表せる数である. 最大値は $5 \times 7 \times 9 \times 16 = 5040$ である.

引数 n に 1 を設定して呼び出すと 2 次元の複素素因子高速フーリエ変換になる. さらに, 引数 n に 1 を, そして引数 m に 1 を設定して呼び出すと 1 次元の複素素因子高速フーリエ変換になる.

c) 一般的な 3 次元フーリエ変換の定義について

3 次元フーリエ変換及び, 逆変換は一般的に(3), (4)で定義される.

$$\alpha_{k_1 k_2 k_3} = \frac{1}{n_1 n_2 n_3} \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} \sum_{j_3=0}^{n_3-1} x_{j_1 j_2 j_3} \omega_1^{-j_1 k_1} \omega_2^{-j_2 k_2} \omega_3^{-j_3 k_3} \quad (3)$$

$$k_r = 0, \dots, n_r - 1,$$

$$\omega_r = \exp(2\pi i / n_r), \quad r = 1, 2, 3$$

$$x_{j_1 j_2 j_3} = \sum_{k_1=0}^{n_1-1} \sum_{k_2=0}^{n_2-1} \sum_{k_3=0}^{n_3-1} \alpha_{k_1 k_2 k_3} \omega_1^{j_1 k_1} \omega_2^{j_2 k_2} \omega_3^{j_3 k_3} \quad (4)$$

$$j_r = 0, \dots, n_r - 1,$$

$$\omega_r = \exp(2\pi i / n_r), r = 1, 2, 3$$

本関数では(3), (4)の左辺に対応して $\{n_1 n_2 n_3 \alpha_{k_1 k_2 k_3}\}$ 又は $\{x_{j_1 j_2 j_3}\}$ を求めるので結果の正規化は必要に応じで行うこと. 本関数により変換・逆変換を正規化せずに連続して実行すると, 入力データの各要素は $n_1 n_2 n_3$ 倍されて出力される.

4. 使用例

複素時系列データをフーリエ変換し, その結果をフーリエ逆変換して, もとのデータと比較する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define N1 4
#define N2 3
#define N3 2

MAIN__()
{
    int ierr, icon;
    double phai, ran, eps;
    double a[N3][N2][N1], b[N3][N2][N1], vw1[N3][N2][N1], vw2[N3][N2][N1];
    double aa[N3][N2][N1], bb[N3][N2][N1];
    int i, j, k, cnt, l, m, n, isn, pr;

    /* generate initial data */
    l = N1;
    m = N2;
    n = N3;
    pr = l*m*n;
    phai = (sqrt(5.0)-1.0)/2;
    cnt = 1;
    for (k=0; k<n; k++) {
        for (j=0; j<m; j++) {
            for (i=0; i<l; i++) {
                ran = cnt*phai;
                a[k][j][i] = ran - (int)ran;
                b[k][j][i] = a[k][j][i] - 0.5;
                cnt++;
            }
        }
    }
    /* keep copy */
    for (k=0; k<n; k++) {
        for (j=0; j<m; j++) {
            for (i=0; i<l; i++) {
                aa[k][j][i] = a[k][j][i];
                bb[k][j][i] = b[k][j][i];
            }
        }
    }
    /* perform normal transform */
    isn = 1;
    ierr = c_dvcpf3((double*)a, (double*)b, l, m, n, isn,
                    (double*)vw1, (double*)vw2, &icon);
    /* perform inverse transform */
    isn = -1;
    ierr = c_dvcpf3((double*)a, (double*)b, l, m, n, isn,
                    (double*)vw1, (double*)vw2, &icon);
    /* check results */
    eps = 1e-6;
    for (k=0; k<n; k++) {
        for (j=0; j<m; j++) {
            for (i=0; i<l; i++) {
                if ((fabs((a[k][j][i]/pr - aa[k][j][i])/aa[k][j][i]) > eps) ||
                    (fabs((b[k][j][i]/pr - bb[k][j][i])/bb[k][j][i]) > eps)) {
```

```
        printf("WARNING: result inaccurate\n");
        exit(1);
    }
}
}
printf("Result OK\n");
return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書 II”の VCPF3 の項目及び[17], [120]を参照のこと.

c_dvcrd

非対称又は不定値のスパース行列の連立 1 次方程式 (MGCR 法, 対角形式格納法)
<pre>ierr = c_dvcrd(a, k, ndiag, n, nofst, b, itmax, eps, iguss, ndirv, x, &iter, vw, &icon);</pre>

1. 機能

$n \times n$ の非対称/不定値なスパース行列を係数行列とする連立 1 次方程式(1)を修正一般化共役残差法 (MGCR:Modified Generalized Conjugate Residuals method)で解く。

$$\mathbf{Ax} = \mathbf{b} \tag{1}$$

係数行列は, 対角形式の格納法で格納する. ベクトル $\mathbf{b}$ 及び $\mathbf{x}$ は n 次元ベクトル.

2. 引数

呼出し形式:

```
ierr = c_dvcrd((double*)a, k, ndiag, n, nofst, b, itmax, eps, iguss, ndirv,
               x, &iter, vw, &icon);
```

引数の説明:

a	double	入力	係数行列の非零要素を対角形式格納法で格納する. (“使用上の注意” a)参照).
	a[ndiag][k]		
k	int	入力	配列 a の整合寸法 ($\geq n$).
ndiag	int	入力	係数行列 A の非零要素を含む対角ベクトル列の総数. 配列 a の 1 次元目の大きさ.
n	int	入力	行列 A の次数 n .
nofst	int	入力	a に格納される対角ベクトルに対応した主対角ベクトルからの距離を格納する. 上対角ベクトル列は正, 下対角ベクトル列は負の値で表す. (“使用上の注意” a)参照).
	nofst[ndiag]		
b	double b[n]	入力	連立 1 次方程式の右辺の定数ベクトルを格納する.
itmax	int	入力	MGCR 法の反復回数の上限值. (> 0).
eps	double	入力	収束判定に用いられる判定値. eps が 0.0 以下のとき, eps は 10^{-6} が設定される. (“使用上の注意” b)参照).
iguss	int	入力	配列 x に指定された解ベクトルの近似値から反復計算を開始するかどうかを示す制御情報. 0 のとき 解ベクトルの近似値を指定しない. 0 以外のとき 配列 x に指定された解ベクトルの近似値から反復計算を開始する.
ndirv	int	入力	MGCR 法で使われる検索方向ベクトルの数 (≥ 1). 一般に 10 から 100 の小さな数.
x	double x[n]	入力	連立 1 次方程式の解ベクトルの近似値を指定することができる. (引数“iguss”の項を参照)
		出力	解ベクトル $\mathbf{x}$ が格納される.
iter	int	出力	MGCR 法での実際の反復回数.

vw double 作業領域 $Vwlen = n * (ndirv + 5) + ndirv * (ndirv + 1)$
 vw[Vwlen]
 icon int 出力 コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20001	反復回数の上限に達した.	処理を打ち切る. 配列 x には, そのときまでに得られている近似値を出力するが, 精度は保証できない.
30000	次のいずれかであった. • $n < 1$ • $k < 1$ • $n > k$ • $ndiag < 1$ • $itmax \leq 0$	処理を打ち切る.
30004	$ndirv < 1$	
32001	$ nofst[i] > n - 1$ ($0 \leq i < ndiag$)	

3. 使用上の注意

a) a と nofst について

$n \times n$ の係数行列は, 2 つの配列を使用する対角形式の格納法で格納する.

係数行列の外側の対角ベクトルの要素は零を設定する必要がある. 対角ベクトル列を配列 a に格納する順序に制限は特にない. この方法の利点は, 行列ベクトル積が間接指標を使わずに計算できる点である. 対角構造を持たない行列は効率よく格納できないという点が欠点である.

対角形式の格納方法については, “1.3.5 一般スパース行列の格納方法”を参照のこと.

b) eps について

MGCR 法は, 残差のユーグリッドノルムが最初の残差のユーグリッドノルムと eps の積以下になったとき収束したと見なす. 正確な解と求められた近似解の誤差はほぼ行列 A の条件数と eps の積に等しくなる.

4. 使用例

$Ax = b$ を解き, 得られた結果を正しい値と比較し確認する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX      100
#define UBANDW     2
#define LBANDW     1
#define NSDIR      50

MAIN__()
{
    double one=1.0, bcoef=10.0, eps=1.e-6;
    int ierr, icon, ndiag, nub, nlb, n, i, j, k;
    int itmax, iguss, ndirv, iter;
    int nofst[UBANDW + LBANDW + 1];
    double a[UBANDW + LBANDW + 1][NMAX], b[NMAX], x[NMAX];
    double vw[NMAX * (NSDIR + 5) + NSDIR * (NSDIR + 1)];
```



```

/* initialize nonsymmetric matrix and vector */
nub  = UBANDW;
nlb  = LBANDW;
ndiag = nub + nlb + 1;
n    = NMAX;
k    = NMAX;
for (i=1; i<=nub; i++) {
    for (j=0 ; j<n-i; j++) a[i][j] = -1.0;
    for (j=n-i; j<n ; j++) a[i][j] = 0.0;
    nofst[i] = i;
}
for (i=1; i<=nlb; i++) {
    for (j=0 ; j<i+1; j++) a[nub + i][j] = 0.0;
    for (j=i+1; j<n ; j++) a[nub + i][j] = -2.0;
    nofst[nub + i] = -(i + 1);
}
nofst[0] = 0;
for (j=0; j<n; j++) {
    a[0][j] = bcoef;
    for (i=1; i<ndiag; i++) a[0][j] -= a[i][j];
    b[j] = bcoef;
}
/* solve the system of linear equations */
itmax = n;
iguss = 0;
ndirv = NSDIR;
ierr = c_dvcrd ((double*)a, k, ndiag, n, nofst, b, itmax, eps,
                iguss, ndirv, x, &iter, vw, &icon);
if (icon != 0) {
    printf("ERROR: c_dvcrd failed with icon = %d\n", icon);
    exit(1);
}
/* check vector */
for (i=0; i<n; i++)
    if (fabs(x[i]-one) > eps) {
        printf("WARNING: result inaccurate\n");
        exit(1);
    }
printf("Result OK\n");
return(0);
}

```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書 II”の VCRD の項目及び[66], [92]を参照のこと。

c_dvcre

非対称又は不定値のスパース行列の連立 1 次方程式(MGCR 法, ELLPACK 形式格納法)

```
ierr = c_dvcre(a, k, iwidt, n, icol, b,
               itmax, eps, iguss, ndirv, x,
               &iter, vw, &icon);
```

1. 機能

$n \times n$ の非対称/不定値なスパース行列を係数行列とする連立 1 次方程式(1)を修正一般化共役残差法 (MGCR:Modified Generalized Conjugate Residuals Method)で解く.

$$\mathbf{Ax} = \mathbf{b} \quad (1)$$

係数行列は, ELLPACK 形式の格納法で格納する. ベクトル $\mathbf{b}$ 及び $\mathbf{x}$ は n 次元ベクトル.

2. 引数

呼出し形式:

```
ierr = c_dvcre((double*)a, k, iwidt, n, (int*)icol, b, itmax, eps, iguss,
               ndirv, x, &iter, vw, &icon);
```

引数の説明:

a	double a[iwidt][k]	入力	係数行列の非零要素を ELLPACK 形式格納法で格納する. ("使用上の注意" a)参照).
k	int	入力	a 及び icol の整合寸法 ($\geq n$).
iwidt	int	入力	係数行列 A の非零要素の行ベクトル方向の最大個数. icol 及び a の 1 次元目の大きさ.
n	int	入力	行列 A の次数 n .
icol	int icol[iwidt][k]	入力	ELLPACK 形式で使用される列指標で, a の対応する要素が いずれの列ベクトルに属するかを示す. ("使用上の注意" a)参 照).
b	double b[n]	入力	連立 1 次方程式の右辺の定数ベクトルを格納する.
itmax	int	入力	MGCR 法の反復回数の上限值. (> 0).
eps	double	入力	収束判定に用いられる判定値. eps が 0.0 以下のとき, eps は 10^{-6} が設定される. ("使用上の注意" b)参照).
iguss	int	入力	配列 x に指定された解ベクトルの近似値から反復計算を開 始するかどうかを示す制御情報. 0 のとき 解ベクトルの近似値を指定しない. 0 以外のとき 配列 x に指定された解ベクトルの近似値か ら反復計算を開始する.
ndirv	int	入力	MGCR 法で使われる検索方向ベクトルの数 (≥ 1). 一般に 10 から 100 の小さな数.
x	double x[n]	入力	連立 1 次方程式の解ベクトルの近似値を指定することがで きる. (引数 iguss の項を参照)
iter	int	出力	解ベクトルが格納される.
		出力	MGCR 法での実際の反復回数.

vw double 作業領域 $Vwlen = n * (ndirv + 5) + ndirv * (ndirv + 1)$
 vw[Vwlen]
 icon int 出力 コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
20001	最大反復回数に達した.	処理を打ち切る. 配列 x には, そのときまでに得られている近似値を出力するが, 精度は保証できない.
30000	次のいずれかであった. • $n < 1$ • $k < 1$ • $n > k$ • $iwidth < 0$ • $itmax \leq 0$	処理を打ち切る.
30004	$ndirv < 1$	

3. 使用上の注意

a) a と icol について

係数行列 **A** は, 2 つの配列を使用する, ELLPACK 形式の格納法で格納する. ELLPACK 形式の格納方法については, “1.3.5 一般スパース行列の格納方法”を参照.

b) eps について

MGCR 法は, 残差のユーグリッドノルムが最初の残差のユーグリッドノルムと eps の積以下になったとき収束したと見なす. 正確な解と求められた近似解の誤差はほぼ行列 **A** の条件数と eps の積に等しくなる.

4. 使用例

Ax = b を解き, 得られた結果を正しい値と比較し確認する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX      100
#define UBANDW    2
#define LBANDW    1
#define NSDIR     50

MAIN__()
{
    double lcf=-2.0, ucf=-1.0, bcoef=10.0, one=1.0, eps=1.e-6;
    int ierr, icon, nlb, nub, iwidth, n, k, itmax, iguss, ndirv, iter, i, j, ix;
    int icol[UBANDW + LBANDW + 1][NMAX];
    double a[UBANDW + LBANDW + 1][NMAX], b[NMAX], x[NMAX];
    double vw[NMAX * (NSDIR + 5) + NSDIR * (NSDIR + 1)];

    /* initialize matrix and vector */
    nub = UBANDW;
    nlb = LBANDW;
    iwidth = UBANDW + LBANDW + 1;
    n = NMAX;
    k = NMAX;
    for (i=0; i<n; i++) b[i] = bcoef;
    for (i=0; i<iwidth; i++)
        for (j=0; j<n; j++) {
```

```
        a[i][j] = 0.0;
        icol[i][j] = j+1;
    }
    for (j=0; j<nlb; j++) {
        for (i=0; i<j; i++) a[i][j] = lcf;
        a[j][j] = bcoef - (double) j * lcf - (double) nub * ucf;
        for (i=j+1; i<j+1+nub; i++) a[i][j] = ucf;
        for (i=0; i<=nub+j; i++) icol[i][j] = i+1;
    }
    for (j=nlb; j<n-nub; j++) {
        for (i=0; i<nlb; i++) a[i][j] = lcf;
        a[nlb][j] = bcoef - (double) nlb * lcf - (double) nub * ucf;
        for (i=nlb+1; i<iwidt; i++) a[i][j] = ucf;
        for (i=0; i<iwidt; i++) icol[i][j] = i+1+j-nlb;
    }
    for (j=n-nub; j<n; j++){
        for (i=0; i<nlb; i++) a[i][j] = lcf;
        a[nlb][j] = bcoef - (double) nlb * lcf - (double) (n-j-1) * ucf;
        for (i=1; i<nub-2+n-j; i++) a[i+nlb][j] = ucf;
        ix = n - (j+nub-nlb-1);
        for (i=n; i>=j+nub-nlb-1; i--) icol[ix--][j] = i;
    }
    /* solve the system of linear equations */
    itmax = n;
    iguss = 0;
    ndirv = NSDIR;
    ierr = c_dvcre ((double*)a, k, iwidt, n, (int*)icol, b, itmax,
                    eps, iguss, ndirv, x, &iter, vw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dvcre failed with icon = %d\n", icon);
        exit(1);
    }
    /* check vector */
    for (i=0; i<n; i++)
        if (fabs(x[i]-one) > eps) {
            printf("WARNING: result inaccurate\n");
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書 II”の VCRE の項目及び[66], [92]を参照のこと。

c_dvgsg2

実対称行列の一般固有値固有ベクトル (並列バイセクション法, 逆反復法)

```
ierr = c_dvgsg2(a, b, n, m, epsz, epst, e,
                ev, k, vw, ivw, &icon);
```

1. 機能

n 次の実対称行列 $\mathbf{A}$, 及び n 次の正値対称行列 $\mathbf{B}$ に対する一般固有値問題,

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{B}\mathbf{x} \quad (1)$$

の固有値を, 並列バイセクション法により, 大きい方から (又は, 小さい方から) m 個求める. また, 対応する m 個の固有ベクトル $x_1, x_2, \dots, x_m$ を逆反復法により求める. 固有ベクトルは,

$$\mathbf{X}^T\mathbf{B}\mathbf{X} = \mathbf{I}$$

ここで $\mathbf{X} = [x_1, x_2, \dots, x_m]$ である. ただし $1 \leq m \leq n$ であること.

2. 引数

呼出し形式:

```
ierr = c_dvgsg2(a, b, n, m, epsz, epst, e, (double *)ev, k, vw, ivw, &icon);
```

引数の説明:

a	double a[Alen]	入力	実対称行列 $\mathbf{A}$. 対称行列用圧縮格納法. $Alen = n(n+1)/2$. 格納方法の詳細については“1.3.2 対称行列の格納方法”を参照のこと.
		出力	演算後, 内容は保存されない.
b	double b[Blen]	入力	正値対称行列 $\mathbf{B}$. 対称行列用圧縮格納法. $Blen = n(n+1)/2$.
		出力	演算後, 内容は保存されない.
n	int	入力	実対称行列 $\mathbf{A}$, 及び正値対称行列 $\mathbf{B}$ の次数 n .
m	int	入力	求める固有値の数 m . $m = +m$ のときは大きい方から求める. $m = -m$ のときは小さい方から求める.
epsz	double	入力	$\mathbf{B}$ の $\mathbf{LL}^T$ 分解におけるピボットの相対零判定値. 零又は負の値を指定すると標準値が採用される. (“使用上の注意” a) 参照)
epst	double	入力	固有値の収束判定に使われる絶対誤差の上限. 負の値を指定すると標準値が採用される. (“使用上の注意” b) 参照)
e	double e[m]	出力	固有値. m が正の場合大きい順に, m が負の場合小さい順に格納される.
ev	double ev[m][k]	出力	固有ベクトル. 固有値 $e[i]$ に対応する固有ベクトルは, $ev[i][j]$, $j = 0, 1, \dots, n-1$ に格納される.
k	int	入力	配列 ev の整合寸法 ($\geq n$).

vw	double	作業領域	
	vw[15*n]		
ivw	int ivw[7*n]	作業領域	
icon	int	出力	コンディションコード. 下の表参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	n=1であった.	ev[0][0] = 1/sqrt(b[0]), e[0] = a[0]/b[0]とする.
15000	固有ベクトルの計算で求まらないものがあった.	求まらない固有ベクトルを零ベクトルとする.
20000	固有ベクトルはすべて求まらなかった.	固有ベクトルはすべて零ベクトルとする.
28000	B の $\mathbf{LL}^T$ 分解においてピボットが負になった. B が正値でない.	処理を打ち切る.
29000	B の $\mathbf{LL}^T$ 分解においてピボットが相対的に零となった. B は非正則の可能性が強い.	処理を打ち切る.
30000	次のいずれかであった. • m = 0 • n < m • k < n	処理を打ち切る.

3. 使用上の注意

a) epsz

引数 epsz の標準値は丸め誤差の単位を μ としたとき, $\text{epsz}=16\mu$ である.

本関数において epsz に 10^{-s} を設定したとすると正値対称行列 **B** の $\mathbf{LL}^T$ 分解の過程でピボットの値が 10 進 s けた以上のけた落ちを生じた場合に, そのピボットを零とみなしコンディションコードを設定し(icon = 29000)処理を打ち切る.

なお, ピボットが小さくなくても計算を続行させる場合は, epsz に非常に小さな値を与えればよいが, 結果の精度は保証されない.

一方, 分解の過程でピボットの値が負となった場合は, 行列 **B** が正則でないとみなし, コンディションコードを設定し(icon = 28000)処理を打ち切る.

b) epst

引数 epst の標準値は, 丸め誤差の単位を μ としたとき,

$$\text{epst} = \mu \cdot \max(|\lambda_{\max}|, |\lambda_{\min}|) \quad (2)$$

である. ここで $\lambda_{\max}$ 及び $\lambda_{\min}$ は, $\mathbf{Ax} = \lambda\mathbf{Bx}$ の固有値の存在範囲 (ゲルシュゴリンの定理により与えられる) の上限と下限である.

固有値の絶対値が非常に大きいものと, 非常に小さいものとが混在しているとき(2)により収束判定すると, 一般に小さい方の固有値を精度よく求めることは困難である. このような場合は, epst に小さな値 (絶対誤差) を設定することにより, 小さい方の固有値も精度よく求めることができる. ただし, 反復回数が増加するため処理速度は遅くなる.

4. 使用例

5×5 行列の固有値及び固有ベクトルを求める。

```
#include <stdlib.h>
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 5

MAIN__()
{
    int ierr, icon;
    int n, m, i, j, k, ij, ivw[7*NMAX];
    double a[NMAX*(NMAX+1)/2], b[NMAX*(NMAX+1)/2];
    double e[NMAX], ev[NMAX][NMAX], vw[15*NMAX], epsz, epst;

    /* initialize matrix */
    n = NMAX;
    ij = 0;
    for (i=0;i<n;i++) {
        for (j=0;j<i;j++) {
            a[ij] = n-i;
            b[ij++] = 0;
        }
        a[ij] = n-i;
        b[ij++] = 1;
    }
    k = NMAX;
    m = n;
    epsz = 0;
    epst = 0;
    /* find eigenvalues and eigenvectors */
    ierr = c_dvgsg2(a, b, n, m, epsz, epst, e, (double*)ev, k, vw, ivw, &icon);
    if (icon >= 20000) {
        printf("ERROR: c_dvgsg2 failed with icon = %d\n", icon);
        exit(1);
    }
    /* print eigenvalues and eigenvectors */
    for (i=0;i<m;i++) {
        printf("e-value %d: %10.4f\n", i+1, e[i]);
        printf("e-vector:");
        for (j=0;j<n;j++)
            printf("%7.4f ", ev[i][j]);
        printf("\n");
    }
    return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書”の VGSG2 の項目及び[16], [118]を参照のこと。

c_dvhevp

エルミート行列の固有値及び固有ベクトル (三重対角化, マルチセクション法, 逆反復法)

<pre>ierr = c_dvhevp(ar, ai, k, n, nf, nl, ivec, &etol, &ctol, nev, e, maxne, m, evr, evi, vw, iw, &icon);</pre>

1. 機能

n 次のエルミート行列 $\mathbf{A}$ の指定されたいいくつかの固有値を求める. 指定に応じて対応する固有ベクトルを計算する.

$$\mathbf{Ax} = \lambda \mathbf{x}$$

2. 引数

呼出し形式:

```
ierr = c_dvhevp((double *)ar, (double *)ai, k, n, nf, nl, ivec, &etol, &ctol,
                nev, e, maxne, (int *)m, (double *)evr, (double *)evi, vw, iw,
                &icon);
```

引数の説明:

ar	double ar[n][k]	入力	固有値・固有ベクトルを求めるエルミート行列 $\mathbf{A}$ の実部を格納する.
ai	double ai[n][k]	入力	固有値・固有ベクトルを求めるエルミート行列 $\mathbf{A}$ の虚部を格納する.
k	int	入力	配列 ar, ai の整合寸法 ($k \geq n$).
n	int	入力	エルミート行列 $\mathbf{A}$ の次数 n .
nf	int	入力	固有値を小さいほうから番号付けして(多重固有値には多重度分番号を割り当てる), 求める最初の固有値の番号.
nl	int	入力	固有値を小さいほうから番号付けして(多重固有値には多重度分番号を割り当てる), 求める最後の固有値の番号.
ivec	int	入力	制御情報. ivec = 1 のとき固有値及び固有ベクトルの両方を求める. ivec $\neq$ 1 のとき固有値のみ求める.
etol	double	入力 出力	固有値が数値的に異なるか多重かを判定するための判定値. etol $< 3 \times 10^{-16}$ のときは, etol = 3×10^{-16} が標準値として設定される. (“使用上の注意”a)参照)
ctol	double	入力 出力	近接している固有値が近似的に多重かを判定するための判定値 ($\geq$ etol). ctol $<$ etol のときは, ctol = etol が標準値として設定される. (“使用上の注意”a)参照)
nev	int nev[3]	出力	求められた固有値の個数に関する情報. nev[0] は, 異なる固有値の個数. nev[1] は, 異なる近似的多重固有値(cluster)の個数. nev[2] は, 多重度も含んだ全ての固有値の個数.
e	double e[maxne]	出力	固有値が格納される. 求められた固有値は e[i-1], i = 1,..., nev[2] に格納される.

maxne	int	入力	計算できる固有値の最大個数. 配列 e の大きさ. (“使用上の注意”b)参照)
m	int m[2][maxne]	出力	求められた固有値の多重度に関する情報. m[0][i-1] は i 番目の固有値 λ_i の多重度を, m[1][i-1] は i 番目の固有値 λ_i に関して, 近接している固有値を近似的多重固有値(cluster)と見なしたときの多重度を示す.
evr	double evr[maxne][k]	出力	ivec = 1 のとき, おおのの固有値に対応して固有ベクトルの実部が格納される. 求められた固有ベクトルは evr[i-1][j-1], $i = 1, \dots, nev[2]$, $j = 1, \dots, n$ に格納される.
evi	double evi[maxne][k]	出力	ivec = 1 のとき, おおのの固有値に対応して固有ベクトルの虚部が格納される. 求められた固有ベクトルは evi[i-1][j-1], $i = 1, \dots, nev[2]$, $j = 1, \dots, n$ に格納される.
vw	double vw[17*k]	作業領域	
iw	int iw[lwlen]	作業領域	lwlen = 9*maxne+128.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	多重固有値の計算中, 固有値の総数が maxne を超えた.	処理を打ち切る. 固有ベクトルを求めることはできないが, 異なる固有値自体は求められている. (“使用上の注意”b)参照)
30000	次のいずれかであった. • $n < 1$ • $k < n$ • $nf < 1$ • $nl > n$ • $nl < nf$ • $maxne < nl - nf + 1$	処理を打ち切る.
30100	入力行列がエルミート行列でない可能性がある.	

3. 使用上の注意

a) etol 及び ctol

本ルーチンは, 多重固有値(cluster)に対して特別の考慮を払っている.

$\varepsilon = \text{etol}$ としたとき, 連続する固有値 λ_j , $j = s, s+1, \dots, s+k$, ($k \geq 0$) に対して,

$$\frac{|\lambda_i - \lambda_{i-1}|}{1 + \max(|\lambda_{i-1}|, |\lambda_i|)} \leq \varepsilon, \quad (1)$$

$i = s, s+1, \dots, s+k$ となる i に対して(1)を満たし, $i = s-1$ 及び $i = s+k+1$ について(1)を満たさないとき, これらの固有値 λ_j , $j = s-1, s, \dots, s+k$ は数値的に多重であると見なされる.

etol の標準値は 3×10^{-16} ～丸め誤差の単位であり, このとき固有値は計算機で求め得る精度まで分離される.

(1)が $\varepsilon = \text{etol}$ に対して成立しないとき, λ_{i-1} 及び λ_i は異なる固有値と考える.

$\varepsilon = \text{etol}$ で異なる固有値と見なされた連続する固有値 λ_m , $m=t-1, t, \dots, t+k$, ($k \geq 0$) に対して, $\varepsilon = \text{ctol}$ としたとき, $i=t, t+1, \dots, t+k$ について(1)を満たし, $i=t-1$ 及び $i=t+k+1$ について(1)を満たさないとき, これらの異なる固有値 λ_m , $m=t-1, t, \dots, t+k$ を近似的に多重(cluster)と見なす. これは, cluster に対応する不変部分空間, つまり 1 次独立な固有ベクトルを計算するために利用される. つまり, 対応する固有ベクトルは直交する初期ベクトルを用いて計算され, 再直交化される. もちろん, $\text{ctol} \geq \text{etol}$ を満たさなければならない. この条件を満たさないとき, ctol は etol の値と等しく設定される.

b) maxne

r 個数の固有値を求めるとき, 最初又は最後の固有値が, 1 より大きい多重度を持つとき, r 個以上の固有値が求まる. 対応する固有ベクトルは増えた固有値も含んで, 対応する固有ベクトルの格納域が足りるときのみに計算される. maxne に, 計算できる固有値の最大個数を指定できる. 計算する固有値の総数が maxne を超えた場合, $\text{icon}=20000$ を返却する. このとき, 固有ベクトルの計算を行うよう指定されていたら固有ベクトルの計算はできない. 固有値は求められているが, 多重度分だけ繰り返して格納はされていない. つまり, 固有値に関しては, 求められた異なる固有値が $e[i-1]$, $i=1, \dots, \text{nev}[0]$ に, および対応する固有値の多重度が $m[0][i-1]$, $i=1, \dots, \text{nev}[0]$ にそれぞれ格納されている. 固有値がすべて異なり, 近似的な多重固有値もない場合, maxne は入力として $\text{nev}[0]$ に指定する求める固有値の総数で十分である. 計算する固有値の総数が maxne を超えた場合, 計算を続けるために必要な maxne の値は $\text{nev}[2]$ に返却される.

4. 使用例

エルミート行列の固有値を計算する.

```
#include <stdio.h>
#include <stdlib.h>
#include "cssl.h" /* standard C-SSL II header file */

#define K          512
#define N          K
#define NF         1
#define NL         28
#define MAXNE      NL-NF+1
#define NVW        19*K
#define NIW        9*MAXNE+128

MAIN__()
{
    double ar[N][K], ai[N][K];
    double e[MAXNE], evr[MAXNE][K], evi[MAXNE][K];
    double vw[NVW];
    double etol, ctol;
    int    nev[3], m[2][MAXNE], iw[NIW];
    int    ierr, icon;
    int    i, j, k, n, nf, nl, maxne, ivec;

    n      = N;
    k      = K;
    nf     = NF;
    nl     = NL;
    ivec   = 1;
    maxne  = MAXNE;
    etol   = 1.0e-14;
    ctol   = 5.0e-12;

    printf(" Number of data points = %d\n", n);
    printf(" Parameter k = %d\n", k);
    printf(" Eigenvalue calculation tolerance = %12.4e\n", etol);
    printf(" Cluster tolerance = %12.4e\n", ctol);
    printf(" First eigenvalue to be found is %d\n", nf);
    printf(" Last eigenvalue to be found is %d\n", nl);

    /* Set up real and imaginary parts of matrix in AR and AI */
    for(i=0; i<n; i++) {
        for(j=0; j<n; j++) {
            ar[i][j] = (double)(i+j+2)/(double)n;
            if(i==j) {
                ai[i][j] = 0.0;
            }
        }
    }
}
```

```

        ar[i][j] = (double)(j+1);
    } else {
        ai[i][j] = (double)((i+1)*(j+1))/(double)(n*n);
    }
}
}
for(i=0; i<n; i++) {
    for(j=0; j<n; j++) {
        if(i > j) ai[i][j] = -ai[i][j];
    }
}
/* Call complex eigensolver */
ierr = c_dvhevp ((double*)ar, (double*)ai, k, n, nf, nl, ivec, &etol, &ctol, nev, e,
                maxne, (int*)m, (double*)evr, (double*)evi, vw, iw, &icon);
if (icon > 20000) {
    printf("ERROR: c_dvhevp failed with icon = %d\n", icon);
    exit(1);
}
printf("icon = %i\n", icon);
/* print eigenvalues */
printf(" Number of Hermitian eigenvalues = %d\n", nev[2]);
printf(" Eigenvaluse of complex Hermitian matrix\n");
for(i=0; i<nev[2]; i++) {
    printf("  e[%d] = %12.4e\n", i, e[i]);
}
return(0);
}

```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書 II”の VHEVP の項目及び[81], [118]を参照のこと。

c_dvland

実対称スパース行列の固有値及び固有ベクトル (Lanczos 法, 対角形式格納法)

```
ierr = c_dvland(a, k, ndiag, n, nofst, ivec,
               ix, eps, nmin, nmax, nlmin, nlmax,
               kr, maxc, e, indx, &ncmin, &ncmax,
               ev, vw, ivw, &icon);
```

1. 機能

大規模実対称スパース行列 A の最大固有値から少数個又は/及び最小固有値から少数個の固有値と対応する固有ベクトルを Lanczos 法で求める。

$$Ax = \lambda x \quad (1)$$

大規模実対称スパース行列の最大又は最小固有値を求める有効な方法である。

2. 引数

呼出し形式:

```
ierr = c_dvland((double *) a, k, ndiag, n, nofst, ivec, ix, eps, nmin, nmax,
               nlmin, nlmax, kr, maxc, e, indx, &ncmin, &ncmax, (double *) ev,
               vw, iwv, &icon);
```

引数の説明:

a	double a[ndiag][k]	入力	実対称スパース行列 A の非零要素を格納する. 実対称係数スパース行列の非零要素を含む対角ベクトルを, 対角形式格納法を利用して格納する. 格納法の詳細については“1.3.5 一般スパース行列の格納方法”を参照のこと.
k	int	入力	配列 a 及び ev の整合寸法($\geq n$).
ndiag	int	入力	係数行列 A の非零要素を含む対角ベクトル列の総数.
n	int	入力	行列 A の次数 n .
nofst	int nofst[ndiag]	入力	配列 a に格納される対角ベクトルに対応した主対角ベクトルからの距離を格納する. 上対角ベクトル列は正, 下対角ベクトル列は負の値で表す.
ivec	int	入力	ev[0][i], i = 0,...,n-1 に指定した初期ベクトルを利用するかどうかを示す制御情報. 1 のとき, ev に格納されたベクトルを初期ベクトルとして利用する. 1 以外のとき, 初期ベクトルを乱数列を生成して構成する. (“使用上の注意”a)参照)
ix	int	入力	ivec $\neq 1$ の場合に, 初期ベクトルを乱数列を生成して構成するとき, 乱数列を生成するための初期値. 1~100000 の任意の整数値を与える. (“使用上の注意”a)参照)
eps	double	入力	計算の途中で得られた固有値・固有ベクトル (λ_i, x_i) が行列 A の固有値・固有ベクトルであるかを判定する判定値. eps ≤ 0 のときは 10^{-6} が標準値として設定される. (“使用上の注意”b)参照)
nmin	int	入力	求める最小固有値及び対応する固有ベクトルの個数 (≥ 0). 小

nmax	int	入力	小さな数で, $nmax \geq 1$ なら 0 でもよい. 求める最小固有値及び対応する固有ベクトルの個数 (≥ 0). 小さな数で, $nmin \geq 1$ なら 0 でもよい.
nlmin	int	入力	最小固有値の計算で使われる数 ($\geq nmin$). 多くの場合は, $nlmin = 2 \times nmin$ で十分である. (“使用上の注意”c)参照)
nlmax	int	入力	最大固有値の計算で使われる数 ($\geq nmax$). 多くの場合は, $nlmax = 2 \times nmax$ で十分である. (“使用上の注意”c)参照)
kr	int	入力	Lanczos 法で生成されるクリーロフ部分空間の次元数の上限値 ($\geq nlmin + nlmax$). (“使用上の注意”d)参照)
maxc	int	入力	密集している (cluster) 固有値の最大個数. 例えば 10. (“使用上の注意”e)参照)
e	double e[Elen]	出力	求められた最大及び最小固有値が, 間接指標リスト indx を使って昇順に格納される. $Elen = nlmin + nlmax$. $e[indx[i-1]], i = 1, \dots, n_{cmin}$ に最小固有値が格納され, $e[indx[nmin+nmax-i]], i = 1, \dots, n_{cmax}$ に最大固有値が格納される.
indx	int indx [nmin+nmax]	出力	配列 e 及び配列 ev の間接指標が格納される. 固有値 $e[indx[i]]$ に対応する固有ベクトルは $ev[indx[i]][j], j = 0, \dots, n-1; i = 0, \dots, nmin+nmax-1$ に格納される.
ncmin	int	出力	求められた最小固有値及び対応する固有ベクトルの個数.
ncmax	int	出力	求められた最大固有値及び対応する固有ベクトルの個数.
ev	double ev[Evlen][k]	入力 出力	$ivec = 1$ のとき, 初期ベクトルを $ev[0][j], j = 0, \dots, n-1$ に格納する. $Evlen = nlmin + nlmax$. 計算された固有ベクトル. 固有値に対して, 同様に間接指標リスト indx を使って引用できる.
vw	double vw[Vwlen]	作業領域	$Vwlen = (maxc + mnl)(kr + 2) + (md + 14)(kr + 1) + 7k$. ここで $mnl = \max\{nlmin, nlmax\}, md = nlmin + nlmax$.
ivw	int ivw[Ivwlen]	作業領域	$Ivwlen = 11 \times (maxc + mnl) + md + 128$. ここで $mnl = \max\{nlmin, nlmax\}, md = nlmin + nlmax$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	密集固有値の計算中, 固有値の総数が maxc を超えた.	処理を打ち切る.
30000	次のいずれかであった: • $n < 1$ • $k < n$ • $ndiag < 1$ • $ix < 1$ 又は $ix > 100000$ • $nlmin < nmin$ 又は $nlmax < nmax$ • $nmin < 0$ 又は $nmax < 0$ • $nmin = nmax = 0$	処理を打ち切る.

コード	意味	処理内容
30004	$kr < nlmin + nlmax$	処理を打ち切る.
32001	$ nofst[i-1] > n-1, i = 1, \dots, ndiag$	処理を打ち切る.
39001	初期ベクトルが零か又は零に近い.	処理を打ち切る.
39006	入力行列が対称行列でない.	処理を打ち切る.

3. 使用上の注意

a) ivec 及び ix

Lanczos 法は確定的な手法ではない. Householder 変換による三重対角化をベースにした方法ほど堅固な方法ではない.

Lanczos 法で得た結果は初期ベクトルの選択に大きく依存する. 初期ベクトルが求める固有ベクトルの比較的大きな要素を含むとき, よい近似で望む固有値・固有ベクトルが計算される. これらの要素が小さいか又は無いとき, 望む固有値・固有ベクトルは得られないことがある. しかし, 計算された固有値・固有ベクトルは行列 A の望まれたものとは別の固有値・固有ベクトルの近似となっている.

ほとんどの場合, 計算前に初期ベクトルは知られていないので, 初期ベクトルは乱数を使って生成される. 初期値 ix を変えて異なる乱数列を初期ベクトルとして本ルーチンを複数回呼び出すことを勧める.

b) 精度について

固有値・固有ベクトルの対 $(\lambda_i, \mathbf{x}_i)$ は $\|\mathbf{A}\mathbf{x}_i - \lambda_i \mathbf{x}_i\| \leq n\varepsilon |\lambda_i|$ を満たすとき行列 A の固有値・固有ベクトルとして受け入れる. 満たさないときは, この対は捨てられる. ここで $\varepsilon = \text{eps}$, $n = \text{KR}$, KR はクリーロフ部分空間の次元数. eps の値に対する依存性は緩やかである. ただし, eps が大き過ぎると得られた固有値・固有ベクトルが十分精度の良いものでない可能性がある.

c) nlmin 及び nlmax

Lanczos 法では, 元の行列 A の固有値・固有ベクトルではない偽り (spurious) の固有値・固有ベクトルが計算されることがあるため, これらを除外している. このため求める固有値・固有ベクトルを多めに計算する必要がある. nlmin 及び nlmax はこの個数を示す. この大きさは注意して決める必要がある. ほとんどの場合, $nlmin = nmin$, $nlmax = nmax$ では不十分である. $nlmin = 2 \times nmin$, $nlmax = 2 \times nmax$ 程度の大きさで普通の場合には十分である.

d) kr

kr を大きくすると近似の良い固有値・固有ベクトルが得られるが, より多くのメモリと計算量が必要となる. kr はできるだけ小さな値とするべきである. しかし, kr を n より小さくできない場合 (例えば, 1次元ラプラシアンを離散化した行列) がある. 他方, kr を n より大きくすることは勧められない. kr と n が等しいときも本ルーチンは正しく動作するが, 計算時間は長くなってしまう.

計算された固有値・固有ベクトルの精度は, かなりクリーロフ部分空間の次元数 kr と初期ベクトルに依存する.

e) 固有値が密集している(cluster)

固有値が密集している(cluster)とは, 隣り合う固有値の距離が丸め誤差単位のオーダーである固有値の集まりのことである.

4. 使用例

実対称スパース行列の固有値・固有ベクトルの最大及び最小の固有値と対応する固有ベクトルを求める.

```
#include <stdio.h>
#include <stdlib.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NDIM 15
```

```

#define NDIAG 5
#define NMIN 1
#define NMAX 1
#define NEV NMIN+NMAX
#define NLMIN 2*NMIN
#define NLMAX 2*NMAX
#define NEVL NLMIN+NLMAX

#define max(a,b) ((a) > (b) ? (a) : (b))

MAIN__()
{
    int ierr, icon;
    int n, i, j, k;
    int ivec, ix, nmin, nmax, nlmin, nlmax, kr, maxc, ncmin, ncmax;
    double a[NDIAG][NDIM], e[NEVL], ev[NEVL][NDIM];
    int ndiag, nofst[NDIAG], indx[NEV], *iw, mnl, md;
    double *wv, eps;

    /* initialize matrix */
    ndiag = NDIAG;
    n = NDIM;
    k = NDIM;

    for (i=0;i<n;i++) {
        a[0][i] = -6;
        a[1][i] = -3;
        a[2][i] = 10;
        a[3][i] = -3;
        a[4][i] = -6;
    }
    a[0][0] = 0;
    a[0][1] = 0;
    a[1][0] = 0;
    a[3][n-1] = 0;
    a[4][n-2] = 0;
    a[4][n-1] = 0;
    nofst[0] = -2;
    nofst[1] = -1;
    nofst[2] = 0;
    nofst[3] = 1;
    nofst[4] = 2;
    ivec = 0;
    ix = 1;
    eps = 1e-6;
    nmin = NMIN;
    nmax = NMAX;
    nlmin = NLMIN;
    nlmax = NLMAX;
    kr = n;
    maxc = 10;
    mnl = max(nlmin,nlmax);
    md = nlmin+nlmax;
    wv = (double*)malloc(((maxc+mnl)*(kr+2)+md*(kr+1)+7*k+14*(kr+1))
                        *sizeof(double));
    iw = (int*)malloc((11*(maxc+mnl)+md+128)*sizeof(int));
    /* find eigenvalues and eigenvectors */
    ierr = c_dvland((double*)a, k, ndiag, n, nofst, ivec, ix, eps, nmin, nmax,
                  nlmin, nlmax, kr, maxc, e, indx, &ncmin, &ncmax,
                  (double*)ev, wv, iw, &icon);
    printf("icon = %i\n", icon);
    /* print smallest eigenvalues and eigenvectors */
    for (i=0;i<ncmin;i++) {
        printf("eigenvalue: %7.4f\n", e[indx[i]]);
        printf("eigenvector: ");
        for (j=0;j<n;j++)
            printf("%7.4f ", ev[indx[i]][j]);
        printf("\n");
    }
    /* print largest eigenvalues and eigenvectors */
    for (i=0;i<ncmax;i++) {
        printf("eigenvalue: %7.4f\n", e[indx[NEV-ncmax+i]]);
        printf("eigenvector: ");
        for (j=0;j<n;j++)
            printf("%7.4f ", ev[indx[NEV-ncmax+i]][j]);
        printf("\n");
    }
}

```

```
    free(wv);  
    free(iw);  
    return(0);  
}
```

5. 手法概要

Lanczos 法については[25], [42]を参照のこと. 本ルーチンで使われているアルゴリズムは行列 A の次数より小さくて近い等しい次数の三重対角行列を生成して, 次にこの三重対角行列の固有値・固有ベクトル (Ritz 固有ベクトル) をマルチセクションによるスツルムの方法と逆反復法で求める (“FUJITSU SSL II 拡張機能使用手引書 II”の VTDEV の項目参照). 最後に, 行列 A の固有ベクトルを Lanczos 法で生成されたクリーロフ部分空間のベクトルと Ritz 固有ベクトルを用いて計算する.

c_dvlax

実行列の連立 1 次方程式 (ブロッキング LU 分解法)
<code>ierr = c_dvlax(a, k, n, b, epsz, isw, &is, vw, ip, &icon);</code>

1. 機能

実係数連立 1 次方程式をブロッキング LU 分解法 (ガウスの消去法) で解く。

$$\mathbf{Ax} = \mathbf{b}$$

ただし、 $\mathbf{A}$ は $n \times n$ の正則な実行列、 $\mathbf{b}$ は n 次元の実定数ベクトル、 $\mathbf{x}$ は n 次元の解ベクトルである。 $n \geq 1$ であること。

2. 引数

呼出し形式:

```
ierr = c_dvlax((double*)a, k, n, b, epsz, isw, &is, vw, ip, &icon);
```

引数の説明:

a	double a[n][k]	入力	係数行列 $\mathbf{A}$. 演算後, 内容は保存されない.
k	int	入力	配列 a の整合寸法 ($\geq n$).
n	int	入力	係数行列 $\mathbf{A}$ の次数 n .
b	double b[n]	入力	定数ベクトル $\mathbf{b}$.
		出力	解ベクトル $\mathbf{x}$.
epsz	double	入力	ピボットの相対零判定値 (≥ 0.0). 0.0 のときは標準値が採用される. (“使用上の注意” a)参照).
isw	int	入力	制御情報. 同一の係数行列を持つ $l (\geq 1)$ 組の方程式を解くとき, 次のとおり指定する. $isw = 1$ のとき 1 組目の方程式を解く. $isw = 2$ のとき 2 組目以降の方程式を解く. ただしこのとき $\mathbf{b}$ の値だけを新しい定数ベクトル $\mathbf{b}$ の値に変え, それ以外の引数はそのまま使う. (“使用上の注意” b)参照).
is	int	出力	行列 $\mathbf{A}$ の行列式を求めるための情報. 演算後の配列 a の n 個の対角要素と is の値を掛け合わせると行列式が得られる.
vw	double vw[n]	作業領域	
ip	int ip[n]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	係数行列のある行の要素がすべて零であったか又はピボットが相対的に零となった. 係数行列は非正則の可能性が高い.	処理を打ち切る.

コード	意味	処理内容
30000	次のいずれかであった. • $k < n$ • $n < 1$ • $\text{epsz} < 0$ • $\text{isw} \neq 1, 2$	処理を打ち切る.

3. 使用上の注意

a) **epsz** について

ピボットの相対零判定値 **epsz** に値を設定したとすると、この値は次の意味を持っている。すなわち、選択されたピボット要素が、実行列 $A = (a_{ij})$ の絶対値最大要素 $\max |a_{ij}|$ と **epsz** の積以下なら、

$$|a_{kk}^k| \leq \max |a_{ij}| \text{ epsz}$$

そのピボットを相対的に零とみなし、**icon** = 20000 として処理を打ち切る。**epsz** の標準値は丸め誤差の単位を μ としたとき、**epsz** = 16μ である。

なお、ピボットが小さくなくても計算を続行させる場合には、**epsz** へ極小の値を与えればよいが、その結果は保証されない。

b) **isw** について

同一の係数行列を持つ連立 1 次方程式をいくつか続けて解く場合には、2 回目以降 **isw** = 2 として解くと、係数行列 **A** の LU 分解過程を省略するので計算時間が少なくなる。なお、この場合 **is** の値は、**isw** = 1 のときの値が保障される。

4. 使用例

行列 **A** とベクトル **x** の積 **b** を求め、**Ax = b** を解く。得られた解と元の **x** の値を比較し、結果を確認する。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 100

MAIN__()
{
    int ierr, icon;
    int n, i, j, k, isw, is;
    double epsz, eps;
    double a[NMAX][NMAX], b[NMAX], x[NMAX], vw[NMAX];
    int ip[NMAX];

    /* initialize matrix and vector */
    n = NMAX;
    for (i=0; i<n; i++)
        for (j=i; j<n; j++) {
            a[i][j] = n-j;
            a[j][i] = n-j;
        }
    for (i=0; i<n; i++)
        x[i] = i+1;
    k = NMAX;
    /* initialize constant vector b = a*x */
    ierr = c_dmav((double*)a, k, n, n, x, b, &icon);
    epsz = 1e-6;
    isw = 1;
    /* solve system of equations */
    ierr = c_dvlax((double*)a, k, n, b, epsz, isw, &is, vw, ip, &icon);
    if (icon != 0) {
```

```
    printf("ERROR: c_dvlax failed with icon = %d\n", icon);
    exit(1);
}
/* check solution vector */
eps = 1e-6;
for (i=0;i<n;i++)
    if (fabs((x[i]-b[i])/b[i]) > eps) {
        printf("WARNING: result inaccurate\n");
        exit(1);
    }
printf("Result OK\n");
return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書”の VLAX の項目を参照のこと。

c_dvlbx

実バンド行列の連立 1 次方程式(ガウス消去法)

<pre>ierr = c_dvlbx(a, n, nh1, nh2, b, epsz, isw, &is, ip, vw, &icon);</pre>

1. 機能

実係数連立 1 次方程式をガウス消去法で解く.

$$\mathbf{Ax} = \mathbf{b}$$

ただし, $\mathbf{A}$ は $n \times n$, 下バンド幅 h_1 , 上バンド幅 h_2 のバンド行列. $\mathbf{b}$ は n 次元の実定数ベクトル, $\mathbf{x}$ は n 次元の解ベクトル. ($n > h_1 \geq 0, n > h_2 \geq 0$).

2. 引数

呼出し形式:

```
ierr = c_dvlbx(a, n, nh1, nh2, b, epsz, isw, &is, ip, vw, &icon);
```

引数の説明:

a	double a[Alen]	入力	バンド係数行列 $\mathbf{A}$ をバンド行列用圧縮格納法で格納する. $Alen = (2*nh1+nh2+1)*n$. (“使用上の注意” a)参照).
		出力	LU 分解された $\mathbf{L}$ 及び $\mathbf{U}$ が格納される.
n	int	入力	行列 $\mathbf{A}$ の次数 n .
nh1	int	入力	行列 $\mathbf{A}$ の下バンド幅 h_1 .
nh2	int	入力	行列 $\mathbf{A}$ の上バンド幅 h_2 .
b	double b[n]	入力	定数ベクトル $\mathbf{b}$.
		出力	解ベクトル $\mathbf{x}$.
epsz	double	入力	ピボットの相対零判定値 (≥ 0.0). 0.0 のときは, 標準値が設定される. (“使用上の注意” b)参照).
isw	int	入力	制御情報. 同一係数行列を k ($k \geq 1$) 組の方程式を解くとき, 次のとおり指定する. $isw = 1$ のとき 1 組目の方程式を解く. $isw = 2$ のとき 2 組目以降の方程式を解く. ただし, このとき $\mathbf{b}$ の値だけを新しい定数ベクトル $\mathbf{b}$ の値に変え, それ以外の引数はそのまま使用する. (“使用上の注意” c)参照).
is	int	出力	行列 $\mathbf{A}$ の行列式を求めるための情報. (“使用上の注意” d)参照).
ip	int ip[n]	出力	部分ピボットリングによる行の入替えの履歴を示すトランスポジションベクトル.
vw	double vw[n]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	行列 A のある行の要素がすべて零であったか, 又はピボットが相対的に零となった. 行列 A は非正則の可能性が強い.	処理を打ち切る.
30000	次のいずれかであった. • $n \leq nh1$ • $n \leq nh2$ • $nh1 < 0$ • $nh2 < 0$ • $epsz < 0$ • $isw \neq 1, 2$	処理を打ち切る.

3. 使用上の注意

a) **a** について

バンド行列 **A** はバンド行列用圧縮格納法で格納する. 格納方法の詳細については, “1.3.4 バンド行列の格納方法”を参照のこと.

b) **epsz** について

本関数では, **epsz** よりピボットの値が小さいとき相対的に零とみなし, **icon** = 20000 として処理を打ち切る. **epsz** の標準値は丸め誤差の単位を μ としたとき, $epsz = 16\mu$ である.

c) **isw** について

同一係数行列を持つ連立 1 次方程式をいくつか続けて解く場合には, 2 回目以降 **isw** = 2 として解くと, 係数行列 **A** の LU 分解過程を省略するので計算時間が少なくなる.

d) 行列式

係数行列の行列式の値を求めるには, **a** の対角線上の n 個の値, つまり $a[(2*h1+h2+1)*i+h1]$, $i = 0, \dots, n-1$ と **is** の積を計算すればよい.

e) データ格納領域

本関数では, バンド行列の特性を考慮して, データ格納領域を節約できるように, バンド行列を格納しているが, バンド幅の大きさによっては, **c_dvalu** よりデータ格納領域を多く必要とする場合がある. そのときは, **c_dvalu** を用いたほうがデータ格納領域を節約できる. 本関数の特性を生かせるのは, $n > 2*h_1 + h_2 + 1$ のときである.

4. 使用例

行列 **A** とベクトル **x** の積 **b** を求め, $\mathbf{Ax} = \mathbf{b}$ を解く. 得られた解と元の **x** の値を比較し, 結果を確認する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define min(i,j) (i<j) ? i : j
#define max(i,j) (i>j) ? i : j

#define NMAX 100
#define H1MAX 2
#define H2MAX 2
```

```
MAIN__()
{
    int ierr, icon;
    int n, nh1, nh2, i, j, jmin, jmax, isw, is, ip[NMAX];
    double epsz, eps, sum;
    double a[(2*H1MAX+H2MAX+1)*NMAX], b[NMAX], x[NMAX], vw[NMAX];

    /* initialize matrix */
    n = NMAX;
    nh1 = H1MAX;
    nh2 = H2MAX;
    for (i=0;i<n*(2*nh1+nh2+1);i++)
        a[i] = 0;
    for (i=0;i<n;i++) {
        jmin = max(i-nh1,0);
        jmax = min(i+nh2,n-1);
        for (j=jmin;j<=jmax;j++)
            a[i*(2*nh1+1+nh2)+j-i+nh1] = n-fabs(j-i);
    }
    for (i=0;i<n;i++) {
        x[i] = i+1;
    }
    /* initialize constant vector b = a*x */
    for (i=0;i<n;i++) {
        jmin = max(i-nh1,0);
        jmax = min(i+nh2,n-1);
        sum = 0;
        for (j=jmin;j<=jmax;j++)
            sum = sum + a[i*(2*nh1+1+nh2)+j-i+nh1]*x[j];
        b[i] = sum;
    }
    epsz = 1e-6;
    isw = 1;
    /* solve system of equations */
    ierr = c_dvlbx(a, n, nh1, nh2, b, epsz, isw, &is, ip, vw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dvlbx failed with icon = %d\n", icon);
        exit(1);
    }
    /* check solution vector */
    eps = 1e-6;
    for (i=0;i<n;i++)
        if (fabs((x[i]-b[i])/b[i]) > eps) {
            printf("WARNING: result inaccurate\n");
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書 II”の VLBX の項目及び [42]を参照のこと。

c_dvldiv

LDL ^T 分解された正値対称行列の逆行列

ierr = c_dvldiv(a, n, vw, &icon);

1. 機能

LDL^T 分解された $n \times n$ の正値対称行列 $\mathbf{A}$ の逆行列 $\mathbf{A}^{-1}$ を求める.

$$\mathbf{A}^{-1} = (\mathbf{L}^T)^{-1} \mathbf{D}^{-1} \mathbf{L}^{-1}$$

ただし, $\mathbf{L}, \mathbf{D}$ はそれぞれ $n \times n$ の単位下三角行列と対角行列である. $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dvldiv(a, n, vw, &icon);
```

引数の説明:

a	double a[Alen]	入力	行列 $\mathbf{L}$ と行列 $\mathbf{D}^{-1}$ (c_dvsldl の出力). 正値対称行列用圧縮格納法. 格納法の詳細については“1.3.2 対称行列の格納方法”を参照のこと. $Alen = n(n+1)/2$.
		出力	逆行列 $\mathbf{A}^{-1}$ の下三角部分が正値対称行列用圧縮格納法で格納される.
n	int	入力	行列 $\mathbf{L}, \mathbf{D}$ の次数 n .
vw	double vw[n]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	行列が正値でなかった.	処理は続行する.
30000	$n < 1$	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

本関数は, LDL^T 分解された正値対称行列を入力して, 分解された元の正値対称行列の逆行列を計算する. LDL^T 分解は関数 c_dvsldl により行い, 続けて本関数を呼び出すことにより逆行列を求めることができる.

連立 1 次方程式を解く場合には, 関数 c_dvlsx を用いること. 逆行列を求めて解くことは, c_dvlsx を用いるときよりも演算回数が多くなるので避けた方がよい. 本関数は逆行列が必要なときだけ使用すること.

4. 使用例

正値対称行列を LDL^T 分解し, 逆行列を計算し, 結果を確認する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 100
```

```
MAIN__()
{
    int ierr, icon;
    int n, i, j, ij;
    double epsz, eps, sum;
    double a[NMAX*(NMAX+1)/2], b[NMAX], x[NMAX], y[NMAX], vw[2*NMAX];
    int ivw[NMAX];

    /* initialize matrix and vector */
    n = NMAX;
    ij = 0;
    for (j=0;j<n;j++)
        for (i=j;i<n;i++)
            a[ij++] = n-i;
    for (i=0;i<n;i++) {
        x[i] = i+1;
        b[i] = 0;
        y[i] = 0;
    }
    /* initialize constant vector b = a*x */
    ij = 0;
    for (i=0;i<n;i++) {
        sum = a[ij++] * x[i];
        for (j=i+1;j<n;j++) {
            b[j] = b[j] + a[ij] * x[i];
            sum = sum + a[ij++] * x[j];
        }
        b[i] = b[i] + sum;
    }
    epsz = 1e-6;
    /* LDL decomposition of system of equations */
    ierr = c_dvsldl(a, n, epsz, vw, ivw, &icon);
    if (icon > 10000) {
        printf("ERROR: c_dvsldl failed with icon = %d\n", icon);
        exit(1);
    }
    /* find matrix inverse from LDL factors */
    ierr = c_dvldiv(a, n, vw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dvldiv failed with icon = %d\n", icon);
        exit(1);
    }
    /* calculate y = a*b */
    ij = 0;
    for (i=0;i<n;i++) {
        sum = a[ij++] * b[i];
        for (j=i+1;j<n;j++) {
            y[j] = y[j] + a[ij] * b[i];
            sum = sum + a[ij++] * b[j];
        }
        y[i] = y[i] + sum;
    }
    /* compare x and y */
    eps = 1e-6;
    for (i=0;i<n;i++)
        if (fabs((x[i]-y[i])/y[i]) > eps) {
            printf("WARNING: result inaccurate\n");
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の LDIV の項目及び[71]を参照のこと。なお、LDIV と本関数では行列の格納方法が異なっているが、基本的な計算方法は同じである。

c_dvldlx

LDL ^T 分解された正値対称行列の連立 1 次方程式

ierr = c_dvldlx(b, fa, n, &icon);

1. 機能

LDL^T 分解された正値対称な係数行列を持つ連立 1 次方程式(1)を解く.

$$\mathbf{LDL}^T \mathbf{x} = \mathbf{b} \quad (1)$$

ただし, $\mathbf{L}$, $\mathbf{D}$ はそれぞれ $n \times n$ の単位下三角行列と対角行列, $\mathbf{b}$ は n 次元の実定数ベクトル, $\mathbf{x}$ は n 次元の解ベクトルである. $n \geq 1$ であること.

本関数は, 関数 c_dvsldl により LDL^T 分解された行列を受けて求解処理を行うものである

2. 引数

呼出し形式:

```
ierr = c_dvldlx(b, fa, n, &icon);
```

引数の説明:

b	double b[n]	入力	定数ベクトル $\mathbf{b}$.
		出力	解ベクトル $\mathbf{x}$.
fa	double fa[Falen]	入力	行列 $\mathbf{D}^{-1} + (\mathbf{L} - \mathbf{I})$. 正値対称行列用圧縮格納法. 格納法の詳細については“1.3.2 対称行列の格納方法”を参照のこと. $Falen = n(n+1)/2$.
n	int	入力	行列 $\mathbf{L}$, $\mathbf{D}$ の次数 n .
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
10000	系列行列が正値でなかった.	処理は続行する.
30000	$n < 1$	処理を打ち切る.

3. 使用上の注意

本関数を, 関数 c_dvsldl に続けて呼び出すことにより連立 1 次方程式を解くことができる. しかし, 通常は, 関数 c_dvlsx を呼び出せば, 一度に解が求められる.

4. 使用例

正値対称行列を LDL^T 分解し, 続いて連立 1 次方程式を解く.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 100

MAIN__()
{
    int ierr, icon;
    int n, i, j, ij;
```

```
double epsz, eps, sum;
double a[NMAX*(NMAX+1)/2], b[NMAX], x[NMAX], vw[2*NMAX];
int ivw[NMAX];

/* initialize matrix and vector */
n = NMAX;
ij = 0;
for (j=0;j<n;j++)
    for (i=j;i<n;i++)
        a[ij++] = n-i;
for (i=0;i<n;i++) {
    x[i] = i+1;
    b[i] = 0;
}
/* initialize constant vector b = a*x */
ij = 0;
for (i=0;i<n;i++) {
    sum = a[ij++]*x[i];
    for (j=i+1;j<n;j++) {
        b[j] = b[j] + a[ij]*x[i];
        sum = sum + a[ij++]*x[j];
    }
    b[i] = b[i]+sum;
}
epsz = 1e-6;
/* LDL decomposition of system of equations */
ierr = c_dvsldl(a, n, epsz, vw, ivw, &icon);
if (icon > 10000) {
    printf("ERROR: c_dvsldl failed with icon = %d\n", icon);
    exit(1);
}
/* solve decomposed system of equations */
ierr = c_dvldlx(b, a, n, &icon);
if (icon > 10000) {
    printf("ERROR: c_dvldlx failed with icon = %d\n", icon);
    exit(1);
}
/* check solution vector */
eps = 1e-6;
for (i=0;i<n;i++)
    if (fabs((x[i]-b[i])/b[i]) > eps) {
        printf("WARNING: result inaccurate\n");
        exit(1);
    }
printf("Result OK\n");
return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書”の VLDLX の項目を参照のこと。

c_dvlsbx

正値対称バンド行列の連立 1 次方程式(変形コレスキー法)

```
ierr = c_dvlsbx(a, n, nh, b, epsz, isw,
                &icon);
```

1. 機能

実係数連立 1 次方程式(1)を変形コレスキー法で解く.

$$\mathbf{Ax} = \mathbf{b} \quad (1)$$

ただし $\mathbf{A}$ は $n \times n$, 上, 下バンド幅 h の正値対称バンド行列, $\mathbf{b}$ は n 次元の実定数ベクトル, $\mathbf{x}$ は n 次元の解ベクトル. $n > h \geq 0$ を満たさなければならない.

2. 引数

呼出し形式:

```
ierr = c_dvlsbx(a, n, nh, b, epsz, isw, &icon);
```

引数の説明:

a	double a[Alen]	入力	Alen=(nh+1)*n. 係数行列 $\mathbf{A}$ の対角要素とバンド行列の下三角部分を正値対称バンド行列用圧縮格納法で格納する. (“使用上の注意” a)参照).
		出力	$\mathbf{LDL}^T$ 分解された $\mathbf{D}$ 及び $\mathbf{L}$ を格納する. (“使用上の注意” e)参照).
n	int	入力	行列 $\mathbf{A}$ の次数 n .
nh	int	入力	下バンド幅 h .
b	double b[n]	入力	定数ベクトル $\mathbf{b}$.
		出力	解ベクトル $\mathbf{x}$.
epsz	double	入力	ピボットの相対零判定値(≥ 0). 0.0 のときは標準値が採用される. (“使用上の注意” b)参照).
isw	int	入力	制御情報. 同一の係数行列を持つ $k (\geq 1)$ 組の方程式を解くとき, 次のとおり指定する. isw=1 のとき 1 組目の方程式を解く. isw=2 のとき 2 組目以降の方程式を解く. ただし, このとき $\mathbf{b}$ の値だけを新しい定数ベクトル $\mathbf{b}$ の値に変え, それ以外の引数はそのまま使う. (“使用上の注意” c)参照).
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
10000	ピボットが負となった. 行列 $\mathbf{A}$ は正値でない.	処理は続行する.
20000	ピボットが相対的に零となった. 行列 $\mathbf{A}$ は非正則の可能性が強い.	処理を打ち切る.

コード	意味	処理内容
30000	次のいずれかであった。 • $nh < 0$ • $nh \geq n$ • $epsz < 0$ • $isw \neq 1, 2$	処理を打ち切る.

3. 使用上の注意

a) **a** について

正値対称バンド行列 **A** は正値対称バンド行列用圧縮格納法で格納する. 格納方法については, “1.3.4 バンド行列の格納方法”を参照のこと.

b) **epsz** について

本関数では, **epsz** よりピボットの値が小さいとき相対的に零とみなし, **icon** = 20000 として処理を打ち切る. **epsz** の標準値は丸め誤差の単位を μ としたとき, **epsz** = 16μ である.

c) **isw** について

同一係数行列を持つ連立一次方程式をいくつか続けて解く場合には, 2 回目以降 **isw** = 2 として解くと, 係数行列 **A** の **LDL^T** 分解過程を省略するので計算時間が少なくなる.

d) ピボットが負の場合

分解過程でピボットが負になった場合, 係数行列は正値でないことになる. このとき, **icon** = 10000 とするが処理は続行する.

e) 行列式

行列 **D** 及び **L** の要素は配列 **a** に格納される. d_{ii} が $a[(h+1)*(i-1)]$ に格納され, l_{ji} が $a[(h+1)*(i-1)+j-i]$ に格納される. ここで, $i=1, \dots, n, j=i+1, \dots, i+h$ である. 係数行列の行列式の値を求めるには, **A** の対角線上の n 個の値, つまり $a[(h+1)*(i-1)], i=1, \dots, n$ の積を計算すればよい.

4. 使用例

行列 **A** とベクトル **x** の積 **b** を求め, **Ax = b** を解く. 得られた解と元の **x** の値を比較し, 結果を確認する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define min(i,j) (i<j) ? i : j

#define NMAX 100
#define HMAX 2

MAIN__()
{
    int ierr, icon;
    int n, nh, i, j, jmax, imax, isw;
    double epsz, eps, sum;
    double a[(HMAX+1)*NMAX], b[NMAX], x[NMAX];

    /* initialize matrix */
    n = NMAX;
    nh = HMAX;
    for (j=0; j<n; j++) {
        imax = min(j+nh, n-1);
        for (i=j; i<=imax; i++)
            a[j*(nh+1)+i-j] = n-(j-i);
    }
}
```

```

for (i=0;i<n;i++) {
    x[i] = i+1;
    b[i] = 0;
}
/* initialize constant vector b = a*x */
for (i=0;i<n;i++) {
    sum = a[i*(nh+1)]*x[i];
    jmax = min(i+nh,n-1);
    for (j=i+1;j<=jmax;j++) {
        b[j] = b[j] + a[i*nh+j]*x[i];
        sum = sum + a[i*nh+j]*x[j];
    }
    b[i] = b[i]+sum;
}
epsz = 1e-6;
isw = 1;
/* solve system of equations */
ierr = c_dvlsbx(a, n, nh, b, epsz, isw, &icon);
if (icon > 10000) {
    printf("ERROR: c_dvlsbx failed with icon = %d\n", icon);
    exit(1);
}
/* check solution vector */
eps = 1e-6;
for (i=0;i<n;i++)
    if (fabs((x[i]-b[i])/b[i]) > eps) {
        printf("WARNING: result inaccurate\n");
        exit(1);
    }
printf("Result OK\n");
return(0);
}

```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書 II”の VLSBX の項目及び[42], [79]を参照のこと。

c_dvlspx

正値対称行列の連立 1 次方程式 (ブロック化されたコレスキー分解法)

ierr = c_dvlspx(a, k, n, b, epsz, isw, &icon);

1. 機能

実係数連立 1 次方程式(1)の係数行列 **A** を, ブロック化された外積形のコレスキー分解法で, (2)のように分解して方程式を解く.

$$\mathbf{Ax} = \mathbf{b} \quad (1)$$

$$\mathbf{A} = \mathbf{LL}^T \quad (2)$$

ただし, **A** は $n \times n$ の正値対称行列, **b** は n 次元の実定数ベクトル, **x** は n 次元の解ベクトル, **L** は下三角行列である. $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dvlspx((double*)a, k, n, b, epsz, isw, &icon);
```

引数の説明:

a	double a[n][k]	入力	係数行列 A の上三角部分 $a_{ij}, i \leq j$ を, a の上三角部分 a[i-1][j-1], $i \leq j$ に格納する. 演算後, 内容は保存されない. 格納方法の詳細については, 図 dvlspx-1.を参照のこと.
		出力	a の上三角部分 a[i-1][j-1], $i \leq j$ に $\mathbf{L}^T(l_{ij}, i \leq j)$ が格納される.
k	int	入力	配列 a の整合寸法. ($\geq n$)
n	int	入力	係数行列 A の次数 n .
b	double b[n]	入力	定数ベクトル b .
		出力	解ベクトル x .
epsz	double	入力	ピボットの相対零判定値(≥ 0.0). 0.0 のときは標準値が採用される. (“使用上の注意” a)参照).
isw	int	入力	制御情報. 同一の係数行列を持つ複数組の方程式を解くとき, 次のとおり指定する. isw=1 のとき 1 組目の方程式を解く. isw=2 のとき 2 組目以降の方程式を解く. ただし, このとき b の値だけを新しい定数ベクトル b の値に変え, それ以外の引数はそのまま使用する. (“使用上の注意” b)参照).
icon	int	出力	コンディションコード. 下の表を参照.

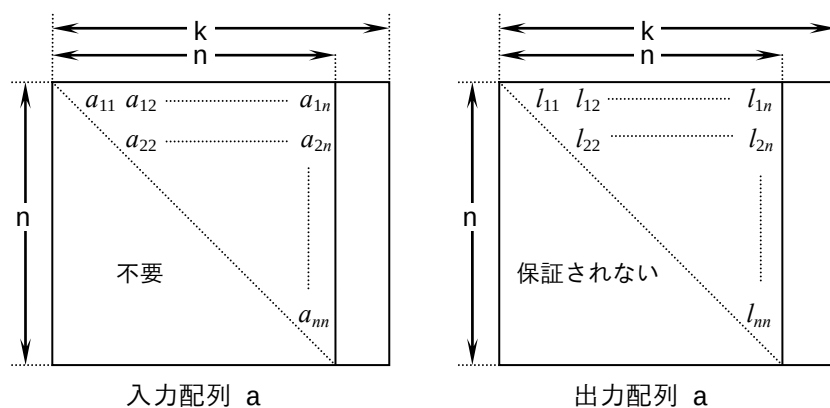


図 dvlspx-1. コレスキー分解法のデータの格納方法

LL^T 分解を行う正値対称行列の対角要素および上三角部分 a_{ij} を配列 $a[i-1][j-1]$, $i=1, \dots, n$, $j=i, \dots, n$ に格納する. LL^T 分解された結果は, 上三角行列 L^T が上三角部分に格納される. 下三角部分の値は保証されない.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	ピボットが相対的に零となった. 係数行列は非正則の可能性が強い.	処理を打ち切る.
20100	ピボットが負となった. 係数行列は正値でない.	
30000	次のいずれかであった. • $n < 1$ • $\text{epsz} < 0$ • $k < n$ • $\text{isw} \neq 1, 2$	

3. 使用上の注意

a) epsz について

ピボットの相対零判定値 epsz に 10^{-5} を設定したとすると, この値は次の意味を持っている. すなわち, コレスキー分解法による LL^T 分解の過程でピボットの値が正であり, その値より小さくなった場合に, そのピボットの値を相対的に零とみなし, $\text{icon} = 20000$ として処理を打ち切る. epsz の標準値は丸め誤差の単位を μ としたとき, $\text{epsz} = 16\mu$ である.

なお, ピボットが小さくなくても計算を続行させる場合には, epsz へ極小の値を与えればよいが, その結果は保証されない.

b) isw について

同一の係数行列を持つ連立 1 次方程式をいくつか続けて解く場合には, 2 回目以降 $\text{isw} = 2$ として解くと, 係数行列 A の LL^T 分解過程を省略するので計算時間が少なくなる.

c) ピボットが負の場合

分解過程でピボットが負になった場合, 係数行列は正値でないことになる. このとき, $\text{icon} = 20100$ として処理を打ち切る.

d) 行列式

係数行列の行列式の値を求めるには、演算後の配列 a の n 個の対角要素を掛け合わせ、その 2 乗をとればよい。

4. 使用例

2000 × 2000 の係数行列の連立 1 次方程式を解く。

```
#include <stdio.h>
#include <stdlib.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX      2000
#define KMAX      NMAX+1

MAIN__()
{
    int    epsz, isw, icon, ierr, i, j;
    double a[NMAX][KMAX], b[NMAX];

    for (i=0; i<NMAX; i++) {
        for (j=i; j<NMAX; j++) {
            a[i][j] = i+1;
        }
    }

    for (i=0; i<NMAX; i++) {
        b[i] = (i+1)*(i+2)/2+(i+1)*(NMAX-i-1);
    }

    isw = 1, epsz = 1e-13;
    ierr = c_dvlspx((double*)a, KMAX, NMAX, b, epsz, isw, &icon);

    if (icon != 0) {
        printf("ERROR: c_dvlspx failed with icon = %d\n", icon);
        exit(1);
    }

    printf ("Solution vector\n");
    for (i=0; i<10; i++) {
        printf ("b[%d] = %15.10le\n", i, b[i]);
    }
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書 II”の VLSPX の項目を参照のこと。

c_dvlsx

正値対称行列の連立 1 次方程式(変形コレスキー法)

<pre>ierr = c_dvlsx(a, n, b, epsz, isw, vw, ivw, &icon);</pre>

1. 機能

実係数連立 1 次方程式を変形コレスキー法で解く。

$$\mathbf{Ax} = \mathbf{b}$$

ただし、 $\mathbf{A}$ は $n \times n$ の正値対称行列、 $\mathbf{b}$ は n 次元の実定数ベクトル、 $\mathbf{x}$ は n 次元の解ベクトルである。 $n \geq 1$ であること。

2. 引数

呼出し形式:

```
ierr = c_dvlsx(a, n, b, epsz, isw, vw, ivw, &icon);
```

引数の説明:

a	double a[Alen]	入力	係数行列 $\mathbf{A}$ を正値対称行列用圧縮格納法で格納する。演算後、内容は保存されない。Alen = n(n+1)/2。格納方法の詳細については“1.3.2 対称行列の格納方法”を参照のこと。
n	int	入力	係数行列 $\mathbf{A}$ の次数 n 。
b	double b[n]	入力	定数ベクトル $\mathbf{b}$ 。
		出力	解ベクトル $\mathbf{x}$ 。
epsz	double	入力	ピボットの相対零判定値(≥ 0.0)。0.0 のときは標準値が採用される。(“使用上の注意” a)参照)。
isw	int	入力	制御情報。同一の係数行列を持つ複数組の方程式を解くとき、次のとおり指定する。 isw=1 のとき、1 組目の方程式を解く。 isw=2 のとき、2 組目以降の方程式を解く。ただし、このとき $\mathbf{b}$ だけを新しい定数ベクトル $\mathbf{b}$ の値に変え、それ以外の引数はそのまま使用する。(“使用上の注意” b)参照)。
vw	double vw[2*n]	作業領域	
ivw	int ivw[n]	作業領域	
icon	int	出力	コンディションコード。下の表を参照。

コンディションコード:

コード	意味	処理内容
0	エラーなし。	正常終了。
10000	ピボットが負となった。係数行列は正値でない。	処理は続行する。
20000	ピボットが相対的に零となった。係数行列は非正則の可能性が強い。	処理を打ち切る。

コード	意味	処理内容
30000	次のいずれかであった。 • $n < 1$ • $\text{epsz} < 0$ • $\text{isw} \neq 1, 2$	処理を打ち切る.

3. 使用上の注意

a) epsz について

ピボットの相対零判定値 epsz に 10^{-5} を設定したとすると, この値は次の意味を持っている. すなわち, 変形コレスキー法による LDL^T 分解の過程でピボットの値が 10 進 s 桁以上の桁落ちを生じた場合にそのピボットを相対的に零とみなし, $\text{icon} = 20000$ として処理を打ち切る. epsz の標準値は丸め誤差の単位を μ としたとき, $\text{epsz} = 16\mu$ である.

なお, ピボットが小さくなくても計算を続行させる場合には, epsz へ極小の値 (例えば 10^{-70}) を与えればよいが, その結果は保証されない.

b) isw について

同一の係数行列を持つ連立 1 次方程式をいくつか続けて解く場合には, 2 回目以降 $\text{isw} = 2$ として解くと, 係数行列 $\mathbf{A}$ の LDL^T 分解過程を省略するので計算時間が少なくなる.

c) ピボットが負の場合

分解過程でピボットが負になった場合, 係数行列は正值でないことになる. このとき, $\text{icon} = 10000$ とするが処理は続行する. ただし, ピボットリングを行っていないため計算誤差は大きい可能性があるので注意が必要である.

d) 行列式

係数行列の行列式の値を求めるには, 演算後の配列 $\mathbf{a}$ の n 個の対角線上の値 ($\mathbf{D}^{-1}$ の対角要素) を掛け合わせ, その逆数をとればよい.

4. 使用例

行列 $\mathbf{A}$ とベクトル $\mathbf{x}$ の積 $\mathbf{b}$ を求め, $\mathbf{Ax} = \mathbf{b}$ を解く. 得られた解と元の $\mathbf{x}$ の値を比較し, 結果を確認する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 100

MAIN__()
{
    int ierr, icon;
    int n, i, j, ij, isw;
    double epsz, eps, sum;
    double a[NMAX*(NMAX+1)/2], b[NMAX], x[NMAX], vw[2*NMAX];
    int ivw[NMAX];

    /* initialize matrix and vector */
    n = NMAX;
    ij = 0;
    for (j=0; j<n; j++)
        for (i=j; i<n; i++)
            a[ij++] = n-i;
    for (i=0; i<n; i++) {
        x[i] = i+1;
        b[i] = 0;
    }
}
```

```
/* initialize constant vector b = a*x */
ij = 0;
for (i=0;i<n;i++) {
    sum = a[ij++]*x[i];
    for (j=i+1;j<n;j++) {
        b[j] = b[j] + a[ij]*x[i];
        sum = sum + a[ij++]*x[j];
    }
    b[i] = b[i]+sum;
}
epsz = 1e-6;
isw = 1;
/* solve system of equations */
ierr = c_dvlsx(a, n, b, epsz, isw, vw, ivw, &icon);
if (icon > 10000) {
    printf("ERROR: c_dvlsx failed with icon = %d\n", icon);
    exit(1);
}
/* check solution vector */
eps = 1e-6;
for (i=0;i<n;i++)
    if (fabs((x[i]-b[i])/b[i]) > eps) {
        printf("WARNING: result inaccurate\n");
        exit(1);
    }
printf("Result OK\n");
return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書”の VLSX の項目を参照のこと。

c_dvltqr

実 3 項行列の連立 1 次方程式 (QR 分解)

ierr = c_dvltqr(su, d, sl, n, b, vw, &icon);

1. 機能

実 3 重対角行列の連立 1 次方程式を QR 分解で解く。

$$\mathbf{T}\mathbf{x} = \mathbf{b}$$

$\mathbf{T}$ は $n \times n$ の正則な実 3 重対角行列, $\mathbf{b}$ は n 次元の実定数ベクトル, $\mathbf{x}$ は n 次元の解ベクトル. $n \geq 1$ であること. 3 重対角行列 $\mathbf{T}$ の要素を t_{ij} としたとき, d_i は対角要素を, $l_i = t_{i,i-1}$, $u_i = t_{i,i+1}$, は副対角要素を表すものとする. ただし, $l_1 = u_n = 0$.

2. 引数

呼出し形式:

```
ierr = c_dvltqr(su, d, sl, n, b, vw, &icon);
```

引数の説明:

su	double su[n]	入力	$\mathbf{T}$ の上副対角行列を su[i], i = 0,...,n-2 に格納する. su[n-1] = 0.
d	double d[n]	入力	$\mathbf{T}$ の対角要素.
sl	double sl[n]	入力	$\mathbf{T}$ の下副対角行列を sl[i], i = 1,...,n-1 に格納する. sl[0] = 0.
n	int	入力	3 重対角行列 $\mathbf{T}$ の次数 n .
b	double b[n]	入力	定数ベクトル $\mathbf{b}$.
vw	double vw[7n]	出力	解ベクトル $\mathbf{x}$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
10000	行列 $\mathbf{T}$ が特異に近い.	正常終了.
20000	行列 $\mathbf{T}$ が特異である.	処理を打ち切る.
30000	$n < 1$	処理を打ち切る.

3. 使用例

実 3 重対角行列の連立 1 次方程式を解く.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 100

MAIN__()
{
    int ierr, icon;
    int n, i;
    double eps, vw[7*NMAX];
    double sl[NMAX], d[NMAX], su[NMAX], b[NMAX], x[NMAX];
```

```

/* initialize matrix and vector */
n = NMAX;
for (i=0;i<n;i++) {
    sl[i] = -1;
    su[i] = -1;
    d[i] = 10;
}
sl[0] = 0;
su[n-1] = 0;
for (i=0;i<n;i++)
    x[i] = i+1;
/* initialize constant vector b=a*x */
b[0] = d[0]*x[0] + su[0]*x[1];
for (i=1;i<n-1;i++) {
    b[i] = sl[i]*x[i-1] + d[i]*x[i] + su[i]*x[i+1];
}
b[n-1] = sl[n-1]*x[n-2] + d[n-1]*x[n-1];
/* solve system of equations */
ierr = c_dvltqr(su, d, sl, n, b, vw, &ierr);
if (ierr > 10000) {
    printf("ERROR: c_dvltqr failed with ierr = %d\n", ierr);
    exit(1);
}
/* check solution vector */
eps = 1e-6;
for (i=0;i<n;i++)
    if (fabs((x[i]-b[i])/b[i]) > eps) {
        printf("WARNING: result inaccurate\n");
        exit(1);
    }
printf("Result OK\n");
return(0);
}

```

4. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書 II”の VLTQR の項目及び[42], [51]を参照のこと。

c_dvltx

実3項行列の連立1次方程式(サイクリック・リダクション法)

<pre>ierr = c_dvltx(sbd, d, spd, n, b, isw, ind, ivw, &icon);</pre>

1. 機能

実3項方程式,

$$\mathbf{Ax} = \mathbf{b}$$

をサイクリック・リダクション法で解く。ただし、 $\mathbf{A}$ は $n \times n$ の既約優対角性を満たす実3項行列、 $\mathbf{b}$ は n 次元の実定数ベクトル、 $\mathbf{x}$ は n 次元の解ベクトルである。 $n \geq 1$ であること。

ここで、 $\mathbf{A}$ が既約優対角であるとは、

$$\mathbf{A} = \begin{bmatrix} d_1 & f_1 & & 0 \\ e_2 & d_2 & f_2 & \\ & e_3 & \cdot & \cdot \\ & & & \cdot & f_{n-1} \\ 0 & & & e_n & d_n \end{bmatrix}$$

とするとき、

$$|d_i| \geq |e_i| + |f_i|, \quad i = 1, 2, \dots, n, \\ (\text{ただし } e_1 = f_n = 0)$$

が成り立ち、かつ、少なくとも一つの i に対して狭義の不等号が成立することである。ちなみに既約優対角行列は正則である。

2. 引数

呼出し形式:

```
ierr = c_dvltx(sbd, d, spd, n, b, isw, ind, ivw, &icon);
```

引数の説明:

sbd	double	入力	係数行列 $\mathbf{A}$ の下副対角部分. $\text{sbd}[i-1] = e_i, i = 2, \dots, n$ のように格納する。
	sbd[2n]	出力	演算後, 内容は保存されない. (“使用上の注意”a)参照)
d	double d[2n]	入力	係数行列 $\mathbf{A}$ の対角部分. $d[i-1] = d_i, i = 1, \dots, n$ ように格納する。
		出力	演算後, 内容は保存されない. (“使用上の注意”a)参照)
spd	double	入力	係数行列 $\mathbf{A}$ の上副対角部分. $\text{spd}[i-1] = f_i, i = 1, \dots, n-1$ のように格納する。
	spd[2n]	出力	演算後, 内容は保存されない. (“使用上の注意”a)参照)
n	int	入力	係数行列 $\mathbf{A}$ の次数 n .
b	double b[2n]	入力	定数ベクトル $\mathbf{b}$. $b[i-1] = b_i, i = 1, \dots, n$ ように格納する。
		出力	解ベクトル $\mathbf{x}$. $b[i-1] = x_i, i = 1, \dots, n$ ように格納される. (“使用上の注意”a)参照)
isw	int	入力	制御情報. 同一の係数行列を持つ複数組の方程式を解くとき、次のとおり指定する。 isw=1 のとき, 1 組目の方程式を解く。 isw=2 のとき, 2 組目以降の方程式を解く。

			ただし、このとき b の値だけを新しい定数ベクトル b の値に変え、それ以外の引数はそのまま使う。 ("使用上の注意"b)参照)
ind	int	入力	制御情報. 係数行列の既約優対角性の判定を行うか否かを次のとおり指定する. ind = 0 のとき, 判定を行う. ind = 1 のとき, 判定を行わない. 通常は, 0 と指定すればよい. ("使用上の注意"c)参照)
ivw	int	ivw[lvwlen]	作業領域
icon	int	出力	ivwlen = $\lceil \log_2 n \rceil + 10$ コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	係数行列が既約優対角でない.	処理を打ち切る.
30000	次のいずれかであった. $n < 1$ $isw \neq 1, 2$ $ind \neq 0, 1$	処理を打ち切る.

3. 使用上の注意

a) sbd, d, spd 及び b

配列 sbd の後半部分 sbd[n], sbd[n+1],..., sbd[2n-1] は作業領域として利用される. 同様に配列 d, spd, 及び b の後半部分も作業領域として使われる.

本関数で, isw = 1 として演算を行うと, 配列 d, sbd, spd には以下のような値が格納される.

$$sbd[i-1] = e_i / d_i, i = 2, \dots, n, \quad d[i-1] = 1 / d_i, i = 1, \dots, n, \quad spd[i-1] = f_i / d_i, i = 1, \dots, n-1.$$

b) isw

同一の係数行列を持つ 3 項方程式をいくつか続けて解く場合, 2 回目以降 isw=2 として解くと, 係数行列の消去過程を省略するので, 計算時間が少なくなる.

c) ind

もし, 係数行列が既約優対角であることがあらかじめ分かっている場合には, ind = 1 として解くと既約優対角性の判定を行わないので, 計算時間が少なくなる. しかし, 既約優対角でない係数行列に対して ind = 1 と指定すると解の精度は保証されない.

d) 本関数の使用方法

本関数は, ベクトル計算機向きのアルゴリズムである. サイクリック・リダクション法を採用しており, ベクトル計算機上での処理速度については, 次のことが言える.

- ・ ガウス消去法による関数 c_dltx に比べ, 著しく高速である.
- ・ 処理時間は, n の増加に対してほぼ一次関数的に増加する.
- ・ 優対角性が強い問題ほど高速である.
- ・ 精度の面では, 既約優対角行列に対しては, 関数 c_dltx と同等である.

4. 使用例

n 次元 3 項方程式を解く.

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

```
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 100

MAIN__()
{
    int ierr, icon;
    int n, i, isw, ind, ivw[20];
    double eps;
    double sbd[2*NMAX], d[2*NMAX], spd[2*NMAX], b[2*NMAX], x[NMAX];

    /* initialize matrix and vector */
    n = NMAX;
    for (i=0;i<n;i++) {
        sbd[i] = -1;
        spd[i] = -1;
        d[i] = 10;
    }
    sbd[0] = 0;
    spd[n-1] = 0;
    for (i=0;i<n;i++)
        x[i] = i+1;
    /* initialize constant vector b=a*x */
    b[0] = d[0]*x[0] + spd[0]*x[1];
    for (i=1;i<n-1;i++) {
        b[i] = sbd[i]*x[i-1] + d[i]*x[i] + spd[i]*x[i+1];
    }
    b[n-1] = sbd[n-1]*x[n-2] + d[n-1]*x[n-1];
    isw = 1;
    ind = 0;
    /* solve system of equations */
    ierr = c_dvltx(sbd, d, spd, n, b, isw, ind, ivw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dvltx failed with icon = %d\n", icon);
        exit(1);
    }
    /* check solution vector */
    eps = 1e-6;
    for (i=0;i<n;i++)
        if (fabs((x[i]-b[i])/b[i]) > eps) {
            printf("WARNING: result inaccurate\n");
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書”の VLTX の項目及び[104]を参照のこと。

c_dvltx1

定数型実3項行列の連立1次方程式(ディリクレ型, サイクリック・リダクション法)

```
ierr = c_dvltx1(d, sd, n, b, isw, vw, ivw,
                &icon);
```

1. 機能

実3項方程式,

$$\mathbf{Ax} = \mathbf{b} \quad (1)$$

をサイクリック・リダクション法で解く。ただし, $\mathbf{A}$ は(2)のような $n \times n$ の既約優対角性を満たす実3項行列である。

$$\mathbf{A} = \begin{bmatrix} d & e & & 0 \\ e & d & e & \\ & e & \cdots & \\ & & \cdots & e \\ 0 & & & e & d \end{bmatrix} \quad (2)$$

$$d \neq 0, |d| \geq 2|e|$$

(1)において, $\mathbf{b}$ は n 次元の実定数ベクトル, $\mathbf{x}$ は n 次元の解ベクトルである。 $n \geq 1$ であること。

本関数は, 係数行列を(2)に限定することにより, 一般の実3項行列を扱う関数 c_dvltx に対して高速化を図ったものである。

2. 引数

呼出し形式:

```
ierr = c_dvltx1(d, sd, n, b, isw, vw, ivw, &icon);
```

引数の説明:

d	double	入力	対角要素 d .
sd	double	入力	非対角要素 e .
n	int	入力	係数行列 $\mathbf{A}$ の次数 n .
b	double b[2n]	入力 出力	定数ベクトル $\mathbf{b}$. $b[i-1] = b_i, i=1, \dots, n$ ように格納する. 解ベクトル $\mathbf{x}$. $b[i-1] = x_i, i=1, \dots, n$ のように格納される. (“使用上の注意”b)参照)
isw	int	入力	制御情報. 同一の係数行列を持つ複数組の方程式を解くとき, 次のとおり, 指定する. $isw=1$ のとき, 1組目の方程式を解く. $isw=2$ のとき, 2組目以降の方程式を解く. ただし, このとき $\mathbf{b}$ の値だけを新しい定数ベクトル $\mathbf{b}$ の値に変え, それ以外の引数はそのまま使う. (“使用上の注意”c)参照)
vw	double vw[Vwlen]	作業領域	$Vwlen = 2(\lceil \log_2 n \rceil + 1)$.
ivw	int ivw[Ivwlen]	作業領域	$Ivwlen = 2(\lceil \log_2 n \rceil + 1) + 10$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	係数行列が既約優待角でない.	処理を打ち切る.
30000	次のいずれかであった: • $n < 1$ • $isw \neq 1, 2$	処理を打ち切る.

3. 使用上の注意

a) 係数行列 A

係数行列(2)は, 簡単なディリクレ境界値問題を離散化するときに現れる.

b) b

$b[n]$, $b[n+1]$, ..., $b[2n-1]$ は本関数内で作業領域として使用される.

c) isw

同一の係数行列を持つ 3 項方程式をいくつか続けて解く場合は, 2 回目以降 $isw=2$ として解くと, 係数行列の消去過程を省略するので計算時間が少なくなる.

d) 本関数の使用方法

本関数は, ベクトル計算機向きのアルゴリズムであるサイクリック・リダクション法を採用しており, ベクトル計算機上での処理速度については, 次のことが言える.

- ガウス消去法による関数 c_dltx 又は c_dlstx に比べ著しく高速である.
- 処理時間は, n の増加に対してほぼ一次関数的に増加する..
- 優対角性が強い問題ほど高速である.
- 精度の面では, 既約優対角行列に対しては関数 c_dltx 又は c_dlstx と同等である

4. 使用例

定数型実 3 項行列の連立 1 次方程式を解く.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 100

MAIN__()
{
    int ierr, icon;
    int n, i, isw, ivw[50];
    double eps;
    double d, sd, b[2*NMAX], x[NMAX], vw[30];

    /* initialize matrix and vector */
    n = NMAX;
    d = 10;
    sd = -1;
    for (i=0; i<n; i++)
        x[i] = i+1;
    /* initialize constant vector b=a*x */
    b[0] = d*x[0] + sd*x[1];
    for (i=1; i<n-1; i++) {
        b[i] = sd*x[i-1] + d*x[i] + sd*x[i+1];
    }
    b[n-1] = sd*x[n-2] + d*x[n-1];
    isw = 1;
    /* solve system of equations */
    ierr = c_dvltx1(d, sd, n, b, isw, vw, ivw, &icon);
    if (icon != 0) {
```

```
    printf("ERROR: c_dvltx1 failed with icon = %d\n", icon);
    exit(1);
}
/* check solution vector */
eps = 1e-6;
for (i=0;i<n;i++)
    if (fabs((x[i]-b[i])/b[i]) > eps) {
        printf("WARNING: result inaccurate\n");
        exit(1);
    }
printf("Result OK\n");
return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書”の VLTX1 の項目を参照のこと。

c_dvltx2

定数型実3項行列の連立1次方程式(ノイマン型, サイクリック・リダクション法)

```
ierr = c_dvltx2(d, sd, n, b, isw, ind, vw,
                ivw, &icon);
```

1. 機能

実3項方程式,

$$\mathbf{Ax} = \mathbf{b} \quad (1)$$

をサイクリック・リダクション法で解く。ただし、 $\mathbf{A}$ は(2), (3)若しくは(4)のような $n \times n$ の既約優対角性を満たす実3項行列である。

$$\begin{bmatrix} d & 2e & & 0 \\ e & d & e & \\ & e & \cdots & \\ & & \cdots & e \\ 0 & & & e & d \end{bmatrix}, \quad d \neq 0, |d| \geq 2|e| \quad (2)$$

$$\begin{bmatrix} d & e & & 0 \\ e & d & e & \\ & e & \cdots & \\ & & \cdots & e \\ 0 & & & 2e & d \end{bmatrix}, \quad d \neq 0, |d| \geq 2|e| \quad (3)$$

$$\begin{bmatrix} d & 2e & & 0 \\ e & d & e & \\ & e & \cdots & \\ & & \cdots & e \\ 0 & & & 2e & d \end{bmatrix}, \quad d \neq 0, |d| > 2|e| \quad (4)$$

(1)において $\mathbf{b}$ は n 次元の実定数ベクトル, $\mathbf{x}$ は n 次元の解ベクトルである。 $n \geq 1$ であること。

本関数は、係数行列を上記に限定することにより、一般の実3項行列を扱う関数 c_dvltx に対して高速化を図ったものである。

2. 引数

呼出し形式:

```
ierr = c_dvltx2(d, sd, n, b, isw, ind, vw, ivw, &icon);
```

引数の説明:

d	double	入力	対角要素 d .
sd	double	入力	非対角要素 e .
n	int	入力	係数行列 $\mathbf{A}$ の次数 n .
b	double b[Blen]	入力	定数ベクトル $\mathbf{b}$. $b[i-1] = b_i, i=1, \dots, n$ ように格納する. $Blen = 2n + \lceil \log_2 n \rceil$.
		出力	解ベクトル $\mathbf{x}$. $b[i-1] = x_i, i=1, \dots, n$ のように格納される. ("使用上の注意"b)参照)
isw	int	入力	制御情報. 同一の係数行列を持つ複数組の方程式を解くとき,

			次のとおり指定する. isw=1 のとき, 1 組目の方程式を解く. isw=2 のとき, 2 組目以降の方程式を解く. ただし, このとき b の値だけを新しい定数ベクトル b の値に変え, それ以外の引数はそのまま使う. (“使用上の注意”c)参照)
ind	int	入力	係数行列の型を指定する制御情報. ind = 1 のとき(2), ind = 2 のとき(3), ind = 3 のとき(4).
vw	double vw[Vwlen]	作業領域	$Vwlen = 2(\lceil \log_2 n \rceil + 1)$.
ivw	int ivw[Ivwlen]	作業領域	$Ivwlen = 2(\lceil \log_2 n \rceil + 1) + 10$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	係数行列が既約優待角でない.	処理を打ち切る.
30000	次のいずれかであった. • $n < 1$ • $isw \neq 1, 2$ • $ind \neq 1, 2, 3$	処理を打ち切る.

3. 使用上の注意

a) 係数行列 **A**

係数行列の(2)から(4)は, 簡単なノイマン境界値問題を離散化するときに見れる.

b) **b**

$b[n], b[n+1], \dots, b[Blen-1]$ は本関数内で作業領域として使用される.

c) **isw**

同一の係数行列を持つ 3 項方程式をいくつか続けて解く場合は, 2 回目以降 isw=2 として解くと, 係数行列の消去過程を省略するので計算時間が少なくなる.

d) 本関数の使用方法

本関数は, ベクトル計算機向きのアルゴリズムであるサイクリック・リダクション法を採用しており, ベクトル計算機上での処理速度については, 次のことが言える.

- ガウス消去法による関数 c_dltx に比べ著しく高速である.
- 処理時間は, n の増加に対してほぼ一次関数的に増加する.
- 優対角性が強い問題ほど高速である.
- 精度の面では, 既約優対角行列に対しては関数 c_dltx と同等である.

4. 使用例

定数型実 3 項行列の連立 1 次方程式を解く. ind には 3 を設定する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */
```

```
#define NMAX 100

MAIN__()
{
    int ierr, icon;
    int n, i, isw, ind, ivw[30];
    double eps;
    double d, sd, b[2*NMAX+10], x[NMAX], vw[20];

    /* initialize matrix and vector */
    n = NMAX;
    d = 10;
    sd = -1;
    for (i=0;i<n;i++)
        x[i] = i+1;
    /* initialize constant vector b=a*x */
    ind = 3;
    b[0] = d*x[0] + 2*sd*x[1];
    for (i=1;i<n-1;i++) {
        b[i] = sd*x[i-1] + d*x[i] + sd*x[i+1];
    }
    b[n-1] = 2*sd*x[n-2] + d*x[n-1];
    isw = 1;
    /* solve system of equations */
    ierr = c_dvltx2(d, sd, n, b, isw, ind, vw, ivw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dvltx2 failed with icon = %d\n", icon);
        exit(1);
    }
    /* check solution vector */
    eps = 1e-6;
    for (i=0;i<n;i++)
        if (fabs((x[i]-b[i])/b[i]) > eps) {
            printf("WARNING: result inaccurate\n");
            printf("%12.5e %12.5e\n", x[i], b[i]);
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書”の VLTX2 の項目を参照のこと。

c_dvltx3

定数型実3項行列の連立1次方程式(周期型, サイクリック・リダクション法)

```
ierr = c_dvltx3(d, sd, n, b, isw, vw, ivw,
                &icon);
```

1. 機能

実3項方程式

$$\mathbf{Ax} = \mathbf{b} \quad (1)$$

をサイクリック・リダクション法で解く. ただし, $\mathbf{A}$ は (2) のような $n \times n$ の既約優対角性を満たす擬似実3項行列である.

$$\mathbf{A} = \begin{bmatrix} d & e & & & e \\ e & d & e & & 0 \\ e & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & \cdots & e & \\ e & & & e & d \end{bmatrix}, \quad d \neq 0, |d| > 2|e| \quad (2)$$

(1)において, $\mathbf{b}$ は n 次元の実定数ベクトル, $\mathbf{x}$ は n 次元の解ベクトルである. $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dvltx3(d, sd, n, b, isw, vw, ivw, &icon);
```

引数の説明:

d	double	入力	対角要素 d .
sd	double	入力	非対角要素 e .
n	int	入力	係数行列 $\mathbf{A}$ の次数 n .
b	double b[Blen]	入力	定数ベクトル $\mathbf{b}$. $\mathbf{b}[\mathbf{i}-1] = b_i, i=1, \dots, n$ のように格納する. $Blen = 2n + \lceil \log_2 n \rceil$.
		出力	解ベクトル $\mathbf{x}$. $\mathbf{b}[\mathbf{i}-1] = x_i, i=1, \dots, n$ のように格納される. ("使用上の注意"b)参照)
isw	int	入力	制御情報. 同一の係数行列を持つ複数組の方程式を解くとき, 次のとおり指定する. $isw=1$ のとき, 1 組目の方程式を解く. $isw=2$ のとき, 2 組目以降の方程式を解く. ただし, このとき $\mathbf{b}$ の値だけを新しい定数ベクトル $\mathbf{b}$ の値に 変え, それ以外の引数はそのまま使う. ("使用上の注意"c)参照)
vw	double vw[Vwlen]	作業領域	$Vwlen = 3(\lceil \log_2 n \rceil + 1)$.
ivw	int ivw[Ivwlen]	作業領域	$Ivwlen = 4(\lceil \log_2 n \rceil + 1) + 10$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	係数行列が既約優対角でない.	処理を打ち切る.
30000	次のいずれかであった: • $n < 1$ • $isw \neq 1, 2$	処理を打ち切る.

3. 使用上の注意

a) 係数行列 A

係数行列(2)は, 簡単な周期的境界値問題を離散化するときに見れる.

b) b

$b[n], b[n+1], \dots, b[Bl_n-1]$ は本関数内で作業領域として使用される.

c) isw

同一の係数行列を持つ 3 項方程式をいくつか続けて解く場合は, 2 回目以降 $isw=2$ として解くと, 係数行列の消去過程を省略するので計算時間が少なくなる.

d) 本関数の使用方法

本関数は, ベクトル計算機向きのアルゴリズムであるサイクリック・リダクション法を採用しており, ベクトル機上での処理速度については, 次のことが言える.

- ガウス消去法に比べ著しく高速である.
- 処理時間は, n の増加に対してほぼ一次関数的に増加する.
- 優対角性が強い問題ほど高速である.
- 精度の面では, ガウス消去法と同じである.

4. 使用例

定数型実 3 項行列の連立 1 次方程式を解く.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 100

MAIN__()
{
    int ierr, icon;
    int n, i, isw, ivw[50];
    double eps;
    double d, sd, b[2*NMAX+10], x[NMAX], vw[30];

    /* initialize matrix and vector */
    n = NMAX;
    d = 10;
    sd = -1;
    for (i=0; i<n; i++)
        x[i] = i+1;
    /* initialize constant vector b=a*x */
    b[0] = d*x[0] + sd*x[1] + sd*x[n-1];
    for (i=1; i<n-1; i++) {
        b[i] = sd*x[i-1] + d*x[i] + sd*x[i+1];
    }
    b[n-1] = sd*x[0] + sd*x[n-2] + d*x[n-1];
    isw = 1;
    /* solve system of equations */
    ierr = c_dvltx3(d, sd, n, b, isw, vw, ivw, &icon);
    if (icon != 0) {
```



```
        printf("ERROR: c_dvltx3 failed with icon = %d\n", icon);
        exit(1);
    }
    /* check solution vector */
    eps = 1e-6;
    for (i=0;i<n;i++)
        if (fabs((x[i]-b[i])/b[i]) > eps) {
            printf("WARNING: result inaccurate\n");
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書”の VLTX3 の項目を参照のこと。

c_dvluiv

LU 分解された実行列の逆行列

ierr = c_dvluiv(fa, k, n, ip, ai, &icon);

1. 機能

LU 分解された $n \times n$ の実行列 **A** の逆行列 $\mathbf{A}^{-1}$ を求める.

$$\mathbf{A}^{-1} = \mathbf{U}^{-1}\mathbf{L}^{-1}\mathbf{P}$$

ただし, **L**, **U** はそれぞれ $n \times n$ の下三角行列と単位上三角行列であり, **P** は LU 分解におけるピボットティングによる行の入換えを示す置換行列である. また, $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dvluiv((double*)fa, k, n, ip, (double*)ai, &icon);
```

引数の説明:

fa	double fa[n][k]	入力	行列 L と行列 U . (“使用上の注意” a)参照). c_dvalu の出力配列をそのまま使う.
k	int	入力	配列 fa, ai の整合寸法 ($\geq n$).
n	int	入力	行列 L , U の次数 n .
ip	int ip[n]	入力	ピボットティングによる行の入換えの履歴を示すトランスポジションベクトル. (“使用上の注意” b)参照).
ai	double* ai[n][k]	出力	逆行列 $\mathbf{A}^{-1}$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	実行列が非正則であった.	処理を打ち切る.
30000	次のいずれかであった. $k < n$ $n < 1$ - ip に誤りがあった.	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

本関数は, LU 分解された実行列を入力して, 分解された元の実行列の逆行列を計算する. 分解は関数 c_dvalu により行い, 続けて本関数を呼び出すことにより逆行列を求めることができる.

連立 1 次方程式を解く場合には, 関数 c_dvlax を用いること. 逆行列を求めて解くことは, c_dvlax を用いるときよりも演算回数が多くなるので避けた方がよい. 本関数は逆行列が必要なときだけ使用すること.

b) ip について

トランスポジションベクトルとは、部分ピボットリングを行う LU 分解

$$\mathbf{PA} = \mathbf{LU}$$

における置換行列 $\mathbf{P}$ に相当する。 “c_dvalu” の “使用上の注意” を参照すること。

4. 使用例

行列 $\mathbf{A}$ とベクトル $\mathbf{x}$ の積 $\mathbf{b}$ を求める。 行列 $\mathbf{A}$ は LU 分解した後、逆行列 $\mathbf{A}^{-1}$ を求める。 $\mathbf{A}^{-1}\mathbf{b}$ を計算して $\mathbf{x}$ と比較することにより、 $\mathbf{A}^{-1}$ が正しく求まっていることを確認する。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 100

MAIN__()
{
    int ierr, icon;
    int n, i, j, k, is;
    double epsz, eps;
    double a[NMAX][NMAX], ai[NMAX][NMAX];
    double b[NMAX], x[NMAX], y[NMAX], vw[NMAX];
    int ip[NMAX];

    /* initialize matrix and vector */
    n = NMAX;
    for (i=0; i<n; i++)
        for (j=i; j<n; j++) {
            a[i][j] = n-j;
            a[j][i] = n-j;
        }
    for (i=0; i<n; i++)
        x[i] = i+1;
    k = NMAX;
    /* initialize constant vector b = a*x */
    ierr = c_dmav((double*)a, k, n, n, x, b, &icon);
    epsz = 1e-6;
    /* perform LU decomposition */
    ierr = c_dvalu((double*)a, k, n, epsz, ip, &is, vw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dvalu failed with icon = %d\n", icon);
        exit(1);
    }
    /* find matrix inverse from LU factors */
    ierr = c_dvluiv((double*)a, k, n, ip, (double*)ai, &icon);
    if (icon != 0) {
        printf("ERROR: c_dvluiv failed with icon = %d\n", icon);
        exit(1);
    }
    /* calculate y = ai*b */
    ierr = c_dmav((double*)ai, k, n, n, b, y, &icon);
    /* compare x and y */
    eps = 1e-6;
    for (i=0; i<n; i++)
        if (fabs((x[i]-y[i])/y[i]) > eps) {
            printf("WARNING: result inaccurate\n");
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については “FUJITSU SSL II 拡張機能使用手引書” の VLUIV の項目を参照のこと。

c_dvmbv

実バンド行列と実ベクトルの積

<code>ierr = c_dvmbv(a, n, nh1, nh2, x, y, &icon);</code>

1. 機能

$n \times n$ 下バンド幅 h_1 , 上バンド幅 h_2 の実バンド行列 **A** と実ベクトル **x** の積 **y** を求める.

$$\mathbf{y} = \mathbf{Ax} \quad (1)$$

ここで, **x** と **y** は n 次元ベクトルである. $0 \leq h_1 < n, 0 \leq h_2 < n$ であること.

2. 引数

呼出し形式:

```
ierr = c_dvmbv(a, n, nh1, nh2, x, y, &icon);
```

引数の説明:

a	double a[Alen]	入力	行列 A . バンド行列用圧縮格納法. 格納法の詳細については“1.3.4 バンド行列の格納方法”を参照のこと. $Alen = (2h_1 + h_2 + 1)n$. (“使用上の注意”参照)
n	int	入力	行列 A の次数 n .
nh1	int	入力	下バンド幅 h_1 .
nh2	int	入力	上バンド幅 h_2 .
x	double x[n]	入力	ベクトル x .
y	double y[n]	出力	行列 A とベクトル x の積 y .
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	次のいずれかであった. $n = 0$ $nh1 < 0$ 又は $nh1 \geq n$ $nh2 < 0$ 又は $nh2 \geq n$	処理を打ち切る.

3. 使用上の注意

a) 引数 n

本関数は通常(1)の計算を主とするが, $n = -n$ として, 任意のベクトル **y'** を引数 **y** に与えると(2)なる計算ができる. 具体的には, 連立 1 次方程式の残差ベクトルの計算等を行う場合に利用できる.

$$\mathbf{y} = \mathbf{y}' - \mathbf{Ax} \quad (2)$$

4. 使用例

実バンド行列と実ベクトルの積を求める.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define min(i,j) (i<j) ? i : j
```

```

#define max(i,j) (i>j) ? i : j

#define NMAX 100
#define H1MAX 2
#define H2MAX 2

MAIN__()
{
    int ierr, icon;
    int n, nh1, nh2, i, j, jmin, jmax;
    double eps, sum;
    double a[(2*H1MAX+H2MAX+1)*NMAX], x[NMAX], y[NMAX], yy[NMAX];

    /* initialize matrix */
    n = NMAX;
    nh1 = H1MAX;
    nh2 = H2MAX;
    for (i=0;i<n*(2*nh1+nh2+1);i++)
        a[i] = 0;
    for (i=0;i<n;i++) {
        jmin = max(i-nh1,0);
        jmax = min(i+nh2,n-1);
        for (j=jmin;j<=jmax;j++)
            a[i*(2*nh1+1+nh2)+j-i+nh1] = n-fabs(j-i);
    }
    for (i=0;i<n;i++) {
        x[i] = i+1;
    }
    /* multiply directly for checking: yy = a*x */
    for (i=0;i<n;i++) {
        jmin = max(i-nh1,0);
        jmax = min(i+nh2,n-1);
        sum = 0;
        for (j=jmin;j<=jmax;j++)
            sum = sum + a[i*(2*nh1+1+nh2)+j-i+nh1]*x[j];
        yy[i] = sum;
    }
    /* perform matrix vector multiply using c_dvmbv */
    ierr = c_dvmbv(a, n, nh1, nh2, x, y, &icon);
    if (icon != 0) {
        printf("ERROR: c_dvmbv failed with icon = %d\n", icon);
        exit(1);
    }
    /* check result */
    eps = 1e-6;
    for (i=0;i<n;i++)
        if (fabs((y[i]-yy[i])/y[i]) > eps) {
            printf("WARNING: result inaccurate\n");
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}

```

5. 手法概要

$n \times n$, 下バンド幅 h_1 , 上バンド幅 h_2 のバンド行列 $\mathbf{A} = (a_{ij})$ と n 次元のベクトル $\mathbf{x} = (x_j)$ の積 $\mathbf{y} = (y_i)$ の要素を次の式により計算する.

$$y_i = \sum_{j=1}^n a_{ij} x_j, \quad i = 1, \dots, n$$

ただし, 本関数では, バンド行列であることを考慮して, 次の式により計算する.

$$y_i = \sum_{j=\max(1, i-h_1)}^{\min(i+h_2, n)} a_{ij} x_j, \quad i = 1, \dots, n$$

c_dvmcf2

1次元, 多重, 多次元離散型複素フーリエ変換 (複素配列格納, 混合基底)

ierr = c_dvmcf2(z, n, m, isn, &icon);

1. 機能

1次元, 多重, 多次元の離散型複素フーリエ変換を, 複素配列を入出力として行なう.

各次元に対してフーリエ変換を行なうかどうか, 正変換か逆変換かを指定することができる.

各次元の次数は任意の数が可能だが, 2, 3, 5 の因子に分解される場合が高速である.

多次元フーリエ変換

m 次元データ $\{x_{j1j2...jm}\}$ を入力し, 式(1)で定義する変換を行い, $\{\alpha_{k1k2...km}\}$ を求める.

$$\alpha_{k1k2...km} = \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \dots \sum_{jm=0}^{nm-1} x_{j1j2...jm} \cdot \omega_{n1}^{-j1k1r1} \omega_{n2}^{-j2k2r2} \dots \omega_{nm}^{-jmkmrnm} \quad (1)$$

$$\begin{array}{ll} k1 = 0, 1, 2, \dots, n1-1 & \omega_{n1} = \exp(2\pi i / n1) \\ k2 = 0, 1, 2, \dots, n2-1 & \text{かつ} \quad \omega_{n2} = \exp(2\pi i / n2) \\ \dots & \dots \\ km = 0, 1, 2, \dots, nm-1 & \omega_{nm} = \exp(2\pi i / nm) \end{array}$$

ここで, $n1, n2, \dots, nm$ は各次元の大きさである.

$ri = 1$ のとき正変換, $ri = -1$ のとき逆変換になる.

例えば, $r = (1, 1, 1)$ のときは,

$$\alpha_{k1k2k3} = \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \sum_{j3=0}^{n3-1} x_{j1j2j3} \cdot \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \omega_{n3}^{-j3k3}$$

という 3 次元正変換になる.

多重変換

$ri=0$ のとき $\sum_{ji=0}^{ni-1}$ の和をとらず, 式 (1) の x の対応する添字 ji を ki とする.

例えば, 1次元多重変換では 1つの和のみになる. 3次元問題の 2次元目の変換のときは, $r = (0, 1, 0)$ と指定することで

$$\alpha_{k1k2k3} = \sum_{j2=0}^{n2-1} x_{k1j2k3} \cdot \omega_{n2}^{-j2k2}$$

が計算される.

2. 引数

呼出し形式:

ierr = c_dvmcf2((dcomplex*)z, n, m, isn, &icon);

引数の説明:

z	dcomplex z[nm]...[n2][n1]	入力	複素数データ $\{x_{j1j2...jm}\}$ を x[jm]...[j2][j1], $jm=0, \dots, n[m-1]-1, \dots, j2=0, \dots, n[1]-1, j1=0, \dots, n[0]-1$, に格納する.

		出力	複素数データ $\{\alpha_{k1k2...km}\}$ が $\times[km]...[k2][k1]$, $km=0, \dots, n[m-1]-1, \dots, k2=0, \dots, n[1]-1, k1=0, \dots, n[0]-1$, に格納される.
n	int n[m]	入力	$n[i-1]$ に i 次元目の大きさを格納する.
m	int	入力	多次元フーリエ変換の次元数 m .
isn	int isn[m]	入力	$isn[i-1]$ に i 次元目のフーリエ変換の方向 ri を示す. 1 正変換. 0 変換なし. -1 逆変換.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	$m \leq 0$ であった.	処理を打ち切る.
30002	$isn[i] < -1$ 又は $isn[i] > 1$ であった.	
30003	$n[i] < 1$ が指定された.	
30004	$isn[i]$ がすべて 0 であった.	

3. 使用上の注意

a) 一般的なフーリエ変換の定義

多次元離散型複素フーリエ変換及び、逆変換は一般的に式(2), (3)で定義される.

$$\alpha_{k1k2...km} = \frac{1}{n1n2...nm} \times \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \dots \sum_{jm=0}^{nm-1} x_{j1j2...jm} \cdot \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \dots \omega_{nm}^{-jmkm} \quad (2)$$

$$x_{j1j2...jm} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \dots \sum_{km=0}^{nm-1} \alpha_{k1k2...km} \omega_{n1}^{j1k1} \omega_{n2}^{j2k2} \dots \omega_{nm}^{jmkm} \quad (3)$$

ここで,

$$\begin{array}{lll} j1 = 0, 1, 2, \dots, n1-1 & k1 = 0, 1, 2, \dots, n1-1 & \omega_{n1} = \exp(2\pi i / n1) \\ j2 = 0, 1, 2, \dots, n2-1 & \text{かつ} & k2 = 0, 1, 2, \dots, n2-1 & \text{かつ} & \omega_{n2} = \exp(2\pi i / n2) \\ \dots & \dots & \dots & \dots & \dots \\ jm = 0, 1, 2, \dots, nm-1 & km = 0, 1, 2, \dots, nm-1 & \omega_{nm} = \exp(2\pi i / nm) \end{array}$$

本関数では、式 (2), (3) の左辺に対応して $\{n1 \ n2 \ \dots \ nm \ \alpha_{k1k2...km}\}$ 又は $\{x_{j1j2...jm}\}$ を求める. したがって、結果の正規化は必要に応じて行うこと.

b) スタックサイズについて

本関数では、内部で使用する作業域をスタックに確保する. このため、スタックが不足すると異常終了する可能性がある. 本関数がスタックに必要なサイズを以下に示す.

- ni が 2, 3, 5 の巾乗の積で表せるとき
 $16 \times \max\{ni \mid i = 1, \dots, m, \text{かつ } isn[i-1] \neq 0\}$ バイト
- ni の中に 2, 3, 5 の巾乗の積で表せない数があるとき
 $80 \times \max\{ni \mid i = 1, \dots, m, \text{かつ } isn[i-1] \neq 0\}$ バイト

この他に、固定サイズの作業領域や、ユーザープログラムで必要とするスタック領域もあるので、limit コマンド又は ulimit コマンドなどで十分な大きさのスタックサイズを指定することを勧める.

4. 使用例

1次元FFTを計算する.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 100000
#define NDIM 1
#define max(a,b) ((a) > (b) ? (a) : (b))

int MAIN__(void)
{
    int nval[6] = { 16199,16200,16201,16383,16384,16385 };

    dcomplex z[NMAX], tmp;
    double   pi, error, theta;
    int      m, n[NDIM], isn[NDIM], icon;
    int      n1, in, i, k, l;

    pi = 4.0 * atan(1.0);

    for (in=0; in<6; in++) {
        n1 = nval[in];
        n[0] = n1;
        l = 79;

        for (i=0; i<n1; i++) {
            z[i].re = 0.0;
            z[i].im = 0.0;
        }

        z[l].re = 1.0;
        z[l].im = 0.0;
        isn[0] = 1;
        m = 1;

        c_dvmcf2(z, n, m, isn, &icon);

        if (icon != 0) {
            printf("icon = %d\n", icon);
        }

        error = 0.0;
        for (k=0; k<n1; k++) {
            theta = pi*2*l*k/(double)n1;
            tmp.re = fabs(z[k].re-cos(theta));
            tmp.im = fabs(z[k].im+sin(theta));
            error = max(error, tmp.re+tmp.im);
        }

        printf("n = %d, error = %10.3e\n", n1, error);
    }
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書 II”のVMCF2の項目を参照のこと.

c_dvmcft

1次元, 多重, 多次元離散型複素フーリエ変換 (実虚別配列, 混合基底)

```
ierr = c_dvmcft(xr, xi, n, m, isn, w, &iw,
                &icon);
```

1. 機能

1次元, 多重, 多次元の離散型複素フーリエ変換を行なう.

各次元に対してフーリエ変換を行なうかどうか, 正変換か逆変換かを指定することができる.

各次元の次数は任意の数が可能だが, 2, 3, 5 の因子に分解される場合が高速である.

多次元フーリエ変換

$\{x_{j_1 j_2 \dots j_m}\}$ を入力し, 式 (1) で定義する変換を行い, $\{\alpha_{k_1 k_2 \dots k_m}\}$ を求める.

$$\alpha_{k_1 k_2 \dots k_m} = \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} \dots \sum_{j_m=0}^{n_m-1} x_{j_1 j_2 \dots j_m} \cdot \omega_{n_1}^{-j_1 k_1 r_1} \cdot \omega_{n_2}^{-j_2 k_2 r_2} \cdot \dots \cdot \omega_{n_m}^{-j_m k_m r_m} \quad (1)$$

ここで,

$$\begin{array}{ll} k_1 = 0, 1, 2, \dots, n_1 - 1 & \omega_{n_1} = \exp(2\pi i / n_1) \\ k_2 = 0, 1, 2, \dots, n_2 - 1 & \omega_{n_2} = \exp(2\pi i / n_2) \\ \vdots & \vdots \\ k_m = 0, 1, 2, \dots, n_m - 1 & \omega_{n_m} = \exp(2\pi i / n_m) \end{array} \quad \text{かつ}$$

$r_i = 1$ のとき正変換, $r_i = -1$ のとき逆変換.

$r_i = 0$ のとき $\sum_{j_i=0}^{n_i-1}$ の和はとらず, 式(1)において x の対応する添字 j_i を k_i とする.

$r = (0, 1, 1)$ のときは,

$$\alpha_{k_1 k_2 k_3} = \sum_{j_2=0}^{n_2-1} \sum_{j_3=0}^{n_3-1} x_{k_1 j_2 j_3} \cdot \omega_{n_2}^{-j_2 k_2} \cdot \omega_{n_3}^{-j_3 k_3}$$

となる.

多重変換

多重変換は 1 つの和のみを持つ. 2次元目の変換のときは,

$$\alpha_{k_1 k_2 k_3} = \sum_{j_2=0}^{n_2-1} x_{k_1 j_2 k_3} \cdot \omega_{n_2}^{-j_2 k_2}$$

となる.

2. 引数

呼出し形式:

```
ierr = c_dvmcft(xr, xi, n, m, isn, w, &iw, &icon);
```

引数の説明:

xr	double	入力	$x_{j_1 j_2 \dots j_m}$ の実部. $Xlen = n_1 \cdot n_2 \cdot \dots \cdot n_m$.
	xr[Xlen]	出力	$\alpha_{k_1 k_2 \dots k_m}$ の実部.
xi	double	入力	$x_{j_1 j_2 \dots j_m}$ の虚部. $Xlen = n_1 \cdot n_2 \cdot \dots \cdot n_m$.
	xi[Xlen]	出力	$\alpha_{k_1 k_2 \dots k_m}$ の虚部.
n	int n[m]	入力	n[i-1]に i 次元目の大きさを入力する.
m	int	入力	多次元フーリエ変換の次元数 m.
isn	int isn[m]	入力	isn[i-1]に i 次元目のフーリエ変換の方向 r_i を示す. 1 正変換. 0 変換なし. -1 逆変換.
w	double w[iw]	作業領域	
iw	int	入力	作業域の大きさ. (“使用上の注意” b)参照)
		出力	作業域の大きさが必要な量より小さいとき, 必要な作業域の大きさを返す.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	m ≤ 0 であった.	処理を打ち切る.
30001	作業領域が不足.	
30002	isn[i] < -1 又は isn[i] > 1 であった.	
30003	n[i] < 1 が指定された.	
30004	isn[i] がすべて 0 であった.	

3. 使用上の注意

a) 一般的なフーリエ変換の定義

多次元離散型複素フーリエ変換及び, 逆変換は一般的に式 (2), (3) で定義される.

$$\alpha_{k_1 k_2 \dots k_m} = \frac{1}{n_1 n_2 \dots n_m} \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} \dots \sum_{j_m=0}^{n_m-1} x_{j_1 j_2 \dots j_m} \cdot \omega_{n_1}^{-j_1 k_1} \cdot \omega_{n_2}^{-j_2 k_2} \cdot \dots \cdot \omega_{n_m}^{-j_m k_m} \quad (2)$$

$$x_{k_1 k_2 \dots k_m} = \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} \dots \sum_{j_m=0}^{n_m-1} \alpha_{j_1 j_2 \dots j_m} \cdot \omega_{n_1}^{j_1 k_1} \cdot \omega_{n_2}^{j_2 k_2} \cdot \dots \cdot \omega_{n_m}^{j_m k_m} \quad (3)$$

ここで,

$$\begin{array}{lll} j_1 = 0, 1, 2, \dots, n_1 - 1 & k_1 = 0, 1, 2, \dots, n_1 - 1 & \omega_{n_1} = \exp(2\pi i / n_1) \\ j_2 = 0, 1, 2, \dots, n_2 - 1 & \text{かつ} & k_2 = 0, 1, 2, \dots, n_2 - 1 & \text{かつ} & \omega_{n_2} = \exp(2\pi i / n_2) \\ \vdots & & \vdots & & \vdots \\ j_m = 0, 1, 2, \dots, n_m - 1 & k_m = 0, 1, 2, \dots, n_m - 1 & \omega_{n_m} = \exp(2\pi i / n_m) \end{array}$$

本関数では, 式 (2), (3) の左辺に対応して $\{n_1 \cdot n_2 \cdot \dots \cdot n_m \cdot \alpha_{k_1 k_2 \dots k_m}\}$ 又は $\{x_{j_1 j_2 \dots j_m}\}$ を求める. したがって, 結果の正規化は必要に応じて行うこと.

b) 作業域の大きさ

以下の説明で, 使う記号を定義する.

- $RADIX = \{ n : \text{正の整数で } 2, 3, 5 \text{ の中乗の積の形に表せる数} \}$
- $NORAD = \text{自然数} - RADIX$
- $minrad(n)$ は $n < m$ かつ $m \in RADIX$ なる最小の自然数 m .
- $relfac(n)$ は $n = p \cdot q$ で $p \in RADIX$, $q \in NORAD$ なる最小の自然数 q .
- $NP = n_1 \cdot n_2 \cdot \dots \cdot n_m$

このとき作業域の大きさは以下の手順で決まる.

$\{ i : i=1, 2, \dots, m \text{ かつ } isn[i-1] \neq 0 \}$

- ① $n_i \in RADIX$ のとき, $\max(2n_i)$
- ② $relfac(n_i) = n_i$ のとき, $\max(2NP \cdot minrad(n_i)/n_i + 4minrad(n_i))$
- ③ その他のとき, $\max(2NP \cdot minrad(relfac(n_i))/relfac(n_i) + \max(4minrad(relfac(n_i)), 2n_i))$

大きさがわからないときは, $iw=0$ と設定して関数を呼び出せば必要な作業域の大きさを知ることができる.

4. 使用例

次数 16384 の 1 次元フーリエ変換を計算する. 入力は 101 番目の要素だけを $1+i0$ とし, 他の要素は 0 とする.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define M      1
#define N     16384
#define W     2*N

#define max(i,j) (i>j) ? i : j

MAIN__()
{
    int ierr, icon;
    double xr[N], xi[N], w[W], eps, pi;
    int i, k, n[M], m, isn[M], iw;

    /* generate initial data */
    m = M;
    n[0] = N;
    k = 100;
    for (i=0; i<n[0]; i++) {
        xr[i] = 0;
        xi[i] = 0;
    }
    xr[k] = 1;
    isn[0] = 1;
    iw = W;
    /* perform transform */
    ierr = c_dvmcft(xr, xi, n, m, isn, w, &iw, &icon);
    /* check results */
    if (icon != 0) {
        printf("ERROR: c_dvmcft failed with icon = %d\n", icon);
        exit(1);
    }
    pi = 4*atan(1);
    eps = 1e-6;
    for (i=0; i<n[0]; i++)
        if ((xr[i]-cos(2*pi*i*k/n[0]) > eps) ||
            (xi[i]+sin(2*pi*i*k/n[0]) > eps)) {
            printf("Inaccurate result\n");
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書 II”のVMCFTの項目及び[49], [50]を参照のこと.

c_dvmcst

離散型 cosine 変換

```
ierr = c_dvmcst(x, k, n, m, isw, tab, &icon);
```

1. 機能

1次元, 多重の離散型 cosine 変換を行なう.

周期 2π の偶関数 $x(t)$ の半周期を n 等分し, 両端を含む $n+1$ 個の標本 $\{x_j\}$

$$x_j = x\left(\frac{\pi}{n}j\right), j = 0, 1, \dots, n$$

が与えられたとき, 以下で定義する離散型 cosine 変換を配列の各行に対して行なう.

$$a_k = x_0 + (-1)^k x_n + 2 \sum_{j=1}^{n-1} x_j \cos \frac{\pi}{n} kj, \quad k = 0, 1, \dots, n \quad (1)$$

2. 引数

呼出し形式:

```
ierr = c_dvmcst((double*)x, k, n, m, isw, tab, &icon);
```

引数の説明:

x	double x[m][k]	入力	m 本の $\{x_j\}, j = 0, 1, \dots, n$ を x[i1][i2], i1 = 0, ..., m-1, i2 = 0, ..., n, に格納する.
		出力	m 本の $\{a_k\}, k = 0, 1, \dots, n$ が x[i1][i2], i1 = 0, ..., m-1, i2 = 0, ..., n, に格納される.
k	int	入力	x の 2 次元目の大きさ. $k \geq n+1$.
n	int	入力	区間の分割数 n . n は偶数. (“使用上の注意” a)参照)
m	int	入力	変換の多重度 m .
isw	int	入力	制御情報. 0 tab 設定及び cosine 変換. 1 tab 設定のみ. 2 設定済みの tab を使用して cosine 変換.
tab	double tab[2×n]	作業領域	三角関数表を格納する. (“使用上の注意” b)参照)
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	次のいずれかであった. $n \leq 0$ $k < n+1$ $m \leq 0$ $isw \neq 0, 1, 2$ n が偶数でない.	処理を打ち切る.

3. 使用上の注意

a) n の値について

区間の分割数 n は偶数であれば任意の数が指定可能であるが、2, 3, 5 の因子に分解される場合が高速である。

b) tab の再利用について

同じ n で本関数を繰り返し呼び出す場合、初回に $isw = 0$ 又は 1 で呼び出した後、2 回目以降は $isw = 2$ で呼び出すことにより、tab の作成が省略されて処理が高速化できる。

c) 正規化について

(1) 式で定義された cosine 変換は、それ自身が逆変換になっている。c_dvmcst 関数で二回変換を行うと、入力データが $2n$ 倍されたデータが出力される。

d) スタックサイズについて

本関数では、内部で使用する作業域をスタックに確保している。このため、スタックが不足すると異常終了する可能性がある。本関数がスタックに必要とするサイズは、 $8 \times n$ バイトである。この他に、固定サイズの作業領域や、ユーザプログラムで必要とするスタック領域もあるので、limit コマンド又は ulimit コマンドなどで十分な大きさのスタックサイズを指定することを勧める。

4. 使用例

$n = 1024$, 標本点数 1023 の cosine 変換を 5 本計算する。

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define N 1024
#define K (1024+1)
#define M 5
#define max(a,b) ((a) > (b) ? (a) : (b))

int MAIN__(void)
{
    double x[M][K], tab[N*2];
    double vnrm, error, t1, t2;
    int isw, icon;
    int i, j;

    for (j=0; j<M; j++) {
        for (i=0; i<N+1; i++) {
            x[j][i]=(double)max(i, (N-i)/(j+1));
        }
    }

    /* FORWARD TRANSFORM */
    isw=0;
    c_dvmcst((double*)x, K, N, M, isw, tab, &icon);

    printf("icon = %d\n", icon);

    /* BACKWARD TRANSFORM */
    isw=2;
    c_dvmcst((double*)x, K, N, M, isw, tab, &icon);

    printf("icon = %d\n", icon);

    for (j=0; j<M; j++) {
        error=0.0;
        vnrm =0.0;
        for (i=0; i<N+1; i++) {
            t1=x[j][i]/(double)(N*2);
            t2=t1-(double)(max(i, (N-i)/(j+1)));
            vnrm +=t1*t1;
            error+=t2*t2;
        }
    }
}
```

```
        printf("error = %e\n",sqrt(error/vnrm));  
    }  
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書 II”のVMCSTの項目を参照のこと。

c_dvmggm

行列の積 (実行列)
<pre>ierr = c_dvmggm(a, ka, b, kb, c, kc, m, n, l, &icon);</pre>

1. 機能

$m \times n$ の実行列 **A** と、 $n \times l$ の実行列 **B** の積 **C** を求める.

$$\mathbf{C} = \mathbf{AB}$$

ここで **C** は、 $m \times l$ の実行列である. $m, n, l \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dvmggm((double*)a, ka, (double*)b, kb, (double*)c, kc, m, n, l,
                 &icon);
```

引数の説明:

a	double	入力	行列 A .
	a[m][ka]		
ka	int	入力	配列 a の整合寸法 ($\geq n$).
b	double	入力	行列 B .
	b[n][kb]		
kb	int	入力	配列 b の整合寸法 ($\geq l$).
c	double	出力	行列 C (“使用上の注意” a)参照).
	c[m][kc]		
kc	int	入力	配列 c の整合寸法 ($\geq l$).
m	int	入力	行列 A , C の行数 m .
n	int	入力	行列 A の列数, 及び行列 B の行数 n .
l	int	入力	行列 B , C の列数 l .
icon	int	出力	コンディションコード.下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	次のいずれかであった. • $m < 1$ • $n < 1$ • $l < 1$ • $ka < n$ • $kb < l$ • $kc < l$	処理を打ち切る.

3. 使用上の注意

a) 領域の節約について

領域の節約をするために配列 a 又は、配列 b に行列 C を格納するような使用はできない。必ず、行列 A, B, C は別々の領域を確保すること。

4. 使用例

行列積を計算し、結果を確認する。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 100

MAIN__()
{
    int ierr, icon;
    int n, i, j;
    double eps;
    double a[NMAX][NMAX], b[NMAX][NMAX], c[NMAX][NMAX];

    /* initialize matrices */
    n = NMAX;
    for (i=0;i<n;i++)
        for (j=0;j<n;j++) {
            a[i][j] = j+1;
            b[j][i] = 1.0/(j+1);
        }
    /* matrix matrix multiply */
    ierr = c_dvmggm((double*)a, NMAX, (double*)b, NMAX,
                    (double*)c, NMAX, n, n, n, &icon);
    /* check result */
    eps = 1e-5;
    for (i=0;i<n;i++)
        for (j=0;j<n;j++)
            if (fabs((c[i][j]-n)/n) > eps) {
                printf("WARNING: result inaccurate\n");
                exit(1);
            }
    printf("Result OK\n");
    return(0);
}
```

c_dvmrf2

1次元, 多重, 多次元離散型実フーリエ変換 (混合基底)

ierr = c_dvmrf2(x, n, m, isin, isn, &icon);

1. 機能

1次元, 多重, 多次元の離散型実フーリエ変換を行う。

各次元に対してフーリエ変換を行わないか, 変換を行う場合にその向きを指定することができる。1次元目に関しては変換なしを指定できない。また, 1次元目の大きさは偶数でなければならない。その他の次元の大きさは任意であるが, 2, 3, 5 の積として表される数の場合が高速である。

多重, 多次元の離散型実フーリエ変換の結果には複素共役の関係があるため, 1次元目に関して最初の $n_1/2+1$ 個の複素数要素を格納する。

a) 多次元実フーリエ変換

変換:

m 次元実データ $\{x_{j1j2...jm}\}$ を入力し, (1) で定義する変換を行い, $\{\alpha_{k1k2...km}\}$ を求める。

$$\alpha_{k1k2...km} = \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \dots \sum_{jm=0}^{nm-1} x_{j1j2...jm} \omega_{n1}^{-j1k1r1} \omega_{n2}^{-j2k2r2} \dots \omega_{nm}^{-jmkrmr}, \quad (1)$$

$$\begin{aligned} k1 &= 0, 1, 2, \dots, n1-1 & \omega_{n1} &= \exp(2\pi i / n1) \\ k2 &= 0, 1, 2, \dots, n2-1 & \omega_{n2} &= \exp(2\pi i / n2) \\ &\dots & &\dots \\ km &= 0, 1, 2, \dots, nm-1 & \omega_{nm} &= \exp(2\pi i / nm) \end{aligned} \quad \text{かつ}$$

ここで, $n1, n2, \dots, nm$ は各次元の大きさである。

変換の方向は $ri = 1$ 又は $ri = -1$ を指定できる。

例えば, $r = (1, 1, 1)$ のときの変換は,

$$\alpha_{k1k2k3} = \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \sum_{j3=0}^{n3-1} x_{j1j2j3} \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \omega_{n3}^{-j3k3}$$

という 3 次元離散型実フーリエ正変換になる。

逆変換:

m 次元データ $\{\alpha_{k1k2...km}\}$ を入力し, (2) で定義する変換を行い, $\{x_{j1j2...jm}\}$ を求める。

$$x_{j1j2...jm} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \dots \sum_{km=0}^{nm-1} \alpha_{k1k2...km} \omega_{n1}^{-j1k1r1} \omega_{n2}^{-j2k2r2} \dots \omega_{nm}^{-jmkrmr} \quad (2)$$

$$\begin{aligned} j1 &= 0, 1, 2, \dots, n1-1 & \omega_{n1} &= \exp(2\pi i / n1) \\ j2 &= 0, 1, 2, \dots, n2-1 & \omega_{n2} &= \exp(2\pi i / n2) \\ &\dots & &\dots \\ jm &= 0, 1, 2, \dots, nm-1 & \omega_{nm} &= \exp(2\pi i / nm) \end{aligned} \quad \text{かつ}$$

ここで, $n1, n2, \dots, nm$ は各次元の大きさである。

逆変換では, 変換で指定した方向と逆向きを指定する。 $ri = -1$ 又は $ri = 1$

b) 多重変換

$ri=0$ のとき $\sum_{ji=0}^{ni-1}$ の和はとらない。

このとき、正変換では、式(1)の x の対応する添字 ji を ki とする。逆変換では、式(2)の α の対応する添字 ki を ji とする。

例えば、3次元配列の1次元目の多重変換は $r=(1, 0, 0)$ と指定することで、

$$\alpha_{k1k2k3} = \sum_{j1=0}^{n1-1} x_{j1k2k3} \omega_{n1}^{-j1k1}$$

が計算される。

2. 引数

呼出し形式:

```
ierr = c_dvmrf2((double *)x, n, m, isin, isn, &icon);
```

引数の説明:

x	double x[nm]...[n2][n1+2]	入力	isn=1(実数から複素数への変換)のとき, x[jm]...[j2][j1], jm=0, ..., n[m-1]-1, ..., j2=0, ..., n[1]-1, j1=0, ..., n[0]-1 に実データ { $x_{j1j2...jm}$ } を格納する. isn=-1(複素数から実数への変換)のとき, x[km]...[k2][k1], km=0, ..., n[m-1]-1, ..., k2=0, ..., n[1]-1, k1=0, ..., n[0]+1 に { $\alpha_{k1k2...km}$ } の実数成分と虚数成分を交互に格納する.
		出力	isn=1(実数から複素数への変換)のとき, x[km]...[k2][k1], km=0, ..., n[m-1]-1, ..., k2=0, ..., n[1]-1, k1=0, ..., n[0]+1 に { $\alpha_{k1k2...km}$ } の実数成分と虚数成分が交互に格納される. isn=-1(複素数から実数への変換)のとき, x[jm]...[j2][j1], jm=0, ..., n[m-1]-1, ..., j2=0, ..., n[1]-1, j1=0, ..., n[0]-1 に実データ { $x_{j1j2...jm}$ } が格納される.
n	int n[m]	入力	n[i-1]に i 次元目の大きさ n_i を格納する. $n1$ は、偶数でなければならない.
m	int	入力	多次元フーリエ変換の次元数 m .
isin	int isin[m]	入力	各次元のフーリエ変換の方向 $ri, i=1, 2, \dots, m$ を示す. isin[i-1]=1 のとき $ri=1$. isin[i-1]=0 のとき変換なし. isin[i-1]=-1 のとき $ri=-1$. ただし, isin[0] $\neq 0$.
isn	int	入力	制御情報. isn=1 のとき変換(実数から複素数). isn=-1 のとき逆変換(複素数から実数).
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30001	次のいずれかであった. • $n[i] \leq 0$ • $m \leq 0$.	処理を打ち切る.

コード	意味	処理内容
30016	$\text{isin}[i] \neq -1, 0, 1$ 又は $\text{isin}[0] = 0$.	
30032	$\text{isn} \neq -1, 1$.	
30512	$n[0]$ が奇数である.	

3. 使用上の注意

a) 一般的なフーリエ変換の定義

多次元離散型フーリエ変換及び、逆変換は一般的に(3), (4)で定義される.

$$\alpha_{k1k2\dots km} = \frac{1}{n1n2\dots nm} \times \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \dots \sum_{jm=0}^{nm-1} x_{j1j2\dots jm} \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \dots \omega_{nm}^{-jmkm} \quad (3)$$

$$\begin{aligned} k1 &= 0, 1, 2, \dots, n1-1 & \omega_{n1} &= \exp(2\pi i / n1) \\ k2 &= 0, 1, 2, \dots, n2-1 & \omega_{n2} &= \exp(2\pi i / n2) \\ &\dots & \dots & \\ km &= 0, 1, 2, \dots, nm-1 & \omega_{nm} &= \exp(2\pi i / nm) \end{aligned} \quad \text{かつ}$$

$$x_{j1j2\dots jm} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \dots \sum_{km=0}^{nm-1} \alpha_{k1k2\dots km} \omega_{n1}^{j1k1} \omega_{n2}^{j2k2} \dots \omega_{nm}^{jmkm}, \quad (4)$$

$$\begin{aligned} j1 &= 0, 1, 2, \dots, n1-1 & \omega_{n1} &= \exp(2\pi i / n1) \\ j2 &= 0, 1, 2, \dots, n2-1 & \omega_{n2} &= \exp(2\pi i / n2) \\ &\dots & \dots & \\ jm &= 0, 1, 2, \dots, nm-1 & \omega_{nm} &= \exp(2\pi i / nm) \end{aligned} \quad \text{かつ}$$

本関数では, (3), (4)の左辺に対応して $\{n1 \ n2 \ \dots \ nm \ \alpha_{k1k2\dots km}\}$ 又は $\{x_{j1j2\dots jm}\}$ を求める. ただし $\text{isin}[i-1] = 0$ となる i に対しては, ni は 1 と置き換える. 結果の正規化は必要に応じて行うこと.

b) 複素共役関係

多次元離散型実フーリエ変換の結果には以下の複素共役の関係がある.

$$\begin{aligned} \alpha_{k1k2\dots km} &= \overline{\alpha_{n1-k1 \ n2-k2 \ \dots \ nm-km}} \\ k1 &= 0, 1, 2, \dots, n1/2 \\ k2 &= 0, 1, 2, \dots, n2-1 \\ &\dots \\ km &= 0, 1, 2, \dots, nm-1 \end{aligned}$$

ただし, $ni - ki$ は $ki=0$ と見なす. $\text{isin}[h] = 0$ となる h に対して, 右辺の h 番目の添字は kh のままである. この関係を使って残りの部分を計算することができる.

c) スタックサイズについて

本関数では, 内部で使用する作業域をスタックに確保する. このため, スタックが不足すると異常終了する可能性がある. 本関数がスタックに必要とするサイズを以下に示す.

- ni が 2, 3, 5 の巾乗の積で表せるとき
 $24 \times \max\{ni \mid i = 1, \dots, m, \text{かつ } \text{isn}[i-1] \neq 0\}$ バイト
- ni の中に 2, 3, 5 の巾乗の積で表せない数があるとき
 $80 \times \max\{ni \mid i = 1, \dots, m, \text{かつ } \text{isn}[i-1] \neq 0\}$ バイト

この他に, 固定サイズの作業領域や, ユーザープログラムで必要とするスタック領域もあるので, limit コマンド又は ulimit コマンドなどで十分な大きさのスタックサイズを指定することを勧める.

4. 使用例

2次元実フーリエ変換及び逆変換を計算する.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define N1 1024
#define N2 1024
#define M 2
#define max(a,b) ((a) > (b) ? (a) : (b))

int MAIN__(void)
{
    double x[N2][N1+2], error, tmp;
    int n[M], isn, isin[M], icon;
    int i, j;

    for (i=0; i<N2; i++) {
        for (j=0; j<N1; j++) {
            x[i][j]=(double)(N1*i+j+1);
        }
    }

    n[0] = N1;
    n[1] = N2;
    isin[0] = 1;
    isin[1] = 1;
    isn = 1;

    /* REAL TO COMPLEX TRANSFORM */
    c_dvmrf2((double*)x, n, M, isin, isn, &icon);

    printf("icon = %d\n",icon);

    n[0] = N1;
    n[1] = N2;
    isin[0] = -1;
    isin[1] = -1;
    isn = -1;

    /* COMPLEX TO REAL TRANSFORM */
    c_dvmrf2((double*)x, n, M, isin, isn, &icon);

    printf("icon = %d\n",icon);

    error = 0.0;
    for (i=0; i<N2; i++) {
        for (j=0; j<N1; j++) {
            tmp = fabs(x[i][j]/((double)(N1*N2)-((double)(N1*i+j+1)));
            error = max(error,tmp);
        }
    }

    printf("error = %e\n",error);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書 II”のVMRF2の項目を参照のこと.

c_dvmrft

多重, 多次元離散型実フーリエ変換 (2, 3 及び 5 の混合基底)

ierr = c_dvmrft(x, n, m, isin, isn, w, &icon);

1. 機能

多重, 多次元の離散型実フーリエ変換を行う。

各次元に対してフーリエ変換を行わないか, 変換を行う場合にその向きを指定することができる。変換を行う次元の大きさは 2, 3, 5 の中の積として表される数でなければならない。

1~m-1 次元の大きさのうち少なくとも 1 つは偶数であること。また m 次元目に関しては変換なしを指定することはできない。

多重, 多次元の離散型実フーリエ変換の結果には複素共役の関係があるため, m 番目の次元に関しては最初の $(n_m / 2) + 1$ 個を格納する。

a) 多次元実フーリエ変換

変換:

$\{x_{j_1 j_2 \dots j_m}\}$ を入力し, (1) で定義する変換を行い, $\{n_1 n_2 \dots n_m \alpha_{k_1 k_2 \dots k_m}\}$ を求める。

$$n_1 n_2 \dots n_m \alpha_{k_1 k_2 \dots k_m} = \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} \dots \sum_{j_m=0}^{n_m-1} x_{j_1 j_2 \dots j_m} \omega_{n_1}^{-j_1 k_1 r_1} \omega_{n_2}^{-j_2 k_2 r_2} \dots \omega_{n_m}^{-j_m k_m r_m}, \quad (1)$$

$$k_\ell = 0, \dots, n_\ell - 1,$$

$$\omega_{n_\ell} = \exp(2\pi i / n_\ell),$$

$$\ell = 1, 2, \dots, m.$$

変換方法は $r_\ell = 1$ 又は $r_\ell = -1$ を指定できる。 $r_\ell = 0$ のとき $\sum_{j_\ell=0}^{n_\ell-1}$ の和はとらず, x の対応する添字 j_ℓ を k_ℓ とする。このとき, 式(1)の左辺の n_ℓ は 1 に置き換わる。例えば $r = (0, 1, 1)$ のときの変換は次のようになる。

$$n_2 n_3 \alpha_{k_1 k_2 k_3} = \sum_{j_2=0}^{n_2-1} \sum_{j_3=0}^{n_3-1} x_{k_1 j_2 j_3} \omega_{n_2}^{-j_2 k_2} \omega_{n_3}^{-j_3 k_3}$$

逆変換:

$\{\alpha_{k_1 k_2 \dots k_m}\}$ を入力し, (2) で定義する変換を行い, $\{x_{j_1 j_2 \dots j_m}\}$ を求めます。

$$x_{j_1 j_2 \dots j_m} = \sum_{k_1=0}^{n_1-1} \sum_{k_2=0}^{n_2-1} \dots \sum_{k_m=0}^{n_m-1} \alpha_{k_1 k_2 \dots k_m} \omega_{n_1}^{-j_1 k_1 r_1} \omega_{n_2}^{-j_2 k_2 r_2} \dots \omega_{n_m}^{-j_m k_m r_m} \quad (2)$$

$$j_\ell = 0, \dots, n_\ell - 1,$$

$$\omega_{n_\ell} = \exp(2\pi i / n_\ell),$$

$$\ell = 1, 2, \dots, m.$$

逆変換では, 変換で指定した方向と逆向きを指定する。

変換方法は $r_\ell = 1$ 又は $r_\ell = -1$ を指定できる。 $r_\ell = 0$ のとき $\sum_{k_\ell=0}^{n_\ell-1}$ の和はとらず, x の対応する添字 k_ℓ を j_ℓ とする。

b) 多重変換

多重変換は 1 つの和のみを持つ, 3 次元の変換のときは以下のようなになる。

$$n_3 \alpha_{k_1 k_2 k_3} = \sum_{j_3=0}^{n_3-1} x_{k_1 k_2 j_3} \omega_{n_3}^{-j_3 k_3 r_3}$$

2. 引数

呼出し形式:

```
ierr = c_dvmrft((double *)x, n, m, isin, isn, w, &icon);
```

引数の説明:

x	double x[Nmlen] [n[m-2]]...[n[0]]	入力	isn = 1 (実数から複素数への変換) のとき, 実データ $\{x_{j_1 j_2 \dots j_m}\}$ を格納する. isn = -1 (複素数から実数への変換) のとき, 複素データ $\{\alpha_{k_1 k_2 \dots k_m}\}$ を格納する. $Nmlen = 2 \times (\text{floor}(n_m/2) + 1) = 2(n[m-1]/2 + 1)$. ("使用上の注意" a)参照)
		出力	isn = 1 (実数から複素数への変換) のとき, 複素データ $\{n_1 n_2 \dots n_m \alpha_{k_1 k_2 \dots k_m}\}$ が格納される. isn = -1 (複素数から実数への変換) のとき, 実データ $\{x_{j_1 j_2 \dots j_m}\}$ が格納される. ("使用上の注意" a)参照)
n	int n[m]	入力	i 次元目の大きさ $n_i, i = 1, 2, \dots, m$ を格納する. $n[i-1] = n_i$. n_i は 2, 3, 5 の巾乗の積で表せる数. $n_i (i = 1, \dots, m-1)$ の少なくとも 1 つは, 偶数でなければならない.
m	int	入力	多次元フーリエ変換の次元数 m.
isin	int isin[m]	入力	各次元のフーリエ変換の方向 $r_i, i = 1, 2, \dots, m$ を示す. isin[i-1] = 1 のとき $r_i = 1$. isin[i-1] = 0 のとき変換なし. isin[i-1] = -1 のとき $r_i = -1$. ただし, isin[m-1] $\neq$ 0.
isn	int	入力	制御情報. isn = 1 のとき変換 (実数から複素数). isn = -1 のとき逆変換 (複素数から実数).
w	double w[Wlen]	作業領域	$Wlen = 2(\max(n_1, n_2, \dots, n_m) + n_1 n_2 \dots n_{m-1}(\text{floor}(n_m/2) + 1))$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30001	次のいずれかであった. • $n[i] \leq 0$ • $m < 2$.	処理を打ち切る.
30008	isin[i] $\neq 0$ なる n[i] が 2, 3, 5 の巾の積であらわされる整数でない.	処理を打ち切る.
30016	isin[i] $\neq -1, 0, 1$ 又は isin[m-1] = 0.	処理を打ち切る.
30032	isn $\neq -1, 1$.	処理を打ち切る.
30512	$n[i-1], i = 1, 2, \dots, m$ がすべて奇数である.	処理を打ち切る.

3. 使用上の注意

a) データの格納方法

実データは配列 x に次のように格納する.

$$x[j_m] \dots [j_2] [j_1] = x_{j_1 j_2 \dots j_m}, j_i = 0, 1, \dots, n_i - 1, i = 1, 2, \dots, m$$

複素データは実部, 虚部をそれぞれ次のように格納する.

$$\begin{aligned} x[k_m] \dots [k_2] [k_1] &= \text{Re}(\alpha_{k_1 k_2 \dots k_m}) \text{ 又は } \text{Re}(n_1 n_2 \dots n_m \alpha_{k_1 k_2 \dots k_m}), & k_i &= 0, 1, \dots, n_i - 1, i = 1, 2, \dots, m-1 \\ x[k_m + n[m-1]/2 + 1] \dots [k_2] [k_1] &= \text{Im}(\alpha_{k_1 k_2 \dots k_m}) \text{ 又は } \text{Im}(n_1 n_2 \dots n_m \alpha_{k_1 k_2 \dots k_m}), & k_m &= 0, \dots, \text{floor}(n_m/2) \end{aligned}$$

b) 複素共役関係

多次元離散型実フーリエ変換の結果には以下の複素共役の関係がある.

$$\alpha_{k_1 k_2 \dots k_m} = \overline{\alpha_{n_1 - k_1 \ n_2 - k_2 \ \dots \ n_m - k_m}}$$

$k_i = 0, 1, \dots, n_i - 1, i = 1, 2, \dots, m-1, k_m = 1, 2, \dots, \text{floor}(n_m/2)$. ただし, $k_i = 0$ のとき, $n_i - k_i$ は 0 と見なす.

$\text{isin}[h] = 0$ となる h に対して, 右辺の h 番目の添字は k_h のままである.

c) 一般的なフーリエ変換の定義

多次元離散型フーリエ変換及び, 逆変換は一般的に(3), (4)で定義される.

$$\begin{aligned} \alpha_{k_1 k_2 \dots k_m} &= \frac{1}{n_1 n_2 \dots n_m} \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} \dots \sum_{j_m=0}^{n_m-1} x_{j_1 j_2 \dots j_m} \omega_{n_1}^{-j_1 k_1 r_1} \omega_{n_2}^{-j_2 k_2 r_2} \dots \omega_{n_m}^{-j_m k_m r_m}, \\ k_\ell &= 0, \dots, n_\ell - 1, \\ \omega_{n_\ell} &= \exp(2\pi i / n_\ell), \\ r_\ell &= 1 \text{ 又は } -1, \ell = 1, 2, \dots, m \end{aligned} \quad (3)$$

$$\begin{aligned} x_{j_1 j_2 \dots j_m} &= \sum_{k_1=0}^{n_1-1} \sum_{k_2=0}^{n_2-1} \dots \sum_{k_m=0}^{n_m-1} \alpha_{k_1 k_2 \dots k_m} \omega_{n_1}^{-j_1 k_1 r_1} \omega_{n_2}^{-j_2 k_2 r_2} \dots \omega_{n_m}^{-j_m k_m r_m}, \\ j_\ell &= 0, \dots, n_\ell - 1, \\ \omega_{n_\ell} &= \exp(2\pi i / n_\ell), \\ r_\ell &= -1 \text{ 又は } 1, \ell = 1, 2, \dots, m \end{aligned} \quad (4)$$

本関数では, (3), (4)の左辺に対応して $\{n_1 n_2 \dots n_m \alpha_{k_1 k_2 \dots k_m}\}$ 又は $\{x_{j_1 j_2 \dots j_m}\}$ を求める. ただし $\text{isin}[i-1] = 0$ となる i に対して, n_i を 1 と置き換える. 従って, 結果の正規化は必要に応じて行うこと.

4. 使用例

2次元実フーリエ変換及び逆変換を計算する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define M      2
#define N1     4
#define N2     4
#define LDIM   2*(N2/2+1)
#define WLEN   2*N1+N1*LDIM

MAIN__()
{
    int ierr, icon;
    double x[LDIM][N1], xx[LDIM][N1], w[WLEN], eps;
    double (*cx)[2][LDIM/2][N1]; /* pointer to complex data */
    int i, j, n[M], isn, isin[M], m, pr;
```



```

/* generate initial data */
m = M;
n[0] = N1;
n[1] = N2;
for (j=0;j<n[1];j++)
    for (i=0;i<n[0];i++)
        x[j][i] = (i+1)*(j+1);
/* keep copy */
for (j=0;j<N2;j++)
    for (i=0;i<N1;i++)
        xx[j][i] = x[j][i];
/* perform normal transform */
isn = 1;
isin[0] = 0;
isin[1] = 1;
pr = n[1];
ierr = c_dvmrft((double*)x, n, m, isin, isn, w, &icon);
/* check results */
if (icon != 0) {
    printf("ERROR: c_dvmrft failed with icon = %d\n", icon);
    exit(1);
}
/* print complex transformed data */
cx = (double(*)[2][LDIM/2][N1])x; /* complex data overwrites real data */
for (j=0;j<N2/2+1;j++) {
    for (i=0;i<N1;i++) {
        printf("%8.5f + i*%8.5f ", (*cx)[0][j][i], (*cx)[1][j][i]);
    }
    printf("\n");
}
/* perform inverse transform */
isn = -1;
isin[0] = 0;
isin[1] = -1;
ierr = c_dvmrft((double*)x, n, m, isin, isn, w, &icon);
/* check results */
eps = 1e-6;
for (j=0;j<n[1];j++)
    for (i=0;i<n[0];i++)
        if (fabs((x[j][i]/pr - xx[j][i])/xx[j][i]) > eps) {
            printf("Inaccurate result\n");
            exit(1);
        }
printf("Result OK\n");
return(0);
}

```

c_dvmsnt

離散型 sine 変換

ierr = c_dvmsnt(x, k, n, m, isw, tab, &icon);

1. 機能

1次元, 多重の離散型 sine 変換を行なう.

周期 2π の奇関数 $x(t)$ の半周期を n 等分し, 零とみなされる両端の値を含まない, $n-1$ 個の標本 $\{x_j\}$

$$x_j = x\left(\frac{\pi}{n}j\right), j = 1, 2, \dots, n-1$$

が与えられたとき, 以下で定義する離散型 sine 変換を配列の各行に対して行なう.

$$a_k = 2 \sum_{j=1}^{n-1} x_j \sin \frac{\pi}{n} kj, \quad k = 1, 2, \dots, n-1 \quad (1)$$

2. 引数

呼出し形式:

```
ierr = c_dvmsnt((double*)x, k, n, m, isw, tab, &icon);
```

引数の説明:

x	double x[m][k]	入力	m 本の $\{x_j\}, j = 1, 2, \dots, n-1$ を x[i1][i2], i1 = 0, ..., m-1, i2 = 0, ..., n-2, に格納する.
		出力	m 本の $\{a_k\}, k = 1, 2, \dots, n-1$ が x[i1][i2], i1 = 0, ..., m-1, i2 = 0, ..., n-2, に格納される.
k	int	入力	x の 2 次元目の大きさ. $k \geq n-1$.
n	int	入力	区間の分割数 n . n は偶数. (“使用上の注意” a)参照)
m	int	入力	変換の多重度 m .
isw	int	入力	制御情報. 0 tab 設定及び sine 変換. 1 tab 設定のみ. 2 設定済みの tab を使用して sine 変換.
tab	double tab[2×n]	作業領域	三角関数表を格納する. (“使用上の注意” b)参照)
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	次のいずれかであった. $n \leq 0$ $k < n-1$ $m \leq 0$ $isw \neq 0, 1, 2$ n が偶数でない.	処理を打ち切る.

3. 使用上の注意

a) n の値について

区間の分割数 n は偶数であれば任意の数が指定可能であるが、2, 3, 5 の因子に分解される場合が高速である。

b) tab の再利用について

同じ n で本関数を繰り返し呼び出す場合、初回に $isw = 0$ 又は 1 で呼び出した後、2 回目以降は $isw = 2$ で呼び出すことにより、tab の作成が省略されて処理が高速化できる。

c) 正規化について

(1)式で定義された sine 変換は、それ自身が逆変換になっている。c_dvmsnt 関数で二回変換を行うと、入力データが $2n$ 倍されたデータが出力される。必要に応じて正規化を行なうこと。

d) スタックサイズについて

本関数では、内部で使用する作業域をスタックに確保している。このため、スタックが不足すると異常終了する可能性がある。本関数がスタックに必要なサイズは、 $16 \times n$ バイトである。この他に、固定サイズの作業領域や、ユーザプログラムで必要とするスタック領域もあるので、limit コマンド又は ulimit コマンドなどで十分な大きさのスタックサイズを指定することを勧める。

4. 使用例

$n = 1024$, 標本点数 1023 の sine 変換を 5 本計算する。

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define N 1024
#define K (N-1)
#define M 5
#define min(a,b) ((a) < (b) ? (a) : (b))

int MAIN__(void)
{
    double x[M][K], tab[N*2];
    double vnrm, error, t1, t2;
    int isw, icon, i, j;

    for (j=0; j<M; j++) {
        for (i=0; i<N-1; i++) {
            x[j][i]=(double)min(i+1, (N-i-1)/(j+1));
        }
    }

    /* FORWARD TRANSFORM */
    isw = 0;
    c_dvmsnt((double*)x, K, N, M, isw, tab, &icon);

    printf("icon = %d\n", icon);

    /* BACKWARD TRANSFORM */
    isw = 2;
    c_dvmsnt((double*)x, K, N, M, isw, tab, &icon);

    printf("icon = %d\n", icon);

    for (j=0; j<M; j++) {
        error = 0.0;
        vnrm = 0.0;
        for (i=0; i<N-1; i++) {
            t1=x[j][i]/(double)(N*2);
            t2=t1-(double)(min(i+1, (N-i-1)/(j+1)));
            vnrm +=t1*t1;
            error+=t2*t2;
        }
        printf("error = %e\n",sqrt(error/vnrm));
    }
}
```

```
} }
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書 II”のVMSNTの項目を参照のこと。

c_dvmvsd

実スパース行列と実ベクトルの積 (対角形式格納法)

<pre>ierr = c_dvmvsd(a, k, ndiag, n, nofst, nlb, x, y, &icon);</pre>

1. 機能

$n \times n$ のスパース行列とベクトルの積を計算する.

$$\mathbf{y} = \mathbf{A}\mathbf{x}$$

スパース行列 $\mathbf{A}$ は, 対角形式の格納法で格納する. ベクトル $\mathbf{x}$ 及び $\mathbf{y}$ は n 次元ベクトル.

2. 引数

呼出し形式:

```
ierr = c_dvmvsd((double*)a, k, ndiag, n, nofst, nlb, x, y, &icon);
```

引数の説明:

a	double	入力	係数行列の非零要素を格納する. (“使用上の注意” a)参照).
	a[ndiag][k]		
k	int	入力	配列 a の整合寸法($\geq n$).
ndiag	int	入力	a に格納する係数行列の非零要素を含む対角ベクトル列の総数. a の 1 次元目の大きさ.
n	int	入力	行列 A の次数 n .
nofst	int	入力	a に格納する対角ベクトルに対応した主対角ベクトルからの距離を格納する. 上対角ベクトル列は正, 下対角ベクトル列は負の値で表す. (“使用上の注意”参照).
	nofst[ndiag]		
nlb	int	入力	行列 A の下バンド幅.
x	double x[Xlen]	入力	ベクトル $\mathbf{x}$ を x[i], $nlb \leq i < nlb+n$ に格納する. $Xlen = n + nlb + nub$. nlb:下バンド幅, nub:上バンド幅.
y	double y[n]	出力	行列とベクトルの積の結果が格納される.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	次のいずれかであった. $k < 1$ $n < 1$ $n > k$ $ndiag < 1$ $nlb \neq \max(-nofst[i]), 0 \leq i < ndiag$ $nofst[i] > n-1, 0 \leq i < ndiag$	処理を打ち切る.

3. 使用上の注意

a) **a** と **nofst** について

行列 **A** は, 2つの配列を使用する対角形式の格納法で格納する. 対角形式の格納方法については, “1.3.5 一般スパース行列の格納方法”を参照のこと.

行列の外側の対角ベクトルの要素は零を設定する必要がある. 対角ベクトル列を配列に格納する順序に制限は特にない. この方法の利点は, 行列ベクトル積が間接指標を使わずに計算できる点である. 対角構造を持たない行列を効率よく格納できないことが欠点である.

4. 使用例

実スパース行列と実ベクトルの積を計算し, 結果を確認する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 100
#define UBANDW 2
#define LBANDW 1

MAIN__()
{
    double one=1.0, eps=1.e-6;
    int ierr, icon;
    int ndiag, nlb, nub, n, i, j, k;
    int nofst[UBANDW + LBANDW + 1];
    double a[UBANDW + LBANDW + 1][NMAX], x[NMAX + UBANDW + LBANDW], y[NMAX];

    /* initialize matrix and vector */
    ndiag = UBANDW + LBANDW + 1;
    nlb = LBANDW;
    nub = UBANDW;
    n = NMAX;
    k = NMAX;
    for (i=1; i<=nub; i++) {
        for (j=0; j<n-i; j++) a[i][j] = -1.0;
        for (j=n-i; j<n; j++) a[i][j] = 0.0;
        nofst[i] = i;
    }
    for (i=1; i<=nlb; i++) {
        for (j=0; j<i; j++) a[nub + i][j] = 0.0;
        for (j=i; j<n; j++) a[nub + i][j] = -2.0;
        nofst[nub + i] = -i;
    }
    for (i=0; i<n+nlb+nub; i++) x[i] = 0.0;
    nofst[0] = 0;
    for (j=0; j<n; j++) {
        a[0][j] = one;
        for (i=1; i<ndiag; i++) a[0][j] -= a[i][j];
        x[nlb + j] = one;
    }
    /* perform matrix-vector multiply */
    ierr = c_dvmvsd((double*)a, k, ndiag, n, nofst, nlb, x, y, &icon);
    if (icon != 0) {
        printf("ERROR: c_dvmvsd failed with icon = %d\n", icon);
        exit(1);
    }
    /* check vector */
    for (i=0; i<n; i++)
        if (fabs(y[i]-one) > eps) {
            printf("WARNING: result inaccurate\n");
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}
```

c_dvmvse

実スパース行列と実ベクトルの積 (ELLPACK 形式格納法)

<pre>ierr = c_dvmvse(a, k, nw, n, icol, x, y, &icon);</pre>

1. 機能

$n \times n$ のスパース行列とベクトルの積を計算する.

$$\mathbf{y} = \mathbf{Ax}$$

$n \times n$ の係数行列は, 2 つの配列を使用する ELLPACK 形式の格納法で格納する. $\mathbf{y}$ 及び $\mathbf{x}$ は n 次元ベクトル.

2. 引数

呼出し形式:

```
ierr = c_dvmvse((double*)a, k, nw, n, (int*)icol, x, y, &icon);
```

引数の説明:

a	double a[nw][k]	入力	係数行列の非零要素を格納する. (“使用上の注意” a)参照).
k	int	入力	配列 a 及び icol の整合寸法 ($\geq n$).
nw	int	入力	a に格納される行列 A の非零要素の行ベクトル方向の最大個数. icol 及び a の 1 次元目の大きさ.
n	int	入力	a に格納される行列 A の次数.
icol	int icol[nw][k]	入力	ELLPACK 形式で使用される列指標で, a の対応する要素がいずれの列ベクトルに属するかを示す. (“使用上の注意”参照).
x	double x[n]	入力	ベクトル $\mathbf{x}$ を格納する.
y	double y[n]	出力	行列とベクトルの積の結果が格納される.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	次のいずれかであった. • $k < 1$ • $n \leq 0$ • $nw < 1$ • $n > k$	処理を打ち切る.

3. 使用上の注意

a) a と icol について

行列 $\mathbf{A}$ は, 2 つの配列を使用する ELLPACK 形式の格納法で格納する. ELLPACK 形式の格納方法については, “1.3.5 一般スパース行列の格納方法”を参照のこと.

ELLPACK 形式でデータを格納する前に、配列 `a` を零で初期化し、`icol` を行ベクトルの番号で初期化することを勧める。

4. 使用例

実スパース行列と実ベクトルの積を計算し、結果を確認する。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX      1000
#define UBANDW     2
#define LBANDW     1

MAIN__()
{
    double lcf=-2.0, ucf=-1.0, one=1.0, eps=1.e-6;
    int ierr, icon;
    int nlb, nub, nw, n, i, j, k, ix;
    int icol[UBANDW + LBANDW + 1][NMAX];
    double a[UBANDW + LBANDW + 1][NMAX], x[NMAX], y[NMAX];

    /* initialize matrix and vector */
    nub = UBANDW;
    nlb = LBANDW;
    nw = UBANDW + LBANDW + 1;
    n = NMAX;
    k = NMAX;
    for (i=0; i<n; i++) x[i] = one;
    for (i=0; i<nw; i++)
        for (j=0; j<n; j++) {
            a[i][j] = 0.0;
            icol[i][j] = j+1;
        }
    for (j=0; j<nlb; j++) {
        for (i=0; i<j; i++) a[i][j] = lcf;
        a[j][j] = one - (double) j * lcf - (double) nub * ucf;
        for (i=j+1; i<j+1+nub; i++) a[i][j] = ucf;
        for (i=0; i<=nub+j; i++) icol[i][j] = i+1;
    }
    for (j=nlb; j<n-nub; j++) {
        for (i=0; i<nlb; i++) a[i][j] = lcf;
        a[nlb][j] = one - (double) nlb * lcf - (double) nub * ucf;
        for (i=nlb+1; i<nw; i++) a[i][j] = ucf;
        for (i=0; i<nw; i++) icol[i][j] = i+1+j-nlb;
    }
    for (j=n-nub; j<n; j++){
        for (i=0; i<nlb; i++) a[i][j] = lcf;
        a[nlb][j] = one - (double) nlb * lcf - (double) (n-j-1) * ucf;
        for (i=1; i<nub-2+n-j; i++) a[i+nlb][j] = ucf;
        ix = n - (j+nub-nlb-1);
        for (i=n; i>=j+nub-nlb-1; i--) icol[ix--][j] = i;
    }
    /* perform matrix-vector multiply */
    ierr = c_dvmvse((double*)a, k, nw, n, (int*)icol, x, y, &icon);
    if (icon != 0) {
        printf("ERROR: c_dvmvse failed with icon = %d\n", icon);
        exit(1);
    }
    /* check vector */
    for (i=0; i<n; i++)
        if (fabs(y[i]-one) > eps) {
            printf("WARNING: result inaccurate\n");
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}
```


c_dvqmrld

非対称又は不定値のスパース実行列の連立 1 次方程式 (QMR 法, 対角形式格納法)

```
ierr = c_dvqmrld(a, k, ndiag, n, nofst, at,
                  ntofst, b, itmax, eps, iguss, x,
                  &iter, vw, &icon);
```

1. 機能

$n \times n$ の非対称/不定値なスパース行列を係数行列とする連立 1 次方程式(1)を準最小残差法(QMR:Quasi-Minimal Residual method)で解く.

$$\mathbf{Ax} = \mathbf{b} \quad (1)$$

$n \times n$ の係数行列 $\mathbf{A}$ 及び $\mathbf{A}^T$ の 2 つの行列を使用する. これら是对角形式格納法で格納する. ベクトル $\mathbf{b}$ 及び $\mathbf{x}$ は n 次元ベクトルである.

本方法は一般に MGCR 法より高速だが, 反復計算が係数行列や初期ベクトルの特性のため続けられない場合 (break down)がある. これは再帰的計算公式で, 非零を期待される計算のある中間結果が零になるためである. この場合, break down を起こさない MGCR 法をお使いいただきたい.

2. 引数

呼出し形式:

```
ierr = c_dvqmrld((double *) a, k, ndiag, n, nofst, (double *) at, ntofst, b,
                  itmax, eps, iguss, x, &iter, vw, &icon);
```

引数の説明:

a	double a[ndiag][k]	入力	係数行列 $\mathbf{A}$. 一般スパース行列の対角形式格納法. 格納法の詳細については“1.3.5 一般スパース行列の格納方法”を参照のこと.
k	int	入力	配列 a の整合寸法 ($\geq n$).
ndiag	int	入力	係数行列 $\mathbf{A}$ の非零要素を含む対角ベクトル列の総数
n	int	入力	行列 $\mathbf{A}$ の次数 n .
nofst	int nofst[ndiag]	入力	配列 a に格納される対角ベクトルに対応した主対角ベクトルからの距離を格納する. 上対角ベクトル列は正, 下対角ベクトル列は負の値で表す.
at	double at[ndiag][k]	入力	転置した係数行列 $\mathbf{A}^T$. 一般スパース行列の対角形式格納法. 格納法の詳細については“1.3.5 一般スパース行列の格納方法”を参照のこと.
ntofst	int ntofst[ndiag]	入力	配列 at に格納される対角ベクトルに対応した主対角ベクトルからの距離を格納する. 上対角ベクトル列は正, 下対角ベクトル列は負の値で表す.
b	double b[n]	入力	定数ベクトル $\mathbf{b}$.
itmax	int	入力	QMR 法の反復回数の上限值(> 0). 2000 程度で十分である.
eps	double	入力	収束判定に用いられる判定値. eps が 0.0 以下のとき, eps は 10^{-6} が設定される. (“使用上の注意”a)参照)
iguss	int	入力	配列 x に指定された解ベクトルの近似値から反復計算を行うかを指定する制御情報. 0 のとき, 解ベクトルの近似値を指定しない.

			0 以外のとき, 配列 x に指定された解ベクトルの近似値から反復計算を開始する.
x	double $x[n]$	入力	解ベクトルの近似値を指定することができる.
		出力	解ベクトル x .
iter	int	出力	QMR 法での実際の反復回数.
vw	double vw[Vwlen]	作業領域	$Vwlen = 9k + n + ndiag - 1$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	Break down を起こした.	処理を打ち切る.
20001	反復回数の上限に達した.	処理を打ち切る. 配列 x には, そのときまでに得られている近似値を出力するが, 精度は保証できない.
30000	次のいずれかであった. $n < 1$ $k < 1$ 又は $k < n$ $ndiag < 1$ 又は $ndiag > k$ $itmax \leq 0$	処理を打ち切る.
32001	$ nofst[i-1] > n-1$ 又は $ ntofst[i-1] > n-1$	処理を打ち切る.

3. 使用上の注意

a) eps について

QMR 法は, 残差のユーグリッドノルムが最初の残差のユーグリッドノルムと eps の積以下になったとき収束したと見なす. 正確な解と求められた近似解の誤差はほぼ行列 A の条件数と eps の積に等しくなる.

b) 対角形式を使う上での注意

係数行列 A の外側の対角ベクトルの要素は零を設定する必要がある. 対角ベクトル列を配列 a に格納する順序に制限は特にない. この方法の利点は, 行列ベクトル積が間接指標を使わずに計算できる点である. 対角構造を持たない行列は効率よく格納できないという点が欠点である.

4. 使用例

実スパース行列の連立 1 次方程式を解く.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX      100
#define UBANDW     2
#define LBANDW     1

MAIN__()
{
    double one=1.0, zero=0.0, lcoef=-2.0, ucoef=-1.0, bcoef=10.0, eps=1.e-06;
    int ierr, icon, ndiag, nub, nlb, n, itmax, iguss, iter, i, j, k;
    int nofst[UBANDW + LBANDW + 1], ntofst[UBANDW + LBANDW + 1];
    double a[UBANDW + LBANDW + 1][NMAX], at[UBANDW + LBANDW + 1][NMAX];
    double b[NMAX], x[NMAX], vw[NMAX * 9 + NMAX + UBANDW + LBANDW];

    nub = UBANDW;
```

```

nlb   = LBANDW;
ndiag = nub + nlb + 1;
n     = NMAX;
k     = NMAX;

/* Set A-mat & b */
for (i=1; i<=nub; i++) {
    for (j=0 ; j<n-i; j++) a[i][j] = ucoef;
    for (j=n-i; j<n ; j++) a[i][j] = zero;
    nofst[i] = i;
}
for (i=1; i<=nlb; i++) {
    for (j=0; j<i; j++) a[nub + i][j] = zero;
    for (j=i; j<n; j++) a[nub + i][j] = lcoef;
    nofst[nub + i] = -i;
}
nofst[0] = 0;
for (j=0; j<n; j++) {
    b[j] = bcoef;
    a[0][j] = bcoef;
    for (i=1; i<ndiag; i++) b[j] += a[i][j];
}
/* Set A-mat transpose */
ntofst[0] = 0;
for (j=0; j<n; j++) at[0][j] = a[0][j];
for (i=1; i<ndiag; i++) {
    ntofst[i] = - nofst[i];
    for (j=0; j<n; j++) at[i][j] = a[i][n-j-1];
}
/* solve the nonsymmetric system of linear equations */
itmax = 2000;
iguss = 0;
ierr = c_dvqmrdr ((double*)a, k, ndiag, n, nofst, (double*)at,
                  ntofst, b, itmax, eps, iguss, x, &iter, vw, &icon);
if (icon != 0) {
    printf("ERROR: c_dvqmrdr failed with icon = %d\n", icon);
    exit(1);
}
/* check result */
for (i=0; i<n; i++)
    if (fabs(x[i]-one) > eps*10.0) {
        printf("WARNING: result maybe inaccurate\n");
        exit(1);
    }
printf("Result OK\n");
exit(0);
}

```

5. 手法概要

QMR 法については[37]を参照のこと.

c_dvqmre

非対称又は不定値のスパース実行列の連立 1 次方程式 (QMR 法, ELLPACK 形式格納法)

<pre>ierr = c_dvqmre(a, k, iwidtt, n, icol, at, iwidtt, icolt, b, itmax, eps, iguss, x, &iter, vw, &icon);</pre>

1. 機能

$n \times n$ の非対称/不定値なスパース行列を係数行列とする連立 1 次方程式(1)を準最小残差法(QMR:Quasi-Minimal Residual method)で解く.

$$\mathbf{Ax} = \mathbf{b} \quad (1)$$

$n \times n$ の係数行列 $\mathbf{A}$ 及び $\mathbf{A}^T$ の 2 つの行列を使用する. これらは ELLPACK 形式格納法で格納する. ベクトル $\mathbf{b}$ 及び $\mathbf{x}$ は n 次元ベクトルである.

本方法は一般に MGCR 法より高速だが, 反復計算が係数行列や初期ベクトルの特性のため続けられない場合 (break down)がある. これは再帰的計算公式で, 非零を期待される計算のある中間結果が零になるためである. この場合, break down を起こさない MGCR 法をお使いいただきたい.

2. 引数

呼出し形式:

```
ierr = c_dvqmre((double *) a, k, iwidtt, n, (double *) icol, (double *) at,
                iwidtt, (double *) icolt, b, itmax, eps, iguss, x, &iter, vw,
                &icon);
```

引数の説明:

a	double a[iwidtt][k]	入力	係数行列 $\mathbf{A}$. 一般スパース行列の ELLPACK 形式格納法. 格納法の詳細については“1.3.5 一般スパース行列の格納方法”を参照のこと.
k	int	入力	配列 a, at, icol, icolt の整合寸法 ($\geq n$).
iwidtt	int	入力	係数行列 $\mathbf{A}$ の非零要素の行ベクトル方向の最大個数.
n	int	入力	行列 $\mathbf{A}$ の次数 n .
icol	int icol[iwidtt][k]	入力	ELLPACK 形式で使用される列指標で, a の対応する要素が いずれの列ベクトルに属するかを示す.
at	double at[iwidttt][k]	入力	転置した係数行列 $\mathbf{A}^T$. 一般スパース行列の ELLPACK 形式格納法. 格納法の詳細については“1.3.5 一般スパース行列の格納方法”を参照のこと.
iwidttt	int	入力	転置した係数行列 $\mathbf{A}^T$ の非零要素の行ベクトル方向の最大個数.
icoltt	int icolt [iwidttt][k]	入力	ELLPACK 形式で使用される列指標で, at の対応する要素が いずれの列ベクトルに属するかを示す.
b	double b[n]	入力	定数ベクトル $\mathbf{b}$.
itmax	int	入力	QMR 法の反復回数の上限值 (> 0). 2000 程度で十分である.
eps	double	入力	収束判定に用いられる判定値. eps が 0.0 以下のとき, eps は 10^{-6} が設定される. (“使用上の注意”参照)
iguss	int	入力	配列 x に指定された解ベクトルの近似値から反復計算を行うかを指定する制御情報.

			0 のとき, 解ベクトルの近似値を指定しない. 0 以外のとき, 配列 x に指定された解ベクトルの近似値から反復計算を開始する.
x	double $x[n]$	入力	解ベクトルの近似値を指定することができる.
		出力	解ベクトル x .
iter	int	出力	QMR 法での実際の反復回数.
vw	double $vw[12k]$	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	Break down を起こした.	処理を打ち切る.
20001	反復回数の上限に達した.	処理を打ち切る. 配列 x には, そのときまでに得られている近似値を出力するが, 精度は保証できない.
30000	次のいずれかであった. $n < 1$ $k < 1$ 又は $k < n$ $iwidth < 1$ 又は $iwidth > k$ $iwidtht < 1$ 又は $iwidtht > k$ $itmax \leq 0$	処理を打ち切る.

3. 使用上の注意

a) eps について

QMR 法は, 残差のユーグリッドノルムが最初の残差のユーグリッドノルムと eps の積以下になったとき収束したと見なす. 正確な解と求められた近似解の誤差はほぼ行列 A の条件数と eps の積に等しくなる.

4. 使用例

実スパース行列の連立 1 次方程式を解く.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX      100
#define UBANDW    2
#define LBANDW    1

MAIN__()
{
    double lcf=-2.0, ucf=-1.0, bcoef=10.0, one=1.0, zero = 0.0, eps=1.e-06;
    int ierr, icon, nlb, nub, iwidth, iwidtht, n, k, itmax, iguss, iter, i, j, ix;
    int icol[UBANDW + LBANDW + 1][NMAX], icolt[UBANDW + LBANDW + 1][NMAX];
    double a[UBANDW + LBANDW + 1][NMAX], at[UBANDW + LBANDW + 1][NMAX];
    double b[NMAX], x[NMAX], vw[NMAX * 12];

    nub    = UBANDW;
    nlb    = LBANDW;
    iwidth = UBANDW + LBANDW + 1;
    iwidtht = iwidth;
    n      = NMAX;
    k      = NMAX;

    /* Initialize A-mat and A-mat transpose */
    for (i=0; i<iwidth; i++)
        for (j=0; j<n; j++) {
```

```
        a[i][j] = zero;
        at[i][j] = zero;
        icol[i][j] = j+1;
        icolt[i][j] = j+1;
    }
    /* Set A-mat & b */
    for (j=0; j<nlb; j++) {
        for (i=0; i<j; i++) a[i][j] = lcf;
        a[j][j] = bcoef;
        b[j] = bcoef + (double) j * lcf + (double) nub * ucf;
        for (i=j+1; i<j+1+nub; i++) a[i][j] = ucf;
        for (i=0; i<=nub+j; i++) icol[i][j] = i+1;
    }
    for (j=nlb; j<n-nub; j++) {
        for (i=0; i<nlb; i++) a[i][j] = lcf;
        a[nlb][j] = bcoef;
        b[j] = bcoef + (double) nlb * lcf + (double) nub * ucf;
        for (i=nlb+1; i<iwidtt; i++) a[i][j] = ucf;
        for (i=0; i<iwidtt; i++) icol[i][j] = i+1+j-nlb;
    }
    for (j=n-nub; j<n; j++){
        for (i=0; i<nlb; i++) a[i][j] = lcf;
        a[nlb][j] = bcoef;
        b[j] = bcoef + (double) nlb * lcf + (double) (n-j-1) * ucf;
        for (i=1; i<nub-2+n-j; i++) a[i+nlb][j] = ucf;
        ix = n - (j+nub-nlb-1);
        for (i=n; i>=j+nub-nlb-1; i--) icol[ix--][j] = i;
    }
    /* Set A-mat transpose */
    for (j=0; j<nub; j++) {
        for (i=0; i<j; i++) at[i][j] = ucf;
        at[j][j] = bcoef;
        for (i=j+1; i<j+1+nlb; i++) at[i][j] = lcf;
        for (i=0; i<=nlb+j; i++) icolt[i][j] = i+1;
    }
    for (j=nub; j<n-nlb; j++) {
        for (i=0; i<nub; i++) at[i][j] = ucf;
        at[nub][j] = bcoef;
        for (i=nub+1; i<iwidtt; i++) at[i][j] = lcf;
        for (i=0; i<iwidtt; i++) icolt[i][j] = i+1+j-nub;
    }
    for (j=n-nlb; j<n; j++){
        for (i=0; i<nub; i++) at[i][j] = ucf;
        at[nub][j] = bcoef;
        for (i=1; i<nlb-1+n-j; i++) at[i+nub][j] = lcf;
        ix = n - (j+nlb-nub);
        for (i=n; i>=j+nlb-nub; i--) icolt[ix--][j] = i;
    }
    /* solve the nonsymmetric system of linear equations */
    itmax = 2000;
    iguss = 0;
    ierr = c_dvqmre ((double*)a, k, iwiddt, n, (int*)icol, (double*)at,
                    iwiddtt, (int*)icolt, b, itmax, eps, iguss, x,
                    &iter, vw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dvqmre failed with icon = %d\n", icon);
        exit(1);
    }
    /* check result */
    for (i=0; i<n; i++)
        if (fabs(x[i]-one) > eps*10.0) {
            printf("WARNING: result maybe inaccurate\n");
            exit(1);
        }
    printf("Result OK\n");
    exit(0);
}
```

5. 手法概要

QMR 法については[37]を参照のこと。

c_dvran3

正規乱数の生成

```
ierr = c_dvran3(dam, dsd, &ix, da, n, dwork,
               nwork, &icon);
```

1. 機能

平均 m と標準偏差 σ を持った正規分布の密度関数から疑似乱数を生成する.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot \exp\left(-\frac{(x-m)^2}{2\sigma^2}\right)$$

2. 引数

呼出し形式:

```
ierr = c_dvran3(dam, dsd, &ix, da, n, dwork, nwork, &icon);
```

引数の説明:

dam	double	入力	正規分布の平均 m .
dsd	double	入力	正規分布の標準偏差 σ .
ix	int	入力	出発値. 初回呼び出しでは, ix に正の値を与え, 2 回目以降の呼び出しでは返却値 0 のまま呼び出すこと. 初回呼び出しで指定する出発値が異なると, 異なる乱数列が生成される. (“使用上の注意” a)参照)
		出力	0.
da	double da[n]	出力	n 個の疑似乱数.
n	int	入力	da に返却される正規分布疑似乱数の個数.
dwork	double dwork[nwork]	作業領域	本関数を繰り返し呼び出す場合は, 内容を変更してはならない. dwork には, 本関数が再度呼び出される場合に必要な情報が格納される.
nwork	int	入力	配列 dwork の大きさ. $nwork \geq 1,156$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30001	nwork が小さ過ぎた.	処理を打ち切る.
30002	ix < 0.	
30003~30008	dwork が変更されていた. 又は, 初回呼び出しで ix に 0 が与えられた.	

3. 使用上の注意

本関数は, 最低でも周期が 10^{52} 以上である, 長周期の一樣乱数を利用する Polar 法で正規乱数を生成している. 異なる出発値は異なる乱数列, つまり長周期の乱数列を少なくとも 2^{60} ($> 10^{18}$) 以上の間隔を持って分割

してできた異なる部分乱数列より乱数列を生成する。詳細は“FUJITSU SSL II 拡張機能使用手引書 II”の DVRAU4 の手法概要を参照すること。

a) ix について

確定的な計算(deterministic program)で擬似乱数列を生成するには、何かランダムな入力が必要になる。このために、利用者が出発値 ix を与える必要がある。本関数の初回呼び出しでは、出発値 ix は正整数である必要がある(例外事項については e)を参照)。2 回目以降の呼び出しでは ix には 0 を指定すること。これは同じ乱数列から続いて乱数を求めるためである。プログラミングを容易にするため、本関数は初回呼び出し後 ix に 0 を返している。

b) n について

本関数は出発値 ix で定義される無限列から、次の n 個の疑似乱数を返却する。n ≤ 0 の時は疑似乱数を返却しない。

効率をよくするためには、利用者は n を十分大きく(例えば n = 100,000)する。それは、関数呼び出しのオーバーヘッドが減少し、しかもベクトル化されるためである。本関数を連続的に呼び出す途中で、n が変更される場合は、配列 da を最大の n の大きさだけ確保する必要がある。

c) dwork について

dwork は、複数回本関数を呼び出す場合、次の呼び出しのための情報を格納する作業領域として使用される。従って、本関数を呼び出している間は、呼び出し側のプログラムで dwork の内容を変更してはならない。

d) nwork について

dwork[0], ..., dwork[nwork-1]は、本関数によって使用される。nwork は、本関数の各呼び出しで変えないこと。nwork には少なくとも 1,156 以上、ベクトルプロセッサ上では十分大きな数(例えば nwork = 100,000)を与えると効率的である。

e) 同じ乱数の再生成について

dwork[0], ..., dwork[nwork-1]が保存される場合、その dwork を再使用し、ix = 0 として本関数を呼び出すことにより、(dwork が保存された時点からの)同じ乱数列が再度生成される。

4. 使用例

10000 個の正規乱数を生成し、平均と標準偏差を計算する。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 10000

MAIN__()
{
    int ierr, icon;
    int n, nwork, ix, i;
    double dsd, dam, dwork[NMAX], da[NMAX], sum, sumsq, mean, dev;

    /* initialize parameters */
    n = NMAX;
    nwork = NMAX;
    ix = 12345;
    dsd = 1;
    dam = 0;

    /* generate pseudo-random numbers */
    ierr = c_dvran3(dam, dsd, &ix, da, n, dwork, nwork, &icon);
    if (icon != 0) {
        printf("ERROR: c_dvran3 failed with icon = %d\n", icon);
        exit(1);
    }
}
```



```
    }  
    /* calculate mean and normal deviation */  
    sum = 0;  
    sumsq = 0;  
    for (i=0;i<n;i++) {  
        sum = sum+da[i];  
        sumsq = sumsq+da[i]*da[i];  
    }  
    mean = sum/n;  
    dev = sqrt(sumsq/n - mean*mean);  
    printf("observed mean = %12.4e    deviation = %12.4e\n",  
           mean, dev);  
    printf("calculated mean = %12.4e    deviation = %12.4e\n",  
           dam, dsd);  
    return(0);  
}
```

5. 手法概要

本関数は正規分布する疑似乱数を生成するために、初等関数の高速な計算法を利用した Polar 法[64] (Polar method)を使用している. 計算法の詳細については[10]を参照のこと. この方法に必要な一様乱数の生成方法については、“FUJITSU SSL II 拡張機能使用手引書 II”の DVRAU4 の項目を参照のこと.

c_dvran4

正規乱数の生成 (Wallace 法)

<pre>ierr = c_dvran4(dam, dsd, &ix, da, n, dwork, nwork, &icon);</pre>

1. 機能

与えられた平均 m と標準偏差 σ を持った正規分布の密度関数 (1) から, 擬似乱数を生成する.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot \exp\left(-\frac{(x-m)^2}{2\sigma^2}\right) \quad (1)$$

2. 引数

呼出し形式:

```
ierr = c_dvran4(dam, dsd, &ix, da, n, dwork, nwork, &icon);
```

引数の説明:

dam	double	入力	正規分布の平均 m .
dsd	double	入力	正規分布の標準偏差 σ .
ix	int	入力	出発値. 初回呼び出しでは, ix に正の値を与え, 2 回目以降の呼び出しでは返却値 0 のまま呼び出すこと. 初回呼び出しで指定する出発値が異なると, 異なる乱数列が生成される. (“使用上の注意” a)参照)
da	double da[n]	出力	0.
n	int	出力	n 個の擬似乱数.
dwork	double dwork[nwork]	入力	da に返却される正規分布擬似乱数の個数.
nwork	int	作業領域	本関数を繰り返し呼び出す場合は, 内容を変更してはならない. dwork には, 本関数が再度呼び出される場合に必要な情報が格納される.
icon	int	入力	配列 dwork の大きさ. $nwork \geq 1,350$.
		出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30001	nwork が小さ過ぎた. 又は ix < 0, dsd ≤ 0.	処理を打ち切る.
30002	内部エラー.	
30003~30008	dwork が変更されていた. 又は, 初回呼び出しで ix に 0 が与えられた.	
30009	ix が大きすぎる.	
40001~40002	dwork は上書きされた. 又は, 初回呼び出しで ix に 0 が与えられた.	

3. 使用上の注意

a) ix について

確定的な計算(deterministic program)で擬似乱数列を生成するには、何かランダムな入力が必要になる。このために、利用者が出発値 ix を与える必要がある。本関数の初回呼び出しでは、出発値 ix は正整数である必要がある(例外事項については e)を参照)。2回目以降の呼び出しでは ix には 0 を指定すること。これは同じ乱数列から続いて乱数を求めるためである。プログラミングを容易にするため、本関数は初回呼び出し後 ix に 0 を返している。

b) n について

本関数は出発値 ix で定義される無限列から、次の n 個の疑似乱数を返却する。n ≤ 0 の時は疑似乱数を返却しない。

効率をよくするためには、利用者は n を十分大きく(例えば n = 100,000)する。それは、関数呼び出しのオーバーヘッドが減少する。本関数を連続的に呼び出す途中で、n が変更される場合は、配列 da を最大の n の大きさだけ確保する必要がある。

c) dwork について

dwork は、複数回本関数を呼び出す場合、次の呼び出しのための情報を格納する作業領域として使用される。従って、本関数を呼び出している間は、呼び出し側のプログラムで dwork の内容を変更してはならない。

d) nwork について

dwork[0], ..., dwork[nwork-1]は、本関数によって使用される。nwork は、本関数の各呼び出しで変えないこと。nwork には少なくとも 1,350 以上を与えると効率的である。

e) 同じ乱数の再生成について

dwork[0], ..., dwork[nwork-1]が保存される場合、その dwork を再使用し、ix = 0 として本関数を呼び出すことにより、(dwork が保存された時点からの)同じ乱数列が再度生成される。

f) Wallace 法について

c_dvran4 で使われている Wallace の方法は c_dvran3 で使われている Polar 法に比べて約 3 倍高速である。ただし、統計的特性が厳しく問われる場合は c_dvran3 の利用を薦める。

4. 使用例

10000 個の正規乱数を生成し、平均と標準偏差を計算する。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 10000

MAIN__()
{
    int ierr, icon;
    int n, nwork, ix, i;
    double dsd, dam, dwork[NMAX], da[NMAX], sum, sumsq, mean, dev;

    /* initialize parameters */
    n = NMAX;
    nwork = NMAX;
    ix = 12345;
    dam = 0;
    dsd = 1;

    /* generate pseudo-random numbers */
```

```
ierr = c_dvran4(dam, dsd, &iix, da, n, dwork, nwork, &icon);
if (icon != 0) {
    printf("ERROR: c_dvran4 failed with icon = %d\n", icon);
    exit(1);
}
/* calculate mean and normal deviation */
sum = 0;
sumsq = 0;
for (i=0;i<n;i++) {
    sum = sum+da[i];
    sumsq = sumsq+da[i]*da[i];
}
mean = sum/n;
dev = sqrt(sumsq/n - mean*mean);
printf("observed mean = %12.4e    deviation = %12.4e\n",
       mean, dev);
printf("calculated mean = %12.4e    deviation = %12.4e\n",
       dam, dsd);
return(0);
}
```

5. 手法概要

本関数は正規分布する疑似乱数を生成するために、Wallace の方法の一種の変形を使用している。計算方法の詳細については[8], [10], [115]を参照のこと。この方法に必要な一様乱数の生成方法については、“FUJITSU SSL II 拡張機能使用手引書 II”の DVRAN4 及び DVRAU4 の項目を参照のこと。

c_dvr4u4

一様乱数 $[0, 1)$ の生成

<pre>ierr = c_dvr4u4(&ix, da, n, dwork, nwork, &icon);</pre>

1. 機能

区間 $[0, 1)$ 上での一様分布から擬似乱数列を生成する。

2. 引数

呼び出し形式:

```
ierr = c_dvr4u4(&ix, da, n, dwork, nwork, &icon);
```

引数の説明:

ix	int	入力	出発値. 初回呼び出しでは ix に正の値を与え, 2 回目以降の呼び出しでは返却値 0 のまま呼び出すこと. 初回呼び出しで指定する ix の値が異なると, 異なる乱数列が生成される. (“使用上の注意” a) 参照).
		出力	0.
da	double da[n]	出力	区間 $[0, 1)$ で一様な n 個の擬似乱数.
n	int	入力	da に返却される一様分布擬似乱数の個数.
dwork	double dwork[nwork]	作業領域	本関数を繰り返し呼び出す場合は, 内容を変更してはならない. dwork には, 本関数が再度呼び出される場合に必要な情報が格納される.
nwork	int	入力	配列 dwork の大きさ. $nwork \geq 388$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30001	nwork が小さ過ぎた.	処理を打ち切る.
30002	ix < 0.	
30003~30008	dwork が変更されていた. 又は, 初回呼び出しで ix に 0 が与えられた.	

3. 使用上の注意

a) ix について

確定的な計算(deterministic program)で擬似乱数列を生成するには, 何かランダムな入力が必要になる. このために, 利用者が出発値 ix を与える必要がある. 本関数の初回呼び出しでは, 出発値 ix は正整数である必要がある(例外事項については e)を参照). 2 回目以降の呼び出しでは ix には 0 を指定すること. これは同じ乱数列から続いて乱数を求めるためである. プログラミングを容易にするため, 本関数は初回呼び出し後 ix に 0 を返している.

b) n について

本関数は出発値 `ix` で定義される無限列から、次の `n` 個の疑似乱数を返却する。 $n \leq 0$ の時は疑似乱数を返却しない。

効率をよくするためには、利用者は `n` を十分大きく(例えば `n = 100,000`)する。それは、関数呼び出しのオーバーヘッドが減少し、しかもベクトル化されるためである。本関数を連続的に呼び出す途中で、`n` が変更される場合は、配列 `da` を最大の `n` の大きさだけ確保する必要がある。

c) dwork について

`dwork` は、複数回本関数を呼び出す場合、次の呼び出しのための情報を格納する作業領域として使用される。従って、本関数を呼び出している間は、呼び出し側のプログラムで `dwork` の内容を変更してはならない。

d) nwork について

`dwork[0], ..., dwork[nwork-1]` は、本関数によって使用される。`nwork` は、本関数の各呼び出しで変えてはならない。`nwork` には少なくとも 388 以上、ベクトルプロセッサ上では十分大きな数(例えば `nwork=45,000`)を与えると効率的である。

e) 同じ乱数の再生成について

`dwork[0], ..., dwork[nwork-1]` が保存される場合、その `dwork` を再使用し、`ix = 0` として本関数を呼び出すことにより、(`dwork` が保存された時点からの)同じ乱数列が再度生成される。

4. 使用例

10000 個の一様乱数を計算し、平均と標準偏差を計算する。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 10000

MAIN__()
{
    int ierr, icon;
    int n, nwork, ix, i;
    double dwork[NMAX], da[NMAX], sum, sumsq, mean, dev;

    /* initialize parameters */
    n = NMAX;
    nwork = NMAX;
    ix = 12345;
    /* generate pseudo-random numbers */
    ierr = c_dvr4u4(&ix, da, n, dwork, nwork, &icon);
    if (icon != 0) {
        printf("ERROR: c_dvr4u4 failed with icon = %d\n", icon);
        exit(1);
    }
    /* calculate mean and normal deviation */
    sum = 0;
    sumsq = 0;
    for (i=0; i<n; i++) {
        sum = sum+da[i];
        sumsq = sumsq+da[i]*da[i];
    }
    mean = sum/n;
    dev = sqrt(sumsq/n - mean*mean);
    printf("observed mean = %12.4e    deviation = %12.4e\n",
           mean, dev);
    printf("calculated mean = %12.4e    deviation = %12.4e\n",
           0.5, sqrt(1.0/12));
    return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書 II”の DVRAU4 の項目及び[6], [11], [12], [32], [58], [70]を参照のこと.

c_dvrcvf

実データの離散畳み込み及び離散相関

<pre>ierr = c_dvrcvf(x, k, n, m, y, ivr, isw, tab, &icon);</pre>

1. 機能

実データの 1 次元, 多重の離散畳み込み又は離散相関をフーリエ変換を利用して計算する.

畳み込み

フィルターと, 1 本のデータをそれぞれ $\{y_i\}$ と $\{x_j\}$ としたとき, 以下で定義される畳み込みをデータ配列の各列に対して行い, $\{z_k\}$ を計算する.

$$z_k = \sum_{i=0}^{n-1} x_{k-i} y_i, \quad k = 0, \dots, n-1$$

相関

フィルターと, 1 本のデータをそれぞれ $\{y_i\}$ と $\{x_j\}$ としたとき, 以下で定義される相関をデータ配列の各列に対して行い, $\{z_k\}$ を計算する.

$$z_k = \sum_{i=0}^{n-1} x_{k+i} y_i, \quad k = 0, \dots, n-1$$

ここで, x_j は周期 n の周期的データである. (“使用上の注意” a) 参照)

2. 引数

呼出し形式:

```
ierr = c_dvrcvf ((double*)x, k, n, m, y, ivr, isw, tab, &icon);
```

引数の説明:

x	double x[m][k]	入力	m 本の $\{x_j\}$ データ, $j = 0, \dots, n-1$, を $x[i][j]$, $i = 0, \dots, m-1, j = 0, \dots, n-1$, に格納する.
		出力	m 本の $\{z_k\}$ データ, $k = 0, \dots, n-1$, が $x[i][k]$, $i = 0, \dots, m-1, k = 0, \dots, n-1$, に格納される.
k	int	入力	データ配列 x の 2 次元目の大きさ.
n	int	入力	1 本のデータの要素数 n . n は偶数. (“使用上の注意” b) 参照)
m	int	入力	多重度 m .
y	double y[n]	入力	フィルター $\{y_i\}$, $i = 0, \dots, n-1$, を $y[i]$ に格納する.
			$isw = 0$ 又は 2 のとき, 値は保存されない. (“使用上の注意” c), d) 参照)
ivr	int	入力	畳み込みと相関の処理を選択する.
			0 畳み込みを計算する.
			1 相関を計算する.
isw	int	入力	制御情報.
			0 1 回の呼出しで処理を完了する.
			以下の isw 指定で順次呼び出すことで, 処理のステップを分けて利用することができる. (“使用上の注意” c) 参照)

			1 <code>tab</code> 設定のみ行う.
			2 y のフーリエ変換を行う.
			3 設定済みの <code>tab</code> と変換済みの y を使用して畳み込み 又は相関を計算する.
<code>tab</code>	<code>double</code> <code>tab[2×n]</code>	作業領域	三角関数表を格納する. ("使用上の注意" c) 参照)
<code>icon</code>	<code>int</code>	出力	コンディションコード. 下の表を参照.

コンディションコード：

コード	意味	処理内容
0	エラーなし	正常終了.
30000	次のいずれかであった. $n \leq 0$ $k < n$ $m \leq 0$ $ivr \neq 0, 1$ $isw \neq 0, 1, 2, 3$ n が偶数でなかった	処理を打ち切る.

3. 使用上の注意

a) 非周期的データに対する畳み込みおよび相関について

データ x の長さが n_x , y の長さが n_y であるとき, $n \geq n_x + n_y - 1$ として $x[i][j]$, $i = 0, \dots, m-1, j = n_x, \dots, n-1$, 及び $y[i]$, $i = n_y, \dots, n-1$, には 0 を設定しておくことで, 非同期データに対する畳み込み及び相関が計算できる. ("使用例" b) 参照)

このとき, 相関の $z_k(k = -n_y + 1, \dots, -1)$ に相当する値は, $x[i][j]$, $i = 0, \dots, m-1, j = n - n_y + 1, \dots, n-1$, に格納される.

b) n の値について

1 本のデータの要素 n は任意の数が指定可能であるが, 2, 3, 5 の因子に分解される場合が高速である.

c) `tab` 及び y の再利用について

同じ n で本関数を繰り返し呼び出す場合, 初回に $isw = 0$ 又は 1 で呼び出した後, 2 回目以降は $isw = 2$ と 3 で呼び出すことにより `tab` 作成を省略し, 処理の高速化ができる. さらに, 同じフィルター y を使用する場合には, $isw = 0$ 又は 2 で変換された y を $isw = 3$ の呼び出しで利用することで処理が効率化できる.

このように本関数を繰り返し呼び出す場合, y に対して 2 回以上の変換が行われないように注意が必要である.

d) 自己相関計算について

引数 x と y に同一の実引数を指定することで, 自己相関の計算が可能である. このとき, $isw = 2$ の指定は無視される. ("使用例" c) 参照)

e) スタックサイズについて

本関数では, 内部で使用する作業域をスタック域に確保している. このため, スタックが不足すると異常終了する可能性がある. 本関数がスタック域に必要なサイズは, $8 \times n$ バイトである.

この他に, 固定サイズの作業域や, ユーザプログラムで必要とするスタック域もあるので, `limit` コマンド又は `ulimit` コマンドなどで十分な大きさのスタックサイズを指定することを勧める.

4. 使用例

a) 周期的データに対する畳み込みを計算する.

```
#include <stdio.h>
#include <stdlib.h>
#include "cssl.h" /* standard C-SSL header file */

#define K 8
#define M 3

int MAIN__(void)
{
    double x[M][K], y[K], tab[K*2];
    int    ivr, isw, icon;
    int    i, j, n;

    n = 8;

    for (j=0; j<M; j++) {
        for (i=0; i<n; i++) {
            x[j][i] = (double)(i+j+1);
        }
    }

    for (i=0; i<n; i++) {
        y[i] = (double)(i+11);
    }

    printf("--INPUT DATA--\n");

    for (j=0; j<M; j++) {
        printf("x[%d][*] : ", j);
        for (i=0; i<n; i++) {
            printf("%8.2f ", x[j][i]);
        }
        printf("\n");
    }

    printf("Filter y : ");
    for (i=0; i<n ; i++) {
        printf("%8.2f ", y[i]);
    }

    ivr = 0;
    isw = 0;
    c_dvrcvf((double*)x, K, n, M, y, ivr, isw, tab, &icon);

    printf("\n\n--OUTPUT DATA--\n");
    for (j=0; j<M; j++) {
        printf("x[%d][*] : ", j);
        for (i=0; i<n; i++) {
            printf("%8.2f ", x[j][i]);
        }
        printf("\n");
    }
}
```

b) 非周期的データに対する畳み込みを計算する.

```
#include <stdio.h>
#include <stdlib.h>
#include "cssl.h" /* standard C-SSL header file */

#define K 16
#define M 3

int MAIN__(void)
{
    double x[M][K], y[K], tab[K*2];
    int    ivr, isw, icon;
    int    i, j, n, nx, ny;

    nx=7, ny=9, n=nx+ny-1;
    if(n%2) n=n+1;

    for (j=0; j<M; j++) {
        for (i=0; i<n; i++) {
            x[j][i] = (double)(i+j+1);
        }
    }
}
```

```

        for (i=nx; i<n; i++) {
            x[j][i] = 0.0;
        }
    }

    for (i=0; i<ny; i++) {
        y[i] = (double)(i+11);
    }
    for (i=ny; i<n; i++) {
        y[i] = 0.0;
    }

    printf("---INPUT DATA--\n");

    for (j=0; j<M; j++) {
        printf("x[%d][*] : ", j);
        for (i=0; i<n; i++) {
            if(i%8==0) printf("\n                ");
            printf("%.2f ", x[j][i]);
        }
        printf("\n");
    }

    printf("Filter y : ");
    for (i=0; i<n ; i++) {
        if(i%8==0) printf("\n                ");
        printf("%.2f ", y[i]);
    }

    ivr = 0;
    isw = 0;
    c_dvrcvf((double*)x, K, n, M, y, ivr, isw, tab, &icon);

    printf("\n\n--OUTPUT DATA--\n");
    for (j=0; j<M; j++) {
        printf("x[%d][*] : ", j);
        for (i=0; i<n; i++) {
            if(i%8==0) printf("\n                ");
            printf("%.2f ", x[j][i]);
        }
        printf("\n");
    }
}

```

c) 自己相関を計算する.

```

#include <stdio.h>
#include <stdlib.h>
#include "cssl.h" /* standard C-SSL header file */

#define K 8
#define M 3

int MAIN__(void)
{
    double x[M][K], tab[K*2];
    int    ivr, isw, icon;
    int    i, j, n, nx;

    nx=4, n=nx*2;

    for (j=0; j<M; j++) {
        for (i=0; i<nx; i++) {
            x[j][i] = (double)(i+j+1);
        }
        for (i=nx; i<n; i++) {
            x[j][i] = 0.0;
        }
    }

    printf("---INPUT DATA--\n");

    for (j=0; j<M; j++) {
        printf("x[%d][*] : ", j);
        for (i=0; i<n; i++) {
            printf("%.2f ", x[j][i]);
        }
    }
}

```

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

ivr = 1;
isw = 1;
c_dvrcvf((double*)x, K, n, M, (double*)x, ivr, isw, tab, &icon);

ivr = 3;
c_dvrcvf((double*)x, K, n, M, (double*)x, ivr, isw, tab, &icon);

printf("\n--OUTPUT DATA--\n");
for (j=0; j<M; j++) {
    printf("x[%d][*]  :", j);
    for (i=0; i<n; i++) {
        printf("%8.2f ", x[j][i]);
    }
    printf("\n");
}
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書”の VRCVF の項目を参照のこと。

c_dvrft1

離散型実フーリエ変換(2 基底 FFT)

```
ierr = c_dvrft1(a, n, isn, isw, vw, ivw,
                &icon);
```

1. 機能

1 次元 (項数 n) の実時系列データ $\{x_j\}$ が与えられたとき、離散型実フーリエ変換、又はその逆変換をベクトル計算機向きの高速変換手法 (FFT) により計算する。ただし、 $n = 2^\ell$ ($\ell: 0$ 又は正整数) であること。

フーリエ変換

$\{x_j\}$ を入力し、次の式で定義する変換を行い、フーリエ係数 $\{na_k\}$, $\{nb_k\}$ を求める。

$$na_k = 2 \cdot \sum_{j=0}^{n/2-1} x_j \cdot \cos(k j \theta), \quad k = 0, 1, \dots, n/2 - 1$$

$$nb_k = 2 \cdot \sum_{j=0}^{n/2-1} x_j \cdot \sin(k j \theta), \quad k = 1, 2, \dots, n/2 - 1$$

ここで $\theta = 2\pi/n$ 。

フーリエ逆変換

$\{a_k\}$, $\{b_k\}$ を入力し、次の式で定義する変換を行い、フーリエ級数の値 $\{2x_j\}$ を求める。

$$2x_j = a_0 + a_{n/2} \cdot \cos(\pi j) + 2 \cdot \sum_{k=1}^{n/2-1} (a_k \cdot \cos(k j \theta) + b_k \cdot \sin(k j \theta)), \quad j = 0, 1, \dots, n-1$$

ここで $\theta = 2\pi/n$ 。

2. 引数

呼出し形式:

```
ierr = c_dvrft1(a, n, isn, isw, vw, ivw, &icon);
```

引数の説明:

a	double a[n+2]	入力	$\{x_j\}$ 又は $\{a_k\}\{b_k\}$. データの格納方法については“使用上の注意” b) 参照のこと.
		出力	$\{na_k\}\{nb_k\}$ 又は $\{2x_j\}$
n	int	入力	変数の項数 n .
isn	int	入力	変換か逆変換かを指定する ($\neq 0$).
		変換	isn = 1
		逆変換	isn = -1
			(“使用上の注意” d) 参照)
isw	int	入力	変換の初期設定を制御する情報.
		0	初回呼出しのとき
		1	継続呼出しのとき

(“使用上の注意” c) 参照)

vw	double	作業領域	$Rlen = \max(n(\ell+1)/2, 1)$.
	vw[Rlen]		
ivw	int ivw[Ilen]	作業領域	$Ilen = n \cdot \max(\ell-4, 2)/2$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし	正常終了.
30000	次のいずれかであった. • isn = 0, • isw ≠ 0 又は 1 • $n \neq 2^\ell$ ($\ell: 0$ 又は正整数)	処理を打ち切る.

3. 使用上の注意

a) 本関数の特徴

本関数は, 実フーリエ変換をベクトル計算機上で高速処理する.

b) データの格納方法

行列 a	$\{x_j\}$	$\{a_k\},$ $\{b_k\}$
a[0]	x_0	a_0
a[1]	x_1	(注)
a[2]	x_2	a_1
a[3]	x_3	b_1
.	.	.
.	.	.
.	.	.
a[n-2]	x_{n-2}	$a_{n/2-1}$
a[n-1]	x_{n-1}	$b_{n/2-1}$
a[n]	(注)	$a_{n/2}$
a[n+1]	(注)	(注)

注) 入力時は任意, 出力時は 0.0 がセットされる.

c) isw による制御

複数列の変換を行う場合, 2 回目以降の呼出しのとき isw = 1 と指定すると, 変換に必要な三角関数表やリストベクトルなどの生成を省略するので, 効率良く処理される. このとき, 配列 vw, ivw の内容は変更せず呼び出すこと.

複数列の変換の項数 n が各々異なる場合でも, isw = 1 の指定は有効である. しかし, 可能な限り同一の項数の変換が連続するように呼び出すことが望ましい.

複素フーリエ変換の関数 c_dvcft1 と組み合わせて本関数を呼び出す場合でも, isw = 1 の指定は有効である.

d) isn の与えかた

isn では, 変換か逆変換かを指定するが, 更に次のように使い分けることができる. すなわち, $\{x_j\}$ 又は $\{a_k\}$, $\{b_k\}$ が, 共に間隔 i で格納されている場合は, 次のように指定する.

変換 : isn = +i

逆変換 : $isn = -i$

この場合, 計算結果も間隔 i で格納される.

ベクトル計算機で実行する場合, メモリアクセスを効率良くするために, 間隔 i は $i = 2 \cdot p + 1$, $p = 1, 2, 3, \dots$ であることが望ましい.

e) 作業領域の大きさ早見表

$16 \leq n \leq 4096$ の場合の大きさを以下に示す.

ℓ	n	vw の大きさ	ivw の大きさ
4	16	40	16
5	32	96	32
6	64	224	64
7	128	512	192
8	256	1152	512
9	512	2560	1280
10	1024	5632	3072
11	2048	12288	7168
12	4096	26624	16384

f) 一般的なフーリエ変換の定義について

離散型実フーリエ変換及び, 逆変換は一般的に (1), (2) で定義される.

$$a_k = \frac{2}{n} \cdot \sum_{j=0}^{n-1} x_j \cdot \cos(k j \theta), \quad k = 0, 1, \dots, n/2$$

$$b_k = \frac{2}{n} \cdot \sum_{j=0}^{n-1} x_j \cdot \sin(k j \theta), \quad k = 1, 2, \dots, n/2 - 1, \quad (1)$$

$$x_j = \frac{1}{2} a_0 + \frac{1}{2} a_{n/2} \cdot \cos(\pi j) + \sum_{k=0}^{n/2-1} (a_k \cdot \cos(k j \theta) + b_k \cdot \sin(k j \theta)), j = 0, 1, \dots, n-1, \quad (2)$$

ここで, $\theta = 2\pi/n$.

本関数では(1), (2)の左辺に対応して $\{na_k\}$, $\{nb_k\}$ 又は $\{2x_j\}$ を求める. したがって, 結果の正規化は必要に応じで行うこと.

4. 使用例

1次元フーリエ変換を求める. 入力は乱数で与える. 変換後, さらに逆変換して元の値と結果を確認する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 1024

MAIN__()
{
    int ierr, icon;
    double phai, ran, eps;
    double a[NMAX+2], b[NMAX+2], vw[NMAX*(10+1)/2];
    int i, n, isw, isn, ivw[NMAX*(10-4)/2];

    /* generate initial data */
    n = NMAX;
    phai = (sqrt(5.0)-1.0)/2;
```

```
for (i=0;i<n;i++) {
    ran = (i+1)*phai;
    a[i] = ran - (int)ran;
}
for (i=0;i<n;i++)
    b[i] = a[i];
/* perform normal transform */
isw = 0;
isn = 1;
ierr = c_dvrft1(a, n, isn, isw, vw, ivw, &icon);
/* perform inverse transform */
isw = 1;
isn = -1;
ierr = c_dvrft1(a, n, isn, isw, vw, ivw, &icon);
/* check results */
eps = 1e-6;
for (i=0;i<n;i++)
    if (fabs((a[i]/(2*n) - b[i])/b[i]) > eps) {
        printf("WARNING: result inaccurate\n");
        exit(1);
    }
printf("Result OK\n");
return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書”の VCFT1 及び VRFT1 の項目を参照のこと。

c_dvrft2

離散型実フーリエ変換 (メモリ節約型, 2 基底 FFT)

```
ierr = c_dvrft2(a, n, isn, isw, vw, ivw,
                &icon);
```

1. 機能

1 次元(項数 n)の実時系列データ $\{x_j\}$ が与えられたとき, 離散型実フーリエ変換, 又はその逆変換をベクトル計算機向き的高速変換手法(FFT)により計算する. ただし, $n=2^l$ ($l: 0$ 又は正整数)であること.

フーリエ変換

$\{x_j\}$ を入力し, 次の式で定義する変換を行い, フーリエ係数 $\{na_k\}$, $\{nb_k\}$ を求める.

$$na_k = 2 \sum_{j=0}^{n-1} x_j \cos(kj\theta), \quad k=0,1,\dots,n/2, \quad \theta=2\pi/n,$$

$$nb_k = 2 \sum_{j=0}^{n-1} x_j \sin(kj\theta), \quad k=1,\dots,n/2-1, \quad \theta=2\pi/n$$

フーリエ逆変換

$\{a_k\}$, $\{b_k\}$ を入力し, (1.2)で定義する変換を行い, フーリエ級数の値 $\{2x_j\}$ を求める.

$$2x_j = a_0 + 2 \sum_{k=1}^{n/2-1} [a_k \cos(kj\theta) + b_k \sin(kj\theta)] + a_{n/2} \cos(\pi j), \quad j=0,1,\dots,n-1, \quad \theta=2\pi/n$$

2. 引数

呼出し形式:

```
ierr = c_dvrft2(a, n, isn, isw, vw, ivw, &icon);
```

引数の説明:

a	double a[n+2]	入力	$\{x_j\}$ 又は $\{a_k\}$, $\{b_k\}$. (“使用上の注意”b)参照)
		出力	$\{na_k\}$, $\{nb_k\}$ 又は $\{2x_j\}$.
n	int	入力	変換の項数 n .
isn	int	入力	変換か逆変換かを指定する ($isn \neq 0$). $isn=1$ のとき変換. $isn=-1$ のとき逆変換. (“使用上の注意”c)参照)
isw	int	入力	変換の初期設定を制御する情報. $isw=0$ のとき初回呼出し. $isw=1$ のとき継続呼出し. (“使用上の注意”d)参照)
vw	double vw[7n/2]	作業領域	
ivw	int ivw[3n/2]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	次のいずれかであった. • $isn = 0$ • $isw \neq 0, 1$ • $n \neq 2^l$ ($l: 0$ 又は正整数)	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

本関数は、ベクトル計算機用のもう一つの関数 c_dvrft1 と同等の機能を持つ。本関数は、単独の変換を行う場合に適している。作業用配列は、必要最小限におさえてあり、言わば、「メモリ節約型」の関数である。複数組の変換を行う場合で、作業用配列を十分に確保できるならば、「性能優先型」の関数 c_dvrft1 が適している。

b) データの格納方法

配列 a	$\{x_j\}$	$\{a_k\},$ $\{b_k\}$
a[0]	x_0	a_0
a[1]	x_1	(注)
a[2]	x_2	a_1
a[3]	x_3	b_1
.	.	.
.	.	.
.	.	.
a[n-2]	x_{n-2}	$a_{n/2-1}$
a[n-1]	x_{n-1}	$b_{n/2-1}$
a[n]	(注)	$a_{n/2}$
a[n+1]	(注)	(注)

注) 入力時は任意, 出力時は 0.0 がセットされる。

c) isn

isn では、変換か逆変換かを指定するが、更に次のように使い分けることができる。すなわち、 $\{x_j\}$ 又は $\{a_k\}$ 、 $\{b_k\}$ が、それぞれ間隔 i で格納されている場合は、次のように指定する。

変換 : $isn = +i$

逆変換: $isn = -i$

この場合、計算結果も間隔 i で格納される。

ベクトル計算機で実行する場合、メモリアクセスを効率良くするために、間隔 i は $i = 2p + 1, p = 1, 2, 3, \dots$ のような値であることが望ましい。

d) isw

複数組の変換を行う場合、2 回目以降の呼出しのとき $isw = 1$ と指定すると、変換に必要な三角関数表やリストベクトルなどの生成を省略するので、多少効率良く処理される。このとき、配列 vw, ivw の内容は変更せず呼び出すこと。複数組の変換の項数 n が各々異なる場合でも、 $isw = 1$ の指定は有効である。しかし、可能な限り同一の項数の変換が連続するように呼び出すことが望ましい。

複素フーリエ変換の関数 c_dvcft2 と組み合わせて本関数を呼び出す場合でも、 $isw = 1$ の指定は有効である。

e) 作業用配列の大きさ早見表

$16 \leq n \leq 4096$ の場合の大きさを以下に示す.

l	n	vw	ivw
4	16	56	24
5	32	112	48
6	64	224	96
7	128	448	192
8	256	896	384
9	512	1792	768
10	1024	3584	1536
11	2048	7168	3072
12	4096	14336	6144

f) 一般的なフーリエ変換の定義について

離散型実フーリエ変換及び、逆変換は一般的に(1), (2)で定義される

$$a_k = \frac{2}{n} \sum_{j=0}^{n-1} x_j \cos(kj\theta), \quad k = 0, 1, \dots, n/2, \quad \theta = 2\pi/n \quad (1)$$

$$b_k = \frac{2}{n} \sum_{j=0}^{n-1} x_j \sin(kj\theta), \quad k = 1, \dots, n/2, \quad \theta = 2\pi/n$$

$$x_j = \frac{1}{2} a_0 + \sum_{k=1}^{n/2-1} [a_k \cos(kj\theta) + b_k \sin(kj\theta)] + \frac{1}{2} a_{n/2} \cos(\pi j), \quad j = 0, 1, \dots, n-1, \quad \theta = 2\pi/n \quad (2)$$

本関数では, (1), (2)の左辺に対応して $\{na_k\}$, $\{nb_k\}$ 又は $\{2x_j\}$ を求める. したがって, 結果の正規化は必要に応じて行うこと.

4. 使用例

1次元のフーリエ変換及びその逆変換連続して計算する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 1024

MAIN__()
{
    int ierr, icon;
    double phai, ran, eps;
    double a[NMAX+2], b[NMAX+2], vw[7*NMAX/2];
    int i, n, isw, isn, ivw[3*NMAX/2];

    /* generate initial data */
    n = NMAX;
    phai = (sqrt(5.0)-1.0)/2;
    for (i=0; i<n; i++) {
        ran = (i+1)*phai;
        a[i] = ran - (int)ran;
    }
    for (i=0; i<n; i++)
        b[i] = a[i];
}
```

```
/* perform normal transform */
isw = 0;
isn = 1;
ierr = c_dvrft2(a, n, isn, isw, vw, ivw, &ierr);
/* perform inverse transform */
isw = 1;
isn = -1;
ierr = c_dvrft2(a, n, isn, isw, vw, ivw, &ierr);
/* check results */
eps = 1e-6;
for (i=0;i<n;i++)
    if (fabs((a[i]/(2*n) - b[i])/b[i]) > eps) {
        printf("WARNING: result inaccurate\n");
        exit(1);
    }
printf("Result OK\n");
return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書”の VRFT2 の項目を参照のこと。

c_dvrpf3

3次元素因子離散型実フーリエ変換

ierr = c_dvrpf3(a, l, m, n, isn, vw, &icon);

1. 機能

3次元の実時系列データ $\{x_{j_1 j_2 j_3}\}$ が与えられたとき、離散型実フーリエ変換、又はその逆変換を素因子フーリエ変換 (a prime factor FFT) により行う。各元の項数は、次の条件を満たす数である。

- 次の数の中で互いに素な因子の積で表せること。
因子 p ($p \in \{2, 3, 4, 5, 7, 8, 9, 16\}$)
- 1次元目の項数は偶数 $2 \times i$ であり、 i は上の条件を満たす。

3次元実フーリエ変換

$\{x_{j_1 j_2 j_3}\}$ を入力し、(1)で定義する変換を行い、 $\{n_1 n_2 n_3 \alpha_{k_1 k_2 k_3}\}$ を求める。

$$n_1 n_2 n_3 \alpha_{k_1 k_2 k_3} = \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} \sum_{j_3=0}^{n_3-1} x_{j_1 j_2 j_3} \omega_1^{-j_1 k_1} \omega_2^{-j_2 k_2} \omega_3^{-j_3 k_3}, \quad (1)$$

$$k_r = 0, \dots, n_r - 1,$$

$$\omega_r = \exp(2\pi i / n_r), \quad r = 1, 2, 3$$

3次元実フーリエ変換の場合、 $\{x_{j_1 j_2 j_3}\}$ が実数なので約半分の計算を行う。つまり1次元についての k_1 は0から $n_1/2$ までの計算を行う。

3次元実フーリエ逆変換

3次元フーリエ逆変換は $\{\alpha_{k_1 k_2 k_3}\}$ を入力し、(2)で定義する変換を行い、 $\{x_{j_1 j_2 j_3}\}$ を求める。

$$x_{j_1 j_2 j_3} = \sum_{k_1=0}^{n_1-1} \sum_{k_2=0}^{n_2-1} \sum_{k_3=0}^{n_3-1} \alpha_{k_1 k_2 k_3} \omega_1^{j_1 k_1} \omega_2^{j_2 k_2} \omega_3^{j_3 k_3}, \quad (2)$$

$$j_r = 0, \dots, n_r - 1,$$

$$\omega_r = \exp(2\pi i / n_r), \quad r = 1, 2, 3$$

3次元実フーリエ逆変換では、 $\{\alpha_{k_1 k_2 k_3}\}$ は $k_1=0, 1, \dots, n_1/2$ しか求めていないが、2次元、3次元の $\sum$ を先に計算して、1次元目の要素に関する共役関係を利用して(2)を計算する。

2. 引数

呼出し形式:

```
ierr = c_dvrpf3((double *) a, l, m, n, isn, vw, &icon);
```

引数の説明:

a	double a[n][m][l]	入力	isn ≥ 0 のとき実数 $\{x_{j_1 j_2 j_3}\}$. isn < 0 のとき、フーリエ変換された複素数 $\{\alpha_{k_1 k_2 k_3}\}$. ("使用上の注意" a)参照)
		出力	isn ≤ 0 のとき、フーリエ変換した複素数 $\{n_1 n_2 n_3 \alpha_{k_1 k_2 k_3}\}$. isn < 0 のとき、逆変換された実数 $\{x_{j_1 j_2 j_3}\}$. ("使用上の注意" a)参照)
l	int	入力	1次元のデータ数 $n_1 + 2$. 配列 a の3次元のサイズとなる。偶数でなければならない。 $(l-2)/2 \leq 5040$.
m	int	入力	2次元のデータ数 n_2 . 配列 a の2次元のサイズとなる。 $m \leq 5040$.

n	int	入力	3次元のデータ数 n_3 . 配列 a の 1 次元のサイズとなる. $n \leq 5040$.
isn	int	入力	制御情報. $isn \geq 0$ (非負の整数)なら変換. $isn < 0$ (負の整数)なら逆変換.
vw	double vw[n*m*l]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	次のいずれかであった. $(l-2)/2, m, n$ のいずれかが 5040 を超えた. $(l-2)/2, m, n$ のいずれかが $\{2, 3, 4, 5, 7, 8, 9, 16\}$ の中の互いに素な因子の積に分解できなかった.	処理を打ち切る.
30000	次のいずれかであった. $l-2$ が偶数でなかった. l, m, n のいずれかが 0 か負であった.	処理を打ち切る.

3. 使用上の注意

a) データの格納方法

実データ (変換の場合の入力及び逆変換の場合の出力)は, 配列 a に以下のように格納される.

$$a[j_3][j_2][j_1] = x_{j_1 j_2 j_3}, \quad j_i = 0, 1, \dots, n_i - 1, \quad i = 1, 2, 3.$$

複素データ (変換の場合の出力及び逆変換の場合の入力)は, 配列 a に実部と虚部が半分ずつ格納される.

$$\begin{aligned} a[k_3][k_2][k_1] &= \operatorname{Re}(\alpha_{k_1 k_2 k_3}) \text{ 又は } \operatorname{Re}(n_1 n_2 n_3 \alpha_{k_1 k_2 k_3}), & k_1 &= 0, 1, \dots, \operatorname{floor}(n_1 / 2), \\ a[k_3+1+n_1/2][k_2][k_1] &= \operatorname{Im}(\alpha_{k_1 k_2 k_3}) \text{ 又は } \operatorname{Im}(n_1 n_2 n_3 \alpha_{k_1 k_2 k_3}), & k_i &= 0, 1, \dots, n_i - 1, \quad i = 2, 3. \end{aligned}$$

b) 項数及び次元数について

項数は次の数の中で互いに素な因子 p ($p \in \{2, 3, 4, 5, 7, 8, 9, 16\}$) の積で表せる数であり, 最大値は $5 \times 7 \times 9 \times 16 = 5040$ である.

引数 n に 1 を設定して呼び出すと 2 次元実素因子高速フーリエ変換になる. さらに, 引数 n に 1 を, そして引数 m に 1 を設定して呼び出すと 1 次元実素因子高速フーリエ変換になる.

c) 一般的な 3 次元フーリエ変換の定義について

3 次元フーリエ変換及び, 逆変換は一般的に (3), (4) で定義される.

$$\alpha_{k_1 k_2 k_3} = \frac{1}{n_1 n_2 n_3} \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} \sum_{j_3=0}^{n_3-1} x_{j_1 j_2 j_3} \omega_1^{-j_1 k_1} \omega_2^{-j_2 k_2} \omega_3^{-j_3 k_3}, \quad (3)$$

$$k_r = 0, \dots, n_r - 1,$$

$$\omega_r = \exp(2\pi i / n_r), r = 1, 2, 3$$

$$x_{j_1 j_2 j_3} = \sum_{k_1=0}^{n_1-1} \sum_{k_2=0}^{n_2-1} \sum_{k_3=0}^{n_3-1} \alpha_{k_1 k_2 k_3} \omega_1^{j_1 k_1} \omega_2^{j_2 k_2} \omega_3^{j_3 k_3}, \quad (4)$$

$$j_r = 0, \dots, n_r - 1,$$

$$\omega_r = \exp(2\pi i / n_r), r = 1, 2, 3$$

本関数では, (3), (4)の左辺に対応して $\{n_1 n_2 n_3 \alpha_{k_1 k_2 k_3}\}$ 又は $\{x_{j_1 j_2 j_3}\}$ を求めるので結果の正規化は必要に応じて行うこと. 本関数により変換・逆変換を正規化せずに連続して実行すると, 入力データの各要素は $n_1 n_2 n_3$ 倍されて出力される.

4. 使用例

実時系列データをフーリエ変換し, その結果をフーリエ逆変換して, もとのデータと比較する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

/* problem dimensions */
#define N1 4
#define N2 3
#define N3 2

MAIN__()
{
    int ierr, icon;
    double phai, ran, eps;
    double a[N3][N2][N1+2]; /* allocate real data */
    double (*b)[2][N3][N2][(N1+2)/2]; /* pointer to complex data */
    double aa[N3][N2][N1+2], vw[N3][N2][N1+2];
    int i, j, k, cnt, l, m, n, isn, pr;

    /* generate initial real data */
    l = N1+2;
    m = N2;
    n = N3;
    pr = (l-2)*m*n;
    phai = (sqrt(5.0)-1.0)/2;
    cnt = 1;
    for (k=0; k<N3; k++) {
        for (j=0; j<N2; j++) {
            for (i=0; i<N1; i++) {
                ran = cnt*phai;
                a[k][j][i] = ran - (int)ran;
                cnt++;
            }
        }
    }
    /* keep copy */
    for (k=0; k<N3; k++) {
        for (j=0; j<N2; j++) {
            for (i=0; i<N1; i++)
                aa[k][j][i] = a[k][j][i];
        }
    }
    /* perform normal transform */
    isn = 1;
    ierr = c_dvrpf3((double*)a, l, m, n, isn, (double*)vw, &icon);
    /* print complex transformed data */
    b = (double(*)[2][N3][N2][(N1+2)/2])a; /* complex data overwrites real data */
    for (k=0; k<N3; k++) {
        for (j=0; j<N2; j++) {
            for (i=0; i<N1/2; i++) {
                printf("%8.5f + i*%8.5f ", (*b)[0][k][j][i], (*b)[1][k][j][i]);
            }
        }
    }
}
```

```
    }
    printf("\n");
  }
  printf("\n");
}
/* perform inverse transform */
isn = -1;
ierr = c_dvrpf3((double*)a, l, m, n, isn, (double*)vw, &icon);
/* check results */
eps = 1e-6;
for (k=0;k<N3;k++) {
  for (j=0;j<N2;j++) {
    for (i=0;i<N1;i++) {
      if (fabs((a[k][j][i]/pr - aa[k][j][i])/aa[k][j][i]) > eps) {
        printf("WARNING: result inaccurate\n");
        exit(1);
      }
    }
  }
}
printf("Result OK\n");
return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書 II”の VRPF3 の項目及び[17], [120]を参照のこと.

c_dvseg2

実対称行列の固有値及び固有ベクトル(並列バイセクション法, 逆反復法)

```
ierr = c_dvseg2(a, n, m, epst, e, ev, k, vw,
                ivw, &icon);
```

1. 機能

n 次の実対称行列 A の固有値問題,

$$Ax = \lambda x$$

に対する固有値を, 並列バイセクション法により, 大きい方から (又は, 小さい方から) m 個求める. また, 対応する m 個の固有ベクトルを逆反復法により求める. 固有ベクトルは, $\|x\|_2 = 1$ と なるように正規化する. ただし, $1 \leq m \leq n$ であること.

2. 引数

呼出し形式:

```
ierr = c_dvseg2(a, n, m, epst, e, (double *)ev, k, vw, ivw, &icon);
```

引数の説明:

a	double a[Alen]	入力	実対称行列 A . 対称行列用圧縮格納法. $Alen = n(n+1)/2$. 格納方法の詳細については“1.3.2 対称行列の格納方法”を参照のこと.
n	int	出力	演算後, 内容は保存されない.
m	int	入力	実対称行列 A の次数 n .
		入力	求める固有値の数 m . $m = +m$ のときは大きい方から求める. $m = -m$ のときは小さい方から求める.
epst	double	入力	固有値の収束判定に使われる絶対誤差の上限. 負の値を指定すると標準値が採用される. (“使用上の注意” a) 参照)
e	double e[m]	出力	固有値. m が正の場合大きい順に, m が負の場合小さい順に格納される.
ev	double ev[m][k]	出力	固有ベクトル. 固有値 $e[i]$ に対応する固有ベクトルは, $ev[i][j]$, $j = 0, 1, \dots, n-1$ に格納される.
k	int	入力	配列 ev の整合寸法 ($\geq n$).
vw	double vw[15*n]	作業領域	
ivw	int ivw[7*n]	作業領域	
icon	int	出力	コンディションコード. 下の表参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	$n = 1$ であった.	$ev[0][0] = 1.0, e[0] = a[0]$ とする.
15000	固有ベクトルの計算で求まらないものがあった.	求まらない固有ベクトルを零ベクトルとする.
20000	固有ベクトルはすべて求まらなかった.	固有ベクトルはすべて零ベクトルとする.
30000	次のいずれかであった. $m = 0$ $n < m$ $k < n$	処理を打ち切る.

3. 使用上の注意

a) epst

引数 epst の標準値は、丸め誤差の単位を μ としたとき、

$$epst = \mu \cdot \max(|\lambda_{\max}|, |\lambda_{\min}|) \quad (1)$$

である. ここで $\lambda_{\max}$ 及び $\lambda_{\min}$ は, $Ax = \lambda x$ の固有値の存在範囲 (ゲルシュゴリンの定理により与えられる) の上限と下限である.

固有値の絶対値が非常に大きいものと、非常に小さいものとが混在しているとき(1)により収束判定すると、一般に小さい方の固有値を精度よく求めることは困難である. このような場合は、epst に小さな値 (絶対誤差) を設定することにより、小さい方の固有値も精度よく求めることができる. ただし、反復回数が増加するため処理速度は遅くなる.

収束判定方法については詳細については“FUJITSU SSL II 拡張機能使用手引書”の VSEG2 の項目を参照のこと.

4. 使用例

5×5 の対称行列の固有値及び固有ベクトルを求める.

```
#include <stdlib.h>
#include <stdio.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 5

MAIN__()
{
    int ierr, icon;
    int n, m, i, j, ij, k, ivw[7*NMAX];
    double a[NMAX*(NMAX+1)/2], e[NMAX], ev[NMAX][NMAX], vw[15*NMAX], epst;

    /* initialize matrix */
    n = NMAX;
    ij = 0;
    for (i=0; i<n; i++)
        for (j=0; j<=i; j++) {
            a[ij++] = n-i;
        }
    k = NMAX;
    m = n;
    epst = 0;
    /* find eigenvalues and eigenvectors */
    ierr = c_dvseg2(a, n, m, epst, e, (double*)ev, k, vw, ivw, &icon);
    if (icon >= 20000) {
```

```
        printf("ERROR: c_dvseg2 failed with icon = %d\n", icon);
        exit(1);
    }
    /* print eigenvalues and eigenvectors */
    for (i=0;i<m;i++) {
        printf("e-value %d: %10.4f\n",i+1,e[i]);
        printf("e-vector:");
        for (j=0;j<n;j++)
            printf("%7.4f ",ev[i][j]);
        printf("\n");
    }
    return(0);
}
```

5. 手法概要

本関数では n 次の実対称行列 $\mathbf{A}$ の固有値を, 並列バイセクション法により, 大きい方から(又は, 小さい方から) m 個求め, 対応する固有ベクトルを逆反復法により求める.

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書”の VSEG2 の項目を参照のこと.

c_dvsevp

実対称行列の固有値及び固有ベクトル (三重対角化, マルチセクション法, 逆反復法)

<pre>ierr = c_dvsevp(a, k, n, nf, nl, ivec, &etol, &ctol, nev, e, maxne, m, ev, vw, iw, &icon);</pre>

1. 機能

n 次の実対称行列 $\mathbf{A}$ の指定されたいくつかの固有値を求める. 指定に応じて, 固有ベクトルを求める.

実対称行列 $\mathbf{A}$ を Householder 変換で三重対角化し, マルチセクション法により固有値を求める. 固有ベクトルは逆反復法で求める.

$$\mathbf{Ax} = \lambda \mathbf{x}$$

2. 引数

呼出し形式:

```
ierr = c_dvsevp((double *)a, k, n, nf, nl, ivec, &etol, &ctol, nev, e, maxne,
                (int *)m, (double *)ev, vw, iw, &icon);
```

引数の説明:

a	double a[n][k]	入力	固有値・固有ベクトルを求める実対称行列 $\mathbf{A}$ を格納する.
k	int	入力	配列 ar, ai の整合寸法 ($k \geq n$).
n	int	入力	実対称行列 $\mathbf{A}$ の次数 n .
nf	int	入力	固有値を小さいほうから番号付けして(多重固有値には多重度分番号を割り当てる), 求める最初の固有値の番号.
nl	int	入力	固有値を小さいほうから番号付けして(多重固有値には多重度分番号を割り当てる), 求める最後の固有値の番号.
ivec	int	入力	制御情報. ivec = 1 のとき固有値及び固有ベクトルの両方を求める. ivec $\neq$ 1 のとき固有値のみ求める.
etol	double	入力 出力	固有値が数値的に異なるか多重かを判定するための判定値. etol $< 3 \times 10^{-16}$ のときは, etol = 3×10^{-16} が標準値として設定される. (“使用上の注意”a)参照)
ctol	double	入力 出力	近接している固有値が近似的に多重かを判定するための判定値 ($\geq$ etol). ctol $<$ etol のときは, ctol = etol が標準値として設定される. (“使用上の注意”a)参照)
nev	int nev[3]	出力	求められた固有値の個数に関する情報. nev[0] は, 異なる固有値の個数. nev[1] は, 異なる近似的多重固有値(cluster)の個数. nev[2] は, 多重度も含んだ全ての固有値の個数.
e	double e[maxne]	出力	固有値が格納される. 求められた固有値は e[i-1], $i = 1, \dots, nev[2]$ に格納される.
maxne	int	入力	計算できる固有値の最大個数. 配列 e の大きさ. (“使用上の注意”b)参照)
m	int m[2][maxne]	出力	求められた固有値の多重度に関する情報. m[0][i-1] は i 番目の固有値 λ_i の多重度を, m[1][i-1] は i

			番目の固有値 λ_i に関して、近接している固有値を近似的多重固有値(cluster)と見なしたときの多重度を示す。
ev	double ev[maxne][k]	出力	ivec = 1 のとき、おのこの固有値に対応して固有ベクトルが格納される。 求められた固有ベクトルは ev[i-1][j-1], i = 1,...,nev[2], j = 1,...,n に格納される。
vw	double vw[17*n]	作業領域	
iw	int iw[lwlen]	作業領域	lwlen = 9*k+128.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	密集固有値の計算中, 固有値の総数が maxne を超えた.	処理を打ち切る. 固有ベクトルを求めることはできないが, 異なる固有値自体は求められている. (“使用上の注意”b)参照)
30000	次のいずれかであった. • $n < 1$ • $k < n$ • $nf < 1$ • $nl > n$ • $nl < nf$ • $maxne < nl - nf + 1$	処理を打ち切る.
30100	入力行列が対称行列でない可能性がある.	

3. 使用上の注意

a) etol 及び ctol

本ルーチンは、多重固有値(cluster)に対して特別の考慮を払っている。

$\varepsilon = \text{etol}$ としたとき、連続する固有値 λ_j , $j = s, s+1, \dots, s+k$, ($k \geq 0$) に対して、

$$\frac{|\lambda_i - \lambda_{i-1}|}{1 + \max(|\lambda_{i-1}|, |\lambda_i|)} \leq \varepsilon, \quad (1)$$

$i = s, s+1, \dots, s+k$ となる i に対して(1)を満たし、 $i = s-1$ 及び $i = s+k+1$ について(1)を満たさないとき、これらの固有値 λ_j , $j = s-1, s, \dots, s+k$ は数値的に多重であると見なされる。

etol の標準値は 3×10^{-16} ～丸め誤差の単位であり、このとき固有値は計算機で求め得る精度まで分離される。

(1)が $\varepsilon = \text{etol}$ に対して成立しないとき、 λ_{i-1} 及び λ_i は異なる固有値と考える。

$\varepsilon = \text{etol}$ で異なる固有値と見なされた連続する固有値 λ_m , $m = t-1, t, \dots, t+k$, ($k \geq 0$) に対して、 $\varepsilon = \text{ctol}$ としたとき、 $i = t, t+1, \dots, t+k$ について(1)を満たし、 $i = t-1$ 及び $i = t+k+1$ について(1)を満たさないとき、これらの異なる固有値 λ_m , $m = t-1, t, \dots, t+k$ を近似的に多重(cluster)と見なす。これは、cluster に対応する不変部分空間、つまり 1 次独立な固有ベクトルを計算するために利用される。つまり、対応する固有ベクトルは直交する初期ベクトルを用いて計算され、再直交化される。もちろん、 $\text{ctol} \geq \text{etol}$ を満たさなければならない。この条件を満たさないとき、ctol は etol の値と等しく設定される。

b) maxne

r 個数の固有値を求めるとき、最初又は最後の固有値が、1 より大きい多重度を持つとき、 r 個以上の固有値が求まる。対応する固有ベクトルは増えた固有値も含んで、対応する固有ベクトルの格納域が足りるときのみ計算される。maxne に、計算できる固有値の最大個数を指定できる。計算する固有値の総数が maxne を超えた場合、icon=20000 を返却する。このとき、固有ベクトルの計算を行うよう指定されていたら固有ベクトルの計算はできない。固有値は求められているが、多重度分だけ繰り返して格納はされていない。つまり、固有値に関しては、求められた異なる固有値が $e[i-1]$, $i=1,...,nev[0]$ に、および対応する固有値の多重度が $m[0][i-1]$, $i=1,...,nev[0]$ にそれぞれ格納されている。固有値がすべて異なり、近似的多重な固有値もない場合、maxne は入力として nev[0] に指定する求める固有値の総数で十分である。計算する固有値の総数が maxne を超えた場合、計算を続けるために必要な maxne の値は nev[2] に返却される。

4. 使用例

実対称行列の固有値を計算する。

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define K          500
#define N          K
#define NF         1
#define NL         100
#define MAXNE      NL-NF+1
#define NVW        17*K
#define NIW        9*MAXNE+128

MAIN__()
{
    double a[N][K], ab[N][K];
    double e[MAXNE], ev[MAXNE][K], vw[NVW];
    double vv[N][K];
    double etol, ctol, pi;
    int    nev[3], m[2][MAXNE], iw[NIW];
    int    ierr, icon;
    int    i, j, k, n, nf, nl, maxne, ivec;

    n      = N;
    k      = K;
    nf     = NF;
    nl     = NL;
    ivec   = 1;
    maxne  = MAXNE;
    etol   = 3.0e-16;
    ctol   = 5.0e-12;

    /* Generate real symmetric matrix with known eigenvalues */
    /* Initialization */
    pi = 4.0 * atan(1.0);
    for(i=0; i<n; i++) {
        for(j=0; j<n; j++) {
            vv[i][j] = sqrt(2.0/(double)(n+1))*sin((double)(i+1)*pi*
                (double)(j+1)/(double)(n+1));
            a[i][j] = 0.0;
        }
    }

    for(i=0; i<n; i++) {
        a[i][i] = (double)(-n/2+(i+1));
    }

    printf(" Input matrix size is %d\n", n);
    printf(" Matrix calculations use k = %d\n", k);
    printf(" Desired eigenvalues are nf to nl %d %d\n", nf, nl);
    printf(" That is, request %d eigenvalues.\n", maxne);
    printf(" True eigenvalues are as follows\n");
    for(i=nf-1; i<nl; i++) {
        printf("a(%d,%d) = %12.4e\n", i, i, a[i][i]);
    }
}
```

```
    }

    ierr = c_dvmggm ((double*)a, k, (double*)vv, k, (double*)ab, k, n, n, n, &icon);
    ierr = c_dvmggm ((double*)vv, k, (double*)ab, k, (double*)a, k, n, n, n, &icon);

    /* Calculate the eigendecomposition of A */
    ierr = c_dvsevp ((double*)a, k, n, nf, nl, ivec, &etol, &ctol, nev, e, maxne,
                    (int*)m, (double*)ev, vw, iw, &icon);
    if (icon > 0) {
        printf("ERROR: c_dvsevp failed with icon = %d\n", icon);
        exit(1);
    }
    printf("icon = %i\n", icon);
    /* print eigenvalues */
    printf(" Number of eigenvalues %d\n", nev[2]);
    printf(" Number of distinct eigenvalues %d\n", nev[0]);
    printf(" Solution to eigenvalues\n");
    for(i=0; i<nev[2]; i++) {
        printf("  e[%d] = %12.4e\n", i, e[i]);
    }
    return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書 II”の VSEVP の項目及び[81]を参照のこと。

c_dvsin1

離散型 sine 変換 (2 基底 FFT)

ierr = c_dvsin1(b, n, tab, vw, ivw, &icon);

1. 機能

周期 2π の奇関数 $x(t)$ の半周期を n 等分した n 個の標本 $\{x_j\}$

$$x_j = x(j \cdot \theta), \quad j = 0, 1, \dots, n-1, \quad \theta = \pi / n$$

が与えられたとき, 離散型 sine 変換, 又はその逆変換をベクトル計算向き的高速変換手法(FFT)により計算する. ただし, $n = 2^\ell$ (ℓ : 正整数)であること.

sine 変換

$\{x_j\}$ を入力し, 次の式で定義する変換を行い, フーリエ係数 $\{2nb_k\}$ を求める.

$$2nb_k = 4 \cdot \sum_{j=0}^{n-1} x_j \cdot \sin(k j \theta), \quad k = 0, 1, \dots, n-1$$

ここで, $\theta = \pi / n$, $x_0 = 0$.

sine 逆変換

$\{b_k\}$ を入力し, 次の式で定義する変換を行い, フーリエ級数の値 $\{4x_j\}$ を求める.

$$4x_j = 4 \cdot \sum_{k=0}^{n-1} b_k \cdot \sin(k j \theta), \quad j = 0, 1, \dots, n-1$$

ここで $\theta = \pi / n$, $b_0 = 0$.

2. 引数

呼出し形式:

```
ierr = c_dvsin1(b, n, tab, vw, ivw, &icon);
```

引数の説明:

b	double b[n+2]	入力	$\{x_j\}$ 又は $\{b_k\}$. $x_1, x_2, \dots, x_{n-1}$ を b[1], b[2], ..., b[n-1] に, 又は $b_1, b_2, \dots, b_{n-1}$ を b[1], b[2], ..., b[n-1] に格納する. b[0], b[n] 及び b[n+1] は任意.
		出力	$\{2nb_k\}$ 又は $\{4x_j\}$. b[0], b[n] 及び b[n+1] には 0.0 がセットされる.
n	int	入力	標本の数 n .
tab	double tab[Tlen]	出力	変換で使用された三角関数表が格納される. $Tlen = 2n+4$.
vw	double vw[Rlen]	作業領域	$Rlen = \max(n(\ell+1)/2, 1)$.
ivw	int ivw[Ilen]	作業領域	$Ilen = n \cdot \max(\ell-4, 2)/2$.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	$n \neq 2^\ell$ (ℓ : 正整数)	処理を打ち切る.

3. 使用上の注意

a) 本関数の使用方法

本関数は, 離散型 sine 変換をベクトル計算機上で高速処理する.

b) 複数组の変換について

複数组の変換を行う場合, 2 回目以降の呼出しのときは, 変換に必要な三角関数表やリストベクトルなどの生成を省略するので, 効率良く処理される. このとき, 配列 `tab`, `vw`, `ivw` の内容は変更せず呼び出すこと.

複数组の変換の項数 n が各々異なる場合でも, それまでに生成した, 配列 `tab`, `vw`, `ivw` の内容は有効である. しかし, 可能な限り同一の項数の変換が連続するように呼び出すことが望ましい.

c) 三角関数表及び作業領域の大きさ早見表

$16 \leq n \leq 4096$ の場合の大きさを以下に示す.

ℓ	n	tab	vw	ivw
4	16	36	40	16
5	32	68	96	32
6	64	132	224	64
7	128	260	512	192
8	256	516	1152	512
9	512	1028	2560	1280
10	1024	2052	5632	3072
11	2048	4100	12288	7168
12	4096	8196	26624	16384

d) 一般的な離散型 sine 変換

離散型 sine 変換及び, 逆変換は一般的に(1), (2) で定義される.

$$b_k = \frac{2}{n} \cdot \sum_{j=1}^{n-1} x_j \cdot \sin(k \cdot j \cdot \theta), \quad k = 0, 1, \dots, n-1, \quad (1)$$

$$x_j = \sum_{k=1}^{n-1} b_k \cdot \sin(k \cdot j \cdot \theta), \quad j = 0, 1, \dots, n-1, \quad (2)$$

ここで $\theta = \pi/n$.

本関数では(1), (2)の左辺に対応して $\{2nb_k\}$ 又は $\{4x_j\}$ を求める. したがって, 結果の正規化は必要に応じて行うこと.

4. 使用例

sine 変換を求める. 入力は乱数により与える. 変換後, さらに逆変換して結果が正しいか確認する.

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

```
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 1024

MAIN__()
{
    int ierr, icon;
    double phai, ran, scale, eps;
    double a[NMAX+2], b[NMAX+2], tab[2*NMAX+4], vw[NMAX*(10+1)/2];
    int i, n, ivw[NMAX*(10-4)/2];

    /* generate initial data */
    n = NMAX;
    phai = (sqrt(5.0)-1.0)/2;
    for (i=1;i<n;i++) {
        ran = (i+1)*phai;
        a[i] = ran - (int)ran;
    }
    for (i=1;i<n;i++)
        b[i] = a[i];
    /* perform normal transform */
    ierr = c_dvsin1(a, n, tab, vw, ivw, &icon);
    /* perform inverse transform */
    ierr = c_dvsin1(a, n, tab, vw, ivw, &icon);
    /* check results */
    scale = 1.0/(8*n);
    eps = 1e-6;
    for (i=0;i<n+1;i++)
        if (fabs((scale*a[i]-b[i])) > eps) {
            printf("WARNING: result inaccurate\n");
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書”の VSIN1 の項目及び[88], [108]を参照のこと。

c_dvsldl

正値対称行列の LDL^T 分解

<code>ierr = c_dvsldl(a, n, epsz, vw, ivw, &icon);</code>

1. 機能

$n \times n$ の正値対称行列 **A** を変形コレスキー法により LDL^T 分解する.

$$\mathbf{A} = \mathbf{L}\mathbf{D}\mathbf{L}^T$$

ただし, **L** は単位下三角行列, **D** は対角行列である. $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dvsldl(a, n, epsz, vw, ivw, &icon);
```

引数の説明:

a	double a[Alen]	入力	係数行列 A . 正値対称行列用圧縮格納法. 格納法の詳細については“1.3.2 対称行列の格納方法”を参照のこと. $Alen = n(n+1)/2$.
		出力	行列 $\mathbf{D}^{-1} + (\mathbf{L} - \mathbf{I})$. 正値対称行列用圧縮格納法. 格納法の詳細については“1.3.2 対称行列の格納方法”を参照のこと.
n	int	入力	係数行列 A の次数 n .
epsz	double	入力	ピボットの相対零判定値 (≥ 0). $epsz = 0$ のときは標準値が採用される. (“使用上の注意”a)参照)
vw	double vw[2n]	作業領域	
ivw	int ivw[n]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
10000	ピボットが負となった. 係数行列は正値でない.	処理は続行する.
20000	ピボットが相対的に零となった. 係数行列は非正則の可能性が強い.	処理を打ち切る.
30000	次のいずれかであった: $n < 1$ $epsz < 0$	処理を打ち切る.

3. 使用上の注意

a) epsz

ピボットの相対零判定値 $epsz$ に 10^{-5} を設定したとすると, この値は次の意味を持っている. すなわち変形コレスキー法による LDL^T 分解の過程でピボットの値が 10 進 s けた以上のけた落ちを生じた場合にそのピボットを相対的に零と見なし, $icon=20000$ として処理を打ち切る. $epsz$ の標準値は丸め誤差の単位を μ としたとき, $epsz=16\mu$ である.

なお、ピボットが小さくなくても計算を続行させる場合には、 epsz へ極小の値を与えればよいが、結果の精度は保証されない。

b) icon

分解過程でピボットが負となった場合、係数行列は正値でないことになる。このとき、 $\text{icon}=10000$ とするが、処理は続行する。ただし、ピボットリングを行っていないため計算誤差は大きい可能性があるので注意が必要である。

c) 行列式

係数行列の行列式の値を求めるには、演算後の配列 a の n 個の対角線上の値($\mathbf{D}^{-1}$ の対角要素)を掛け合わせ、その逆数をとればよい。

4. 使用例

正値対称行列 LDL^T 分解し、連立 1 次方程式を解く。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 100

MAIN__()
{
    int ierr, icon;
    int n, i, j, ij;
    double epsz, eps, sum;
    double a[NMAX*(NMAX+1)/2], b[NMAX], x[NMAX], vw[2*NMAX];
    int ivw[NMAX];

    /* initialize matrix and vector */
    n = NMAX;
    ij = 0;
    for (j=0;j<n;j++)
        for (i=j;i<n;i++)
            a[ij++] = n-i;
    for (i=0;i<n;i++) {
        x[i] = i+1;
        b[i] = 0;
    }
    /* initialize constant vector b = a*x */
    ij = 0;
    for (i=0;i<n;i++) {
        sum = a[ij++]*x[i];
        for (j=i+1;j<n;j++) {
            b[j] = b[j] + a[ij]*x[i];
            sum = sum + a[ij++]*x[j];
        }
        b[i] = b[i]+sum;
    }
    epsz = 1e-6;
    /* LDL decomposition of system of equations */
    ierr = c_dvsldl(a, n, epsz, vw, ivw, &icon);
    if (icon > 10000) {
        printf("ERROR: c_dvsldl failed with icon = %d\n", icon);
        exit(1);
    }
    /* solve decomposed system of equations */
    ierr = c_dvldlx(b, a, n, &icon);
    if (icon > 10000) {
        printf("ERROR: c_dvldlx failed with icon = %d\n", icon);
        exit(1);
    }
    /* check solution vector */
    eps = 1e-6;
    for (i=0;i<n;i++)
        if (fabs((x[i]-b[i])/b[i]) > eps) {
            printf("WARNING: result inaccurate\n");
            exit(1);
        }
}
```

```
    }  
    printf("Result OK\n");  
    return(0);  
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書”の VSLDL の項目を参照のこと。

c_dvspll

正値対称行列 LL^T 分解 (ブロック化されたコレスキー分解法)

```
ierr = c_dvspll(a, k, n, epsz, &icon);
```

1. 機能

$n \times n$ の正値対称行列 A を, ブロック化された外積形のコレスキー分解法により, LL^T 分解する.

$$A = LL^T$$

ただし, L は下三角行列である. $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dvspll((double*)a, k, n, epsz, &icon);
```

引数の説明:

a	double a[n][k]	入力	係数行列 A の上三角部分 $a_{ij}, i \leq j$ を, a の上三角部分 a[i-1][j-1], $i \leq j$ に格納する. 演算後, 内容は保存されない. 格納方法の詳細については, 図 dvsp11-1. を参照のこと.
		出力	a の上三角部分 a[i-1][j-1], $i \leq j$ に $L^T(l_{ij}, i \leq j)$ が格納される.
k	int	入力	配列 a の整合寸法. ($\geq n$)
n	int	入力	係数行列 A の次数 n .
epsz	double	入力	ピボットの相対零判定値 (≥ 0.0). 0.0 のときは標準値が採用される. (“使用上の注意” a) 参照).
icon	int	出力	コンディションコード. 下の表を参照.

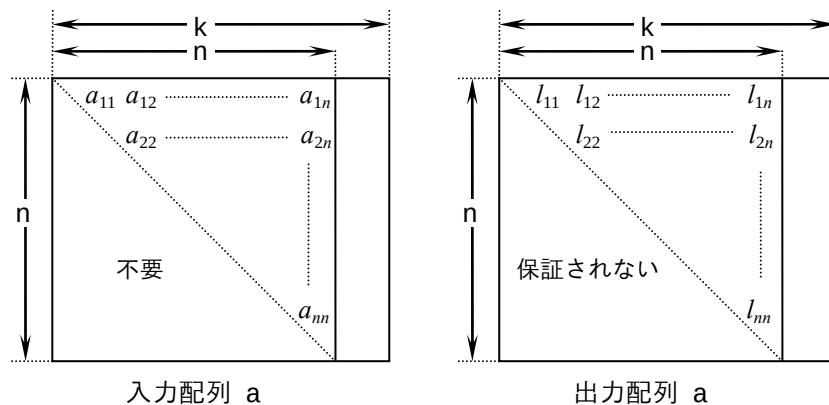


図 dvsp11-1. コレスキー分解法のデータの格納方法

LL^T 分解を行う正値対称行列の対角要素および上三角部分 a_{ij} を配列 a[i-1][j-1], $i=1, \dots, n, j=i, \dots, n$ に格納する. LL^T 分解された結果は, 上三角行列 L^T が上三角部分に格納される. 下三角部分の値は保証されない.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	ピボットが相対的に零となった. 係数行列は非正則の可能性が強い.	処理を打ち切る.
20100	ピボットが負となった. 係数行列は正値でない.	
30000	次のいずれかであった. • $n < 1$ • $\text{epsz} < 0$ • $k < n$	

3. 使用上の注意

a) epsz について

ピボットの相対零判定値 epsz に 10^{-5} を設定したとすると, この値は次の意味を持っている. すなわち, コレスキー分解法による LL^T 分解の過程でピボットの値が正であり, その値より小さくなった場合に, そのピボットの値を相対的に零とみなし, $\text{icon} = 20000$ として処理を打ち切る. epsz の標準値は丸め誤差の単位を μ としたとき, $\text{epsz} = 16\mu$ である.

なお, ピボットが小さくなくても計算を続行させる場合には, epsz へ極小の値を与えればよいが, その結果は保証されない.

b) ピボットが負の場合

分解過程でピボットが負になった場合, 係数行列は正値でないことになる. このとき, $\text{icon} = 20100$ として処理を打ち切る.

c) 行列式

係数行列の行列式の値を求めるには, 演算後の配列 a の n 個の対角要素を掛け合わせ, その 2 乗をとればよい.

4. 使用例

2000 × 2000 の行列を LL^T 分解する.

```
#include <stdio.h>
#include <stdlib.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX      2000
#define KMAX      NMAX+1

MAIN__()
{
    int    epsz, icon, ierr, i, j;
    double a[NMAX][KMAX], b[NMAX], s, det;

    for (i=0; i<NMAX; i++) {
        for (j=i; j<NMAX; j++) {
            a[i][j] = i+1;
        }
    }

    epsz = 0.0;
    ierr = c_dvspll((double*)a, KMAX, NMAX, epsz, &icon);

    if (icon != 0) {
        printf("ERROR: c_dvspll failed with icon = %d\n", icon);
        exit(1);
    }
}
```

```
    for (i=0, s=1.0; i<NMAX; i++) {  
        s = s*a[i][i];  
    }  
  
    det = s*s;  
    printf ("Determinant of matrix = %15.10le\n\n", det);  
    printf ("Decomposed matrix\n");  
  
    for (i=0; i<5; i++) {  
        printf ("i=%d ", i);  
        for (j=i; j<5; j++) {  
            printf ("%15.10le ", a[i][j]);  
        }  
        printf ("\n");  
    }  
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書 II”の VSPLL の項目を参照のこと。

c_dvsplx

LL ^T 分解された正値対称行列の連立 1 次方程式

ierr = c_dvsplx(b, fa, kfa, n, &icon);

1. 機能

LL^T 分解された正値対称な係数行列を持つ連立 1 次方程式(1)を解く.

$$\mathbf{L}\mathbf{L}^T\mathbf{x} = \mathbf{b} \quad (1)$$

ただし, $\mathbf{L}$ は $n \times n$ の下三角行列, $\mathbf{b}$ は n 次元の実定数ベクトル, $\mathbf{x}$ は n 次元の解ベクトルである. $n \geq 1$ であること.

本関数は, 関数 c_dvspll により, LL^T 分解された行列を受けて, 求解処理を行うものである.

2. 引数

呼出し形式:

```
ierr = c_dvsplx(b, (double*)fa, kfa, n, &icon);
```

引数の説明:

b	double b[n]	入力	定数ベクトル $\mathbf{b}$.
		出力	解ベクトル $\mathbf{x}$.
fa	double fa[n][kfa]	入力	LL ^T 分解された行列 $\mathbf{L}^T$ を格納する. LL ^T 分解された行列 $\mathbf{L}^T$ を, fa の上三角部分 fa[i-1][j-1], $i \geq j$ に格納する. 格納方法の詳細については, 図 dvsplx-1. を参照のこと.
kfa	int	入力	配列 fa の整合寸法. ($\geq n$)
n	int	入力	行列 $\mathbf{L}$ の次数 n .
icon	int	出力	コンディションコード. 下の表を参照.

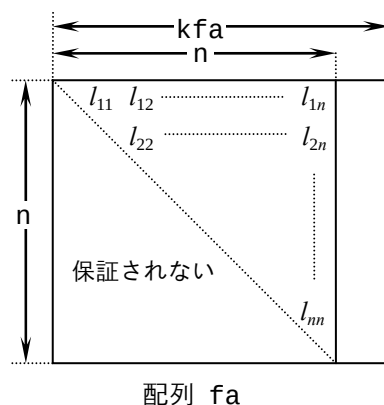


図 dvsplx-1. 配列 fa への行列 $\mathbf{L}^T$ を格納する方法

LL^T 分解された上三角行列 $\mathbf{L}^T$ を, 配列 fa[i-1][j-1], $i=1, \dots, n, j=i, \dots, n$ に格納する.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
20000	係数行列が非正則であった.	処理を打ち切る.
30000	次のいずれかであった. • $n < 1$ • $kfa < n$	

3. 使用上の注意

本関数を、関数 `c_dvspll` に続けて呼び出すことにより、正値対称な係数行列を持つ連立 1 次方程式を解くことができる。しかし、通常は、関数 `c_dvsplx` を呼び出せば、一度に解が求められる。

4. 使用例

2000 × 2000 の行列を LL^T 分解して連立 1 次方程式を解く。

```
#include <stdio.h>
#include <stdlib.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX      2000
#define KMAX      NMAX+1

MAIN__()
{
    int    epsz, isw, icon, ierr, i, j;
    double a[NMAX][KMAX], b[NMAX], s, det;

    for (i=0; i<NMAX; i++) {
        for (j=i; j<NMAX; j++) {
            a[i][j] = i+1;
        }
    }

    for (i=0; i<NMAX; i++) {
        b[i] = (i+1)*(i+2)/2+(i+1)*(NMAX-i-1);
    }

    epsz = 0.0;
    ierr = c_dvspll((double*)a, KMAX, NMAX, epsz, &icon);

    if (icon != 0) {
        printf("ERROR: c_dvspll failed with icon = %d\n", icon);
        exit(1);
    }

    ierr = c_dvsplx(b, (double*)a, KMAX, NMAX, &icon);

    if (icon != 0) {
        printf("ERROR: c_dvsplx failed with icon = %d\n", icon);
        exit(1);
    }

    printf ("Solution vector\n");
    for (i=0; i<10; i++) {
        printf ("b[%d] = %23.16le\n", i, b[i]);
    }

    for (i=0, s=1.0; i<NMAX; i++) {
        s = s*a[i][i];
    }

    det = s*s;
    printf ("\nDeterminant of coefficient matrix = %15.10le\n", det);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書 II”の VSPLX の項目を参照のこと.

c_dvsrft

1次元, 多重離散型実フーリエ変換 (2, 3 及び 5 の混合基底)

ierr = c_dvsrft(x, m, n, isin, isn, w, &icon);

1. 機能

1次元離散型実フーリエ変換を多重(m 組)に行う。変換するデータの大きさ n は 2, 3, 5 巾の積として表される数でなければならない。

変換

$\{x_{kj}\}$ を入力し, (1) で定義する変換を行い, $\{n\alpha_{kh}\}$ を求める。

$$n\alpha_{kh} = \sum_{j=0}^{n-1} x_{kj} \omega_n^{-jhr}, \quad (1)$$

$$\omega_n = \exp(2\pi i / n),$$

$$k = 0, \dots, m-1,$$

$$h = 0, \dots, n-1$$

変換の方向は, $r=1$ 又は $r=-1$ を指定できる。

逆変換

$\{\alpha_{kh}\}$ を入力し, (2) で定義する変換を行い, $\{x_{kj}\}$ を求める。

$$x_{kj} = \sum_{h=0}^{n-1} \alpha_{kh} \omega_n^{jhr}, \quad (2)$$

$$\omega_n = \exp(2\pi i / n),$$

$$k = 0, \dots, m-1,$$

$$j = 0, \dots, n-1$$

$r = -1$ 又は $r = 1$. 逆変換では, 変換で指定した方向と逆向きを指定しなければならない。

実フーリエ変換の結果には複素共役の関係があるため $\{n\alpha_{kh}\}$ の h の最初の $n/2 + 1$ 個を格納する。また m, n の一方は偶数でなければならない。

2. 引数

呼出し形式:

```
ierr = c_dvsrft(x, n, m, isin, isn, w, &icon);
```

引数の説明:

x	double x[Nlen][m] 入力	isn = 1 (実数から複素数への変換) のとき, 実データ $\{x_{kj}\}$ を $x[j][k] = x_{kj}$, $k = 0, \dots, m-1$, $j = 0, \dots, n-1$ に格納する。 isn = -1 (複素数から実数への変換) のとき, 複素データ $\{\alpha_{kh}\}$ の実部を $x[h][k] = \text{Re}(\alpha_{kh})$, $k = 0, \dots, m-1$ に, 虚部を $x[h + (n/2) + 1][k] = \text{Im}(\alpha_{kh})$, $h = 0, \dots, n/2$ に格納する。 $Nlen = n + 4 \times \text{floor}(\sqrt{n/2})$.
	出力	isn = 1 (実数から複素数への変換) のとき, 複素データ $\{n\alpha_{kh}\}$ の実部が $x[h][k] = \text{Re}(n\alpha_{kh})$, $k = 0, \dots, m-1$ に, 虚部が $x[h + (n/2) + 1][k] = \text{Im}(n\alpha_{kh})$, $h = 0, \dots, n/2$ に格納される。 isn = -1 (複素数から実数への変換) のとき, 実データ $\{x_{ki}\}$

			が $x[j][k] = x_{kj}$, $k = 0, \dots, m-1$, $j = 0, \dots, n-1$ に格納される.
m	int	入力	1次元離散型実フーリエ変換を行う多重度(データの本数) m, m, n の一方は偶数でなければならない. (“使用上の注意”a)参照)
n	int	入力	1次元離散型実フーリエ変換を行うデータの大きさ n . n は 2, 3, 5 の中乗の積で表せる数. m, n の一方は偶数でなければ ならない. (“使用上の注意”a)参照)
isin	int	入力	フーリエ変換の方向 r を示す. $isin = 1$ のとき $r = 1$, $isin = -1$ のとき $r = -1$.
isn	int	入力	制御情報. $isn = 1$ のとき 変換(実数から複素数). $isn = -1$ のとき 逆変換(複素数から実数).
w	double w[Wlen]	作業領域	$Wlen = 2n + m(n + 4 \text{ floor}(\sqrt{n/2}))$
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30001	$n \leq 0$ 又は $m \leq 0$.	処理を打ち切る.
30008	n が 2, 3, 5 の積であらわされる整数ではない.	処理を打ち切る.
30016	$isin \neq 1, -1$.	処理を打ち切る.
30032	$isn \neq 1, -1$.	処理を打ち切る.
30512	n, m がともに奇数である.	処理を打ち切る.

3. 使用上の注意

a) n 及び m

n が偶数のときと m が偶数のときとで、別の方法を適用している. n が偶数のとき、ベクトル長は約 $m\sqrt{n}$ である. m が偶数であるときのベクトル長は $m/2$ だが、データ転送量はより少ない. m が十分大きくかつ偶数のとき本ルーチンは最も高速である.

b) 複素共役関係の利用

多重離散型実フーリエ変換には以下の複素共役の関係がある. この関係を使って残りの部分を計算することができる.

$$\alpha_{kh} = \overline{\alpha_{k, n-h}}.$$

c) 一般的なフーリエ変換の定義

多重離散型フーリエ変換及び、逆変換は一般的に(3), (4)で定義される.

$$\alpha_{kh} = \frac{1}{n} \sum_{j=0}^{n-1} x_{kj} \omega_n^{-jhr}, \quad (3)$$

$$\omega_n = \exp(2\pi i / n),$$

$$k = 0, \dots, m-1,$$

$$h = 0, \dots, n-1$$

$$x_{kj} = \sum_{h=0}^{n-1} \alpha_{kh} \omega_n^{jhr} \quad (4)$$

$$\omega_n = \exp(2\pi i / n),$$

$$k = 0, \dots, m-1,$$

$$j = 0, \dots, n-1$$

本関数では, (3), (4)の左辺に対応して $n\alpha_{kh}$ 又は x_{kj} を求める. 従って, 結果の正規化は必要に応じて行うこと.

4. 使用例

実データをフーリエ変換し, その結果をフーリエ逆変換して, もとのデータと比較する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define M 2
#define N 8
#define LDIM (N+4*2)
#define WLEN 2*N+M*LDIM

MAIN__()
{
    int ierr, icon;
    double x[LDIM][M], xx[LDIM][M], eps;
    double (*cx)[2][N/2+1][M]; /* pointer to complex data */
    double w[WLEN];
    int i, j, n, isn, isin, m;

    /* generate initial data */
    m = M;
    n = N;
    for (j=0; j<n; j++)
        for (i=0; i<m; i++)
            x[j][i] = (i+1)*(j+1);
    /* keep copy */
    for (j=0; j<n; j++)
        for (i=0; i<m; i++)
            xx[j][i] = x[j][i];
    /* perform normal transform */
    isn = 1;
    isin = 1;
    ierr = c_dvsrft((double*)x, m, n, isn, isin, w, &icon);
    if (icon != 0) {
        printf("ERROR: c_dvsrft failed with icon = %d\n", icon);
        exit(1);
    }
    /* print complex transformed data */
    cx = (double(*)[2][N/2+1][M])x; /* complex data overwrites real data */
    for (j=0; j<n/2+1; j++) {
        for (i=0; i<m; i++) {
            printf("%8.5f + i*%8.5f ", (*cx)[0][j][i], (*cx)[1][j][i]);
        }
        printf("\n");
    }
    /* perform inverse transform */
    isn = -1;
    isin = -1;
    ierr = c_dvsrft((double*)x, m, n, isn, isin, w, &icon);
    if (icon != 0) {
        printf("ERROR: c_dvsrft failed with icon = %d\n", icon);
        exit(1);
    }
}
```

```
    }
    /* check results */
    eps = 1e-6;
    for (j=0;j<n;j++)
        for (i=0;i<m;i++)
            if (fabs((x[j][i]/n - xx[j][i])/xx[j][i]) > eps) {
                printf("Inaccurate result\n");
                exit(1);
            }
    printf("Result OK\n");
    return(0);
}
```

c_dvtdev

実 3 重対角行列の固有値及び固有ベクトル

```
ierr = c_dvtdev(d, sl, su, n, &nf, ivec, etol,
               ctol, nev, e, &maxne, ev, k, m,
               vw, ivw, &icon);
```

1. 機能

実 3 重対角行列の指定された固有値を求める。指定に応じて対応する固有ベクトルを求める。

$$\mathbf{T}\mathbf{x} = \lambda\mathbf{x}.$$

n 次元の実 3 重対角行列 $\mathbf{T}$ は、次の条件を満たすものとする。

$$l_i u_{i-1} > 0, \quad i = 2, \dots, n,$$

ここで、3 重対角行列 $\mathbf{T}$ の要素を t_{ij} としたとき、 d_i は対角要素を、 $l_i = t_{i,i-1}$ 、 $u_i = t_{i,i+1}$ は副対角要素を表す。ただし、 $l_1 = u_n = 0$ 。

$$(\mathbf{T}\mathbf{x})_i = l_i x_{i-1} + d_i x_i + u_i x_{i+1}, \quad i = 1, \dots, n$$

2. 引数

呼出し形式：

```
ierr = c_dvtdev(d, sl, su, n, &nf, ivec, etol, ctol, nev, e, &maxne,
               (double *) ev, k, (int *) m, vw, ivw, &icon);
```

引数の説明：

d	double d[n]	入力	$\mathbf{T}$ の対角要素 d_i を格納する。
sl	double sl[n]	入力	$\mathbf{T}$ の下副対角要素を $sl[i-1]=l_i, i=1, \dots, n$ のように格納する。
su	double su[n]	入力	$\mathbf{T}$ の上副対角要素を $su[i-1]=u_i, i=1, \dots, n$ のように格納する。
n	int	入力	3 重対角行列の次数 n 。
nf	int	入力	固有値を小さい方から番号付けして (多重固有値には多重度分番号を割り当てる), 求める固有値の最初の番号。固有値は nf から $nf + nev[0] - 1$ まで求める。
		出力	求める最初の固有値が多重固有値である場合も考慮した, 求められた最初の固有値の番号。
ivec	int	入力	制御情報。 $ivec = 1$ のとき固有値及び固有ベクトルの両方を求める。 $ivec \neq 1$ 以外るとき固有値のみ求める。
etol	double	入力	固有値が数値的に異なるか多重かを判定するための判定値。 3.0D-16 以下のときこの値が標準値として設定される。(“使用上の注意”a)参照)
ctol	double	入力	近接している固有値が近似的に多重かを判定するための判定値 ($\geq etol$). $ctol < etol$ のときは, $ctol = etol$ が標準値として設定される。(“使用上の注意”a)参照)
nev	int nev[3]	入力	求める固有値の個数に関する情報。 $nev[0]$ は, 求める固有値の個数。
		出力	求められた固有値の個数に関する情報。 $nev[0]$ は, 異なる固有値の個数。

e	double	出力	nev[1] は, 異なる近似的多重固有値(cluster)の個数.
	e[maxne]		nev[2] は, 多重度も含んだ全ての固有値の個数.
maxne	int	入力	固有値が格納される. 求められた固有値は e[i-1], i = 1,...,nev[2] に格納される.
		出力	計算できる固有値の最大個数. 配列 e の大きさ. (“使用上の注意”b)参照)
			nev[2] > maxne のとき, 固有ベクトルの計算はできない. このとき, maxne に計算を続けるのに必要な最低の値を返却する.
ev	double	出力	ivec = 1 のとき, おおのこの固有値に対応して固有ベクトルが格納される. 求められた固有ベクトルは ev[i-1][j-1], i = 1,...,nev[2], j = 1,...,n に格納される.
	ev[maxne][k]		
k	int	入力	ev の整合寸法 (≥ n).
m	int	出力	求められた固有値の多重度に関する情報.
	m[2][maxne]		m[0][i-1] は i 番目の固有値 λ_i の多重度を, m[1][i-1] は i 番目の固有値 λ_i に関して, 近接している固有値を近似的多重固有値(cluster)と見なしたときの多重度を示す.
vw	double	作業領域	
ivw	int	作業領域	Ivwlen = 9 × maxne + 128.
	ivw[Ivwlen]		
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	多重固有値の計算中, 固有値の総数が maxne を超えた.	処理を打ち切る. 固有ベクトルを求めることはできないが, 異なる固有値自体は求められています. (“使用上の注意”b)参照)
30000	次のいずれかであった. • $n < 1$ • $k < 1$ 又は $k < n$ • $nf < 1$ • $nev[0] < 1$ • $nf + nev[0] > n$	処理を打ち切る.
30100	$sl[i] \times su[i-1] \leq 0$, 行列が対称化できなかった..	処理を打ち切る.

3. 使用上の注意

a) etol 及び ctol

本ルーチンは, 多重固有値(cluster)に対して特別の考慮を払っている.

$\varepsilon = \text{etol}$ としたとき, 連続する固有値 λ_j , $j = s, s+1, \dots, s+k$, ($k \geq 0$) に対して,

$$\frac{|\lambda_i - \lambda_{i-1}|}{1 + \max(|\lambda_{i-1}|, |\lambda_i|)} \leq \varepsilon, \quad (1)$$

$i = s, s+1, \dots, s+k$ となる i に対して(1)を満たし, $i = s-1$ 及び $i = s+k+1$ について(1)を満たさないとき, これらの固有値 λ_j , $j = s-1, s, \dots, s+k$ は数値的に多重であると見なされる.

etol の標準値は 3×10^{-16} ～丸め誤差の単位であり, このとき固有値は計算機で求め得る精度まで分離される.

(1)が $\varepsilon = \text{etol}$ に対して成立しないとき、 λ_{i-1} 及び λ_i は異なる固有値と考える。

$\varepsilon = \text{etol}$ で異なる固有値と見なされた連続する固有値 λ_m , $m=t-1, t, \dots, t+k$, ($k \geq 0$)に対して, $\varepsilon = \text{ctol}$ としたとき, $i=t, t+1, \dots, t+k$ について(1)を満たし, $i=t-1$ 及び $i=t+k+1$ について(1)を満たさないとき, これらの異なる固有値 λ_m , $m=t-1, t, \dots, t+k$ を近似的に多重(cluster)と見なす。これは, clusterに対応する不変部分空間, つまり1次独立な固有ベクトルを計算するために利用される。つまり, 対応する固有ベクトルは直交する初期ベクトルを用いて計算され, 再直交化される。もちろん, $\text{ctol} \geq \text{etol}$ を満たさなければならない。この条件を満たさないとき, ctol は etol の値と等しく設定される。

b) maxne

r 個数の固有値を求めるとき, 最初又は最後の固有値が, 1より大きい多重度を持つとき, r 個以上の固有値が求まる。対応する固有ベクトルは増えた固有値も含んで, 対応する固有ベクトルの格納域が足りるときのみ計算される。maxneに, 計算できる固有値の最大個数を指定できます。固有値の総数がmaxneを超えた場合, icon=20000を返却する。このとき, 固有ベクトルの計算を行うよう指定されていたら固有ベクトルの計算はできない。固有値は求められているが, 多重度分だけ繰り返して格納はされていない。つまり, 固有値に関しては, 求められた異なる固有値が $e[i-1]$, $i=1, \dots, \text{nev}[0]$ に, 対応する固有値の多重度が $m[0][i-1]$, $i=1, \dots, \text{nev}[0]$ にそれぞれ格納されている。固有値がすべて異なり, 近似的な多重な固有値もない場合, maxneは入力として $\text{nev}[0]$ に指定する求める固有値の総数で十分である。

c) 本機能を利用して解ける問題

本機能は, $l_{i-1} > 0$ のみを要求する。このため, (2)の一般化固有値問題を $\mathbf{T} \leftarrow \mathbf{T}\mathbf{D}^{-1}$ と置き換えて解くことができる。

$$\mathbf{T}\mathbf{x} = \lambda\mathbf{D}\mathbf{x}, \quad (2)$$

ここで, $\mathbf{D} > \mathbf{0}$ の対角行列。

また, $\mathbf{T}$ に対する固有値問題を対称な一般化固有値問題に変換することもできる。

$$(\mathbf{D}\mathbf{T} - \lambda\mathbf{D})\mathbf{x} = \mathbf{0},$$

ここで, $\mathbf{D}_1 = 1$, $\mathbf{D}_i = u_{i-1}\mathbf{D}_{i-1}/l_i$, $i = 2, \dots, n$ 。

$\mathbf{D}_i$ がスケーリング問題を引き起こす可能性があるとき, 以下の対称な問題を考えることが好まれる。

$$(\mathbf{D}^{1/2}\mathbf{T}\mathbf{D}^{-1/2} - \lambda\mathbf{I})\mathbf{w} = \mathbf{0},$$

このとき固有ベクトルの間には $\mathbf{w} = \mathbf{D}^{1/2}\mathbf{x}$ なる関係がある。

4. 使用例

実3重対角行列の固有値を計算する。

```
#include <stdio.h>
#include <stdlib.h>
#include "cssl.h" /* standard C-SSL II header file */

#define P1 350
#define Q1 2
#define NMAX P1*Q1
#define N0 584
#define N1 686
#define NE N1-N0+1
#define MAXNE NE+2*Q1

MAIN__()
{
    int ierr, icon;
    int n, m[2][NMAX], nf, ivec, maxne, nev[3], i, j, k, ii;
    double d[NMAX], sl[NMAX], su[NMAX], e[MAXNE], ev[MAXNE][NMAX];
    double etol, ctol, vw[12*NMAX];
    int ivw[9*MAXNE+128];

    /* initialize matrix */
    n = NMAX;
```

```

k = NMAX;
j = (P1+1)/2;
d[j-1] = 0;
for (i=1;i<j;i++) {
    sl[i] = 1;
    su[i-1] = 1;
    sl[j+i-1] = 1;
    su[j+i-2] = 1;
    d[i-1] = j-i;
    d[2*j-i-1] = d[i-1];
}
sl[0] = 0;
su[P1-1] = 0;
for (j=2;j<=Q1;j++) {
    ii = (j-1)*P1;
    for (i=1;i<=P1;i++) {
        sl[ii+i-1] = sl[i-1];
        su[ii+i-1] = su[i-1];
        d[ii+i-1] = d[i-1];
    }
}
sl[0] = 0;
su[n-1] = 0;
nf = N0;
ivec = 1;
etol = 0;
ctol = 0;
nev[0] = NE;
maxne = MAXNE;
/* find eigenvalues only */
ierr = c_dvtdev(d, sl, su, n, &nf, ivec, etol, ctol, nev, e, &maxne,
               (double*)ev, k, (int*)m, vw, ivw, &icon);
if (icon > 20000) {
    printf("ERROR: c_dvtdev failed with icon = %d\n", icon);
    exit(1);
}
printf("icon = %i\n", icon);
/* print distinct eigenvalues */
ii = 0;
for (i=0;i<nev[0];i++) {
    printf("eigenvalue %i : %7.4f with multiplicity %i\n", nf+ii, e[ii], m[0][ii]);
    if (icon == 20000) ii = ii+1;
    else ii = ii+m[0][ii];
}
return(0);
}

```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II 拡張機能使用手引書 II”の VTDEV の項目及び[31], [81], [96], [118]を参照のこと。

c_dvtfqd

非対称又は不定値のスパース実行列の連立 1 次方程式(TFQMR 法, 対角形式格納法)

```
ierr = c_dvtfqd(a, k, ndiag, n, nofst, b,
                itmax, eps, iguss, x, &iter, vw,
                &icon);
```

1. 機能

$n \times n$ の非対称/不定値なスパース行列を係数行列とする連立 1 次方程式を転置なし準最小残差法 (TFQMR: Transpose-Free Quasi-Minimal Residual method) で解く。

$$\mathbf{Ax} = \mathbf{b}$$

係数行列 $\mathbf{A}$ は、対角形式格納法で格納する。ベクトル $\mathbf{b}$ 及び $\mathbf{x}$ は n 次元ベクトル。

本方法は一般に MGCR 法より高速だが、反復計算が係数行列や初期ベクトルの特性のため続けられない場合 (break down) がある。これは再帰的計算式で、非零を期待される計算のある中間結果が零になるためである。この場合 break down を起こさない MGCR 法を使っていたきたい。

2. 引数

呼出し形式:

```
ierr = c_dvtfqd((double *) a, k, ndiag, n, nofst, b, itmax, eps, iguss, x,
                &iter, vw, &icon);
```

引数の説明:

a	double a[ndiag][k]	入力	係数行列 $\mathbf{A}$. 係数行列の非零要素を一般スパース行列の対角形式格納法で格納する。格納法の詳細については“1.3.5 一般スパース行列の格納方法”を参照のこと。
k	int	入力	配列 a の整合寸法 ($\geq n$).
ndiag	int	入力	係数行列 $\mathbf{A}$ の非零要素を含む対角ベクトルの総数。
n	int	入力	行列 $\mathbf{A}$ の次数 n .
nofst	int nofst[ndiag]	入力	配列 a に格納される対角ベクトルに対応した主対角ベクトルからの距離を格納する。上対角ベクトル列は正, 下対角ベクトル列は負の値で表す。
b	double b[n]	入力	定数ベクトル $\mathbf{b}$.
itmax	int	入力	TFQMR 法の反復回数の上限值 (> 0). 2000 程度で十分である。
eps	double	入力	収束判定に用いられる判定値. eps が 0.0 以下のとき, 10^{-6} が設定される. (“使用上の注意”a)参照)
iguss	int	入力	配列 x に指定された解ベクトルの近似値から反復計算を行うかを指定する制御情報。 0 のとき解ベクトルの近似値を指定しない。 0 以外のとき配列 x に指定された解ベクトルの近似値から反復計算を開始する。
x	double x[n]	入力 出力	解ベクトルの近似値を指定することができる。 解ベクトル $\mathbf{x}$ が格納される。
iter	int	出力	TFQMR 法での実際の反復回数。
vw	double vw[Vwlen]	作業領域	$Vwlen = 10k + n + ndiag - 1$.

icon int 出力 コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
20000	break down を起こした.	処理を打ち切る.
20001	反復回数の上限に達した.	処理を打ち切る. 配列 x には, そのときまでに得られている近似値を出力するが, 精度は保証できない.
30000	次のいずれかであった. • $n < 1$ • $k < 1$ 又は $k < n$ • $ndiag < 1$ 又は $ndiag > k$ • $itmax \leq 0$	処理を打ち切る.
32001	$ nofst[i-1] > n-1$	処理を打ち切る.

3. 使用上の注意

a) eps

TFQMR 法は, 残差のユーグリッドノルムが最初の残差のユーグリッドノルムと eps の積以下になったとき収束したと見なす. 正確な解と求められた近似解の誤差はほぼ行列 **A** の条件数と eps の積に等しくなる.

b) 対角形式を使う上での注意

係数行列 **A** の外側の対角ベクトルの要素は零を設定する必要がある. 対角ベクトル列を配列 a に格納する順序に制限はない. この方法の利点は, 行列ベクトル積が間接指標を使わずに計算できる点であり, 対角構造を持たない行列は効率よく格納できないという点が欠点である.

4. 使用例

実スパース行列の連立 1 次方程式を解く.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX      100
#define UBANDW    2
#define LBANDW    1

MAIN__()
{
    double one=1.0, bcoef=10.0, eps=1.e-6;
    int ierr, icon, nub, nlb, n, i, j, k;
    int itmax, iguss, iter;
    int nofst[UBANDW + LBANDW + 1];
    double a[UBANDW + LBANDW + 1][NMAX], b[NMAX], x[NMAX];
    double vw[NMAX*10+NMAX+UBANDW+LBANDW];

    /* initialize nonsymmetric matrix and vector */
    nub = UBANDW;
    nlb = LBANDW;
    ndiag = nub + nlb + 1;
    n = NMAX;
    k = NMAX;
    for (i=1; i<=nub; i++) {
        for (j=0 ; j<n-i; j++) a[i][j] = -1.0;
        for (j=n-i; j<n ; j++) a[i][j] = 0.0;
        nofst[i] = i;
    }
    for (i=1; i<=nlb; i++) {
        for (j=0 ; j<i+1; j++) a[nub + i][j] = 0.0;
```

```
    for (j=i+1; j<n ; j++) a[nub + i][j] = -2.0;
    nofst[nub + i] = -(i + 1);
}
nofst[0] = 0;
for (j=0; j<n; j++) {
    a[0][j] = bcoef;
    for (i=1; i<ndiag; i++) a[0][j] -= a[i][j];
    b[j] = bcoef;
}
/* solve the system of linear equations */
itmax = n;
iguss = 0;
ierr = c_dvtfqd ((double*)a, k, ndiag, n, nofst, b, itmax, eps,
                iguss, x, &iter, vw, &icon);
if (icon != 0) {
    printf("ERROR: c_dvtfqd failed with icon = %d\n", icon);
    exit(1);
}
/* check vector */
for (i=0; i<n; i++)
    if (fabs(x[i]-one) > eps) {
        printf("WARNING: result inaccurate\n");
        exit(1);
    }
printf("Result OK\n");
return(0);
}
```

5. 手法概要

TFQMR 法については[36]を参照のこと.

c_dvtfqe

非対称又は不定値のスパース実行列の連立 1 次方程式 (TFQMR 法, ELLPACK 形式格納法)

```
ierr = c_dvtfqe(a, k, iwid, n, icol, b,
                itmax, eps, iguss, x, &iter, vw,
                &icon);
```

1. 機能

$n \times n$ の非対称/不定値なスパース行列を係数行列とする連立 1 次方程式を転置なし準最小残差法 (TFQMR: Transpose-Free Quasi-Minimal Residual method) で解く。

$$\mathbf{Ax} = \mathbf{b}$$

係数行列は, ELLPACK 形式の格納法で格納する。ベクトル $\mathbf{b}$ 及び $\mathbf{x}$ は n 次元ベクトル。

本方法は一般に MGCR 法より高速だが, 反復計算が係数行列や初期ベクトルの特性のため続けられない場合 (break down) がある。これは再帰的計算式で, 非零を期待される計算のある中間結果が零になるためである。この場合 break down を起こさない MGCR 法をお使いいただきたい。

2. 引数

呼出し形式:

```
ierr = c_dvtfqe((double *) a, k, iwid, n, (double *) icol, b, itmax, eps,
                iguss, x, &iter, vw, &icon);
```

引数の説明:

a	double a[iwid][k]	入力	係数行列 $\mathbf{A}$. 係数行列の非零要素を一般スパース行列の ELLPACK 形式格納法で格納する。格納法の詳細については“1.3.5 一般スパース行列の格納方法”を参照のこと。
k	int	入力	配列 a 及び icol の整合寸法 ($\geq n$).
iwid	int	入力	係数行列 $\mathbf{A}$ の非零要素の行ベクトル方向の最大個数。
n	int	入力	行列 $\mathbf{A}$ の次数 n .
icol	int icol[iwid][k]	入力	ELLPACK 形式で使用される列指標で, a の対応する要素が いずれの列ベクトルに属するかを示す。
b	double b[n]	入力	定数ベクトル $\mathbf{b}$.
itmax	int	入力	TFQMR 法の反復回数の上限值 (>0). 2000 程度で十分である。
eps	double	入力	収束判定に用いられる判定値. eps が 0.0 以下のとき, eps は 10^{-6} が設定される. (“使用上の注意”参照)
iguss	int	入力	配列 x に指定された解ベクトルの近似値から反復計算を開始するかを示す制御情報。 0 のとき解ベクトルの近似値を指定しない。 0 以外のとき配列 x に指定された解ベクトルの近似値から反復計算を開始する。
x	double x[n]	入力 出力	解ベクトルの近似値を指定することができる。 解ベクトル $\mathbf{x}$ が格納される。
iter	int	出力	TFQMR 法での実際の反復回数。
vw	double vw[13k]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照。

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
20000	break down を起こした.	処理を打ち切る.
20001	反復回数の上限に達した.	処理を打ち切る. 配列 x には, そのときまでに得られている近似値を出力するが, 精度は保証できない.
30000	次のいずれかであった: • $n < 1$ • $k < 1$ 又は $k < n$ • $iwidth < 1$ 又は $iwidth > k$ • $itmax \leq 0$	処理を打ち切る.

3. 使用上の注意

a) eps

TFQMR 法は, 残差のユーグリッドノルムが最初の残差のユーグリッドノルムと ϵ の積以下になったとき収束したと見なす. 正確な解と求められた近似解の誤差はほぼ行列 A の条件数と ϵ の積に等しくなる.

4. 使用例

実スパース行列の連立 1 次方程式を解く.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX      100
#define UBANDW     2
#define LBANDW     1

MAIN__()
{
    double lcf=-2.0, ucf=-1.0, bcoef=10.0, one=1.0, eps=1.e-6;
    int ierr, icon, nlb, nub, iwidth, n, k, itmax, iguss, iter, i, j, ix;
    int icol[UBANDW + LBANDW + 1][NMAX];
    double a[UBANDW + LBANDW + 1][NMAX], b[NMAX], x[NMAX];
    double vw[NMAX * 13];

    /* initialize matrix and vector */
    nub = UBANDW;
    nlb = LBANDW;
    iwidth = UBANDW + LBANDW + 1;
    n = NMAX;
    k = NMAX;
    for (i=0; i<n; i++) b[i] = bcoef;
    for (i=0; i<iwidth; i++)
        for (j=0; j<n; j++) {
            a[i][j] = 0.0;
            icol[i][j] = j+1;
        }
    for (j=0; j<nlb; j++) {
        for (i=0; i<j; i++) a[i][j] = lcf;
        a[j][j] = bcoef - (double) j * lcf - (double) nub * ucf;
        for (i=j+1; i<j+1+nub; i++) a[i][j] = ucf;
        for (i=0; i<=nub+j; i++) icol[i][j] = i+1;
    }
    for (j=nlb; j<n-nub; j++) {
        for (i=0; i<nlb; i++) a[i][j] = lcf;
        a[nlb][j] = bcoef - (double) nlb * lcf - (double) nub * ucf;
        for (i=nlb+1; i<iwidth; i++) a[i][j] = ucf;
        for (i=0; i<iwidth; i++) icol[i][j] = i+1+j-nlb;
    }
}
```

```

    for (j=n-nub; j<n; j++){
        for (i=0; i<nlb; i++) a[i][j] = lcf;
        a[nlb][j] = bcoef - (double) nlb * lcf - (double) (n-j-1) * ucf;
        for (i=1; i<nub-2+n-j; i++) a[i+nlb][j] = ucf;
        ix = n - (j+nub-nlb-1);
        for (i=n; i>=j+nub-nlb-1; i--) icol[ix--][j] = i;
    }
    /* solve the system of linear equations */
    itmax = n;
    iguss = 0;
    ierr = c_dvtfqe ((double*)a, k, iwidt, n, (int*)icol, b, itmax,
                    eps, iguss, x, &iter, vw, &icon);
    if (icon != 0) {
        printf("ERROR: c_dvtfqe failed with icon = %d\n", icon);
        exit(1);
    }
    /* check vector */
    for (i=0; i<n; i++)
        if (fabs(x[i]-one) > eps) {
            printf("WARNING: result inaccurate\n");
            exit(1);
        }
    printf("Result OK\n");
    return(0);
}

```

5. 手法概要

TFQMR 法については[36]を参照のこと.

c_dvwflt

ウェーブレットフィルターの生成

ierr = c_dvwflt(f, n, &icon);

1. 機能

コンパクトな台を持つ次数 n の Daubechies のウェーブレットに対応するフィルターを生成する. 次数 n は 2, 4, 6, 12 及び 20 のフィルターが生成できる.

2. 引数

呼出し形式:

```
ierr = c_dvwflt(f, n, &icon);
```

引数の説明:

f	double f[2n]	入力	変換で使われるウェーブレットフィルターが格納される. (“使用上の注意”a)参照)
n	int	入力	ウェーブレットフィルターの係数の数. 2, 4, 6, 12 又は 20 のいずれかの数.
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
30000	n が 2, 4, 6, 12, 20 のいずれの数でもない.	処理を打ち切る.

3. 使用上の注意

a) フィルターの条件

一般に本機能で使われる直交フィルター(orthogonal filter)は, 長さ $2 \times n$ のベクトルで, $f[0], f[1], \dots, f[n-1]$ は, Low-pass フィルターを, $f[n], f[n+1], \dots, f[2n-1]$ は High-pass フィルターを含んでいる. これらの係数には, 以下の関係がある.

$$\sum_{i=0}^{n-1} f[i]^2 = 1, \quad f[2n-1-i] = (-1)^{i+1} f[i], \quad i = 0, 1, \dots, n-1.$$

b) c_dv1dwt 及び c_dv2dwt

本ルーチンで生成したフィルターは c_dv1dwt 又は c_dv2dwt で使われる. 本ルーチンの引数 n 及び f は, c_dv1dwt 及び c_dv2dwt の k 及び f にそれぞれ対応している.

4. 使用例

ウェーブレットフィルターを生成し, 1次元ウェーブレット変換及び逆変換を行う.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 1024
#define KMAX 6

MAIN__()
{
```

```

int ierr, icon;
double phai, ran, eps;
double x[NMAX], y[NMAX], f[2*KMAX], xx[NMAX];
int isn, i, k, ls, n;

/* generate initial data */
n = NMAX;
ls = 10;
k = KMAX;
phai = (sqrt(5.0)-1.0)/2;
for (i=0;i<n;i++) {
    ran = (i+1)*phai;
    x[i] = ran - (int)ran;
}
for (i=0;i<n;i++)
    xx[i] = x[i];
/* generate wavelet filter */
ierr = c_dvwflt(f, k, &icon);
if (icon != 0 ) {
    printf("ERROR: c_dvwflt failed with icon = %i\n", icon);
    exit (1);
}
/* perform normal wavelet transform */
isn = 1;
ierr = c_dvldwt(x, n, y, isn, f, k, ls, &icon);
if (icon != 0 ) {
    printf("ERROR: c_dvldwt failed with icon = %i\n", icon);
    exit (1);
}
/* perform inverse wavelet transform */
isn = -1;
ierr = c_dvldwt(x, n, y, isn, f, k, ls, &icon);
if (icon != 0 ) {
    printf("ERROR: c_dvldwt failed with icon = %i\n", icon);
    exit (1);
}
/* check results */
eps = 1e-6;
for (i=0;i<n;i++)
    if (fabs((x[i]-xx[i])/xx[i]) > eps) {
        printf("Inaccurate result\n");
        exit(1);
    }
printf("Result OK\n");
return(0);
}

```

c_ranb2

二項乱数の生成 (単精度)

<pre>ierr = c_ranb2(m, p, &ix, ia, n, vw, ivw, &icon);</pre>

1. 機能

与えられた母数 m, p で決まる二項分布の確率密度関数から, 指定された n 個の擬似乱数(整数)を生成する.

$$P_k = \binom{m}{k} p^k (1-p)^{m-k}, \quad 0 < p < 1 \quad k = 0, 1, 2, \dots, m \quad m = 1, 2, 3, \dots$$

$n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_ranb2(m, p, &ix, ia, n, vw, ivw, &icon);
```

引数の説明:

m	int	入力	母数 m .
p	float	入力	母数 p .
ix	int	入力	初期値. 非負の整数. (“使用上の注意” a)参照).
		出力	次の本関数呼出しのための初期値.
ia	int ia[n]	出力	n 個の二項乱数.
n	int	入力	生成する二項乱数の個数 n .
vw	float vw[m+1]	作業領域	
ivw	int ivw[m+1]	作業領域	
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意味	処理内容
0	エラーなし.	正常終了.
30000	次のいずれかであった. $m < 1$ $p \leq 0, p \geq 1$ $ix < 0$ $n < 1$	処理を打ち切る.

3. 使用上の注意

a) ix について

本関数は, 一様乱数から二項乱数への変換を行っている. 引数 ix は一様乱数を生成するときの初期値として与えるものである.

b) vw と ivw について

引数 m, p を同じ値で使用している間は, vw と ivw の内容を変更してはならない.

4. 使用例

10000 個の 2 項乱数を求め, 平均と標準偏差を計算する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define N 10000
#define M 20

MAIN__()
{
    int ierr, icon;
    int m, n, ix, i, ia[N], ivw[M+1], sum, sumsq;
    float p, vw[M+1], mean, dev;

    /* initialize parameters */
    n = N;
    ix = 12345;
    m = M;
    p = 0.75;
    /* generate pseudo-random numbers */
    ierr = c_ranb2(m, p, &ix, ia, n, vw, ivw, &icon);
    if (icon != 0) {
        printf("ERROR: c_ranb2 failed with icon = %d\n", icon);
        exit(1);
    }
    /* calculate mean and deviation */
    sum = 0;
    sumsq = 0;
    for (i=0; i<n; i++) {
        sum = sum+ia[i];
        sumsq = sumsq+ia[i]*ia[i];
    }
    mean = (double)sum/n;
    dev = sqrt((double)sumsq/n - mean*mean);
    printf("observed mean = %12.4e    deviation = %12.4e\n",
        mean, dev);
    printf("calculated mean = %12.4e    deviation = %12.4e\n",
        m*p, sqrt(m*p*(1-p)));
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の RANB2 の項目及び[138]を参照のこと.

c_rane2

指数乱数の生成 (単精度)

ierr = c_rane2(am, &ix, a, n, &icon);

1. 機能

与えられた平均 m を持つ指数分布の確率密度関数から, 指定された n 個の擬似乱数を生成する.

$$g(x) = \frac{1}{m} e^{-x/m}$$

ここに, $x \geq 0$, $m > 0$ である. また $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_rane2(am, &ix, a, n, &icon);
```

引数の説明:

am	float	入力	指数分布の平均 m .
ix	int	入力	初期値. 非負の整数.
		出力	次の本関数呼出しのための初期値. (“使用上の注意”参照).
a	float a[n]	出力	n 個の指数乱数.
n	int	入力	生成する指数乱数の個数 n .
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
30000	次のいずれかであった. • am ≤ 0 • ix < 0 • n < 1	処理を打ち切る.

3. 使用上の注意

a) ix について

本関数は一様乱数を生成して, 指数乱数へ変換することを行っている. 引数 ix は一様乱数を生成するときの初期値として与えるものである.

4. 使用例

10000 個の指数乱数を計算し, 平均と分散を求める.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */
```

```
#define NMAX 10000

MAIN__()
{
    int ierr, icon;
    int n, ix, i;
    float a[NMAX], am, sum, sumsq, mean, dev;

    /* initialize parameters */
    n = NMAX;
    ix = 12345;
    am = 1;
    /* generate pseudo-random numbers */
    ierr = c_rane2(am, &ix, a, n, &icon);
    if (icon != 0) {
        printf("ERROR: c_rane2 failed with icon = %d\n", icon);
        exit(1);
    }
    /* calculate mean and deviation */
    sum = 0;
    sumsq = 0;
    for (i=0;i<n;i++) {
        sum = sum+a[i];
        sumsq = sumsq+a[i]*a[i];
    }
    mean = sum/n;
    dev = sqrt(sumsq/n - mean*mean);
    printf("observed mean = %12.4e    deviation = %12.4e\n",
        mean, dev);
    printf("calculated mean = %12.4e    deviation = %12.4e\n",
        1.0, 1.0);
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の RANE2 の項目を参照のこと。

c_ranp2

ポアソン乱数の生成 (単精度)

<pre>ierr = c_ranp2(am, &ix, ia, n, vw, ivw, &icon);</pre>

1. 機能

与えられた平均 m を持つポアソン分布の確率密度関数から指定された n 個の擬似乱数 (整数) を生成する。

$$P_k = \frac{m^k}{k!} e^{-m}$$

ここに, $m > 0$, k は非負の整数である。また $n \geq 1$ であること。

2. 引数

呼出し形式:

```
ierr = c_ranp2(am, &ix, ia, n, vw, ivw, &icon);
```

引数の説明:

am	float	入力	ポアソン分布の平均 m . (使用上の注意 a)参照).
ix	int	入力	初期値. 非負の整数変数.
		出力	次の本関数呼出しのための初期値. (使用上の注意 b)参照).
ia	int ia[n]	出力	n 個のポアソン乱数.
n	int	入力	生成するポアソン乱数の個数 n .
vw	float	作業領域	
	vw[2m+10]		
ivw	int	作業領域	
	ivw[2m+10]		
icon	int	出力	コンディションコード. 下の表を参照.

コンディションコード:

コード	意 味	処 理 内 容
0	エラーなし.	正常終了.
30000	次のいずれかであった. • am ≤ 0 • ix < 0 • n < 1 • am $> \log(f_{\max})$. (使用上の注意 a)参照)	処理を打ち切る.

3. 使用上の注意

a) am について

引数 $\text{am} \leq \log(f_{\max})$ であること. am の値がこれを超えると e^{-m} の計算においてアンダーフローを起こす。

引数 am の値が大きいとき ($\text{am} \geq 20$) は, ポアソン分布を, 平均 m 標準偏差 m の正規分布で近似し, 正規乱数としての擬似乱数を生成したほうがよい。

b) ix について

本関数は一様乱数よりポアソン乱数へ変換することを行っている。引数 ix は一様乱数を生成するときの初期値として与えるものである。

c) vw と ivw について

引数 am を同じ値で使用している間は vw, ivw の内容を変更してはならない。

4. 使用例

10000 個のポアソン乱数を計算し, 平均と分散を求める。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL II header file */

#define NMAX 10000
#define MMAX 20

MAIN__()
{
    int ierr, icon;
    int n, ix, i, ia[NMAX], ivw[2*MMAX+10], sum, sumsq;
    float am, vw[2*MMAX+10], mean, dev;

    /* initialize parameters */
    n = NMAX;
    am = 1;
    ix = 12345;
    /* generate pseudo-random numbers */
    ierr = c_ranp2(am, &ix, ia, n, vw, ivw, &icon);
    if (icon != 0) {
        printf("ERROR: c_ranp2 failed with icon = %d\n", icon);
        exit(1);
    }
    /* calculate mean and deviation */
    sum = 0;
    sumsq = 0;
    for (i=0; i<n; i++) {
        sum = sum+ia[i];
        sumsq = sumsq+ia[i]*ia[i];
    }
    mean = (double)sum/n;
    dev = sqrt((double)sumsq/n - mean*mean);
    printf("observed mean = %12.4e    deviation = %12.4e\n",
        mean, dev);
    printf("calculated mean = %12.4e    deviation = %12.4e\n",
        am, sqrt(am));
    return(0);
}
```

5. 手法概要

計算方法の詳細については“富士通 SSL II 使用手引書”の RANP2 の項目を参照のこと。

付録 A 補助ルーチン

A.1 概要

補助関数とは, C-SSL II の一般関数並びにスレーブ関数の機能を, 分野を問わず内部的に補助する関数である. 例えば, C-SSL II の関数では丸め誤差の単位 μ (unit round off) を頻繁に用いているが, この値 μ を設定する補助関数を用意し, 各関数は必要に応じてこの補助関数を呼び出すことにより, 値 μ を利用している.

c_dcsum

積和計算 (複素ベクトル)

<code>ierr = c_dcsum(za, zb, n, ia, ib, &zsum);</code>

1. 機能

n 次元の複素ベクトル **a**, **b** が与えられたとき, 次のような積和を計算する.

$$\sigma = \sum_{i=1}^n a_i b_i$$

ただし, $\mathbf{a}^T = (a_1, a_2, \dots, a_n)$, $\mathbf{b}^T = (b_1, b_2, \dots, b_n)$. $n \geq 1$ であること.

2. 引数

呼出し形式:

```
ierr = c_dcsum(za, zb, n, ia, ib, &zsum);
```

引数の説明:

za	dcomplex	入力	ベクトル a . $Alen = ia * n$. za[Alen]
zb	dcomplex	入力	ベクトル b . $Blen = ib * n$. zb[Blen]
n	int	入力	ベクトル a , b の次元 n .
ia	int	入力	ベクトル a の要素の間隔 ($\neq 0$). 通常 1 と指定する. (“使用上の注意”参照)
ib	int	入力	ベクトル b の要素の間隔 ($\neq 0$). 通常 1 と指定する. (“使用上の注意”参照)
zsum	dcomplex	出力	積和値 σ . (“使用上の注意”参照)

3. 使用上の注意

a) 配列 **za**, **zb** 内のデータ間隔

ベクトル **a** の要素が, 配列 za 上に間隔 p で格納されている場合, $ia = p$ と指定すればよい. 同様に, ベクトル **b** において間隔 q ならば, $ib = q$ と指定すればよい. なお, $p, q < 0$ の場合は, 配列 za, zb の与え方に注意すること.

4. 使用例

複素行列の行と列による積和を計算する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 100

MAIN__()
{
    int n, i, j, ia, ib;
    double eps;
    dcomplex zsum, zsum2;
    dcomplex zmat[NMAX][NMAX], *za, *zb;
```

```
/* initialize matrix */
n = NMAX;
for (i=0;i<n;i++)
    for (j=0;j<n;j++) {
        zmat[i][j].re = i+j+1;
        zmat[i][j].im = i-j+1;
    }
/* calculate the product sum of row 6 and column 3 */
za = &zmat[6][0];
ia = 1;
zb = &zmat[0][3];
ib = NMAX;
c_dcsum(za, zb, n, ia, ib, &zsum);
/* check sum */
eps = 1e-6;
zsum2.re = 0;
zsum2.im = 0;
for (i=0;i<n;i++) {
    zsum2.re = zsum2.re + za[i*ia].re*zb[i*ib].re-za[i*ia].im*zb[i*ib].im;
    zsum2.im = zsum2.im + za[i*ia].re*zb[i*ib].im+za[i*ia].im*zb[i*ib].re;
}
if ((fabs((zsum2.re-zsum.re)/zsum.re) > eps) ||
    (fabs((zsum2.im-zsum.im)/zsum.im) > eps)) {
    printf("WARNING: result inaccurate\n");
    exit(1);
}
printf("Result OK\n");
return(0);
}
```

c_dfmax

浮動小数点数の正の最大値
<code>result = c_dfmax();</code>

1. 機能

正規化された浮動小数点数の正の最大値 $fl_{\max}$ を設定する. これらの値は計算機の演算方法 (2 進法か 16 進法か) により異なる.

2. 引数

本関数を呼び出すと `double` 型の値を返却する.

呼出し形式:

```
result = c_dfmax();
```

3. 使用上の注意

$fl_{\max}$ の値は次のように計算される.

演算方法	最大値	適用例
16 進法	$(1-16^{-14}) \cdot 16^{63}$	M シリーズ S シリーズ SX/G 200 シリーズ
2 進法	$(1-2^{-53}) \cdot 2^{1024}$	VPP シリーズ FM シリーズ
	$(1-2^{-56}) \cdot 2^{252}$	SX/G 100 シリーズ

c_dfmin

浮動小数点数の正の最小値

<code>result = c_dfmin();</code>

1. 機能

正規化された浮動小数点数の正の最小値 $f_{\min}$ を設定する。これらの値は計算機の演算方法 (2 進法か 16 進法か) により異なる。

2. 引数

本関数を呼び出すと `double` 型の値を返却する。

呼出し形式:

```
result = c_dfmin();
```

3. 使用上の注意

$f_{\min}$ の値は次のように計算される。

演算方法	最小値	適用例
16 進法	$16^{-1} \cdot 16^{-64}$	M シリーズ S シリーズ SX/G 200 シリーズ
2 進法	$2^{-1} \cdot 2^{-1021}$	VPP シリーズ FM シリーズ
	$2^{-1} \cdot 2^{-259}$	SX/G 100 シリーズ

c_dmach

丸め誤差の単位 (unit round-off)
result = c_dmach();

1. 機能

関数の値は、正規化された浮動小数点演算における、丸め誤差の単位 μ (unit round off)の値が double 型で設定される。ここで μ は、 M 進法 L 桁の演算で

$$\mu = M^{1-L/2}$$
$$\mu = M^{1-L}$$

(四捨五入演算)
(切り捨て演算)

である。（“使用上の注意”参照）

2. 引数

呼出し形式：

result = c_dmach();

3. 使用上の注意

丸め誤差の単位

演算方法		dmach	適用例
16進法 : M = 16	切り捨て	L = 14, $\mu = 16^{-13}$	M シリーズ S シリーズ SX/G 200 シリーズ
2進法 : M = 2	四捨五入	L = 52, $\mu = \frac{1}{2} 2^{-51}$	VPP シリーズ FM シリーズ SX/G シリーズ

c_dsum

積和計算 (実ベクトル)

<code>ierr = c_dsum(a, b, n, ia, ib, &sum);</code>

1. 機能

n 次元の実ベクトル $\mathbf{a}, \mathbf{b}$ が与えられたとき, 次のような積和を計算する.

$$\sigma = \sum_{i=1}^n a_i b_i$$

ただし, $\mathbf{a}^T = (a_1, a_2, \dots, a_n)$, $\mathbf{b}^T = (b_1, b_2, \dots, b_n)$.

2. 引数

呼出し形式:

```
ierr = c_dsum(a, b, n, ia, ib, &sum);
```

引数の説明:

a	double a[Alen]	入力	ベクトル $\mathbf{a}$. $Alen = \mathbf{a} * n$.
b	double b[Blen]	入力	ベクトル $\mathbf{b}$. $Blen = \mathbf{b} * n$.
n	int	入力	次元 n .
ia	int	入力	ベクトル $\mathbf{a}$ の要素の間隔 ($\neq 0$). 通常 1 と指定する. (“使用上の注意”b)参照)
ib	int	入力	ベクトル $\mathbf{b}$ の要素の間隔 ($\neq 0$). 通常 1 と指定する. (“使用上の注意”b)参照)
sum	double	出力	積和値 σ .

3. 使用上の注意

a) 本関数の目的

数値計算でしばしば現れる積和計算は, その浮動小数点演算における誤差解析の理論より, 精度を上げて計算することが, 有効数字を保持する上で必要であることが知られている. 本関数では, 使用する計算機に依存した最適な方法により精度をあげて計算するものである.

b) 配列 $\mathbf{a}, \mathbf{b}$ 内のデータ間隔

ベクトル $\mathbf{a}$ の要素が, 配列 $\mathbf{a}$ 上に, 間隔 p で格納されている場合, $ia=p$ と指定すればよい. 同様に, ベクトル $\mathbf{b}$ において間隔 q ならば, $ib=q$ と指定すればよい. なお, $p, q < 0$ の場合は, 配列 $\mathbf{a}, \mathbf{b}$ の与え方に注意すること.

4. 使用例

行列の行と列の積和を計算する.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h" /* standard C-SSL header file */

#define NMAX 100

MAIN__()
{
    int n, i, j, ia, ib;
    double eps, err, sum, sum2;
    double mat[NMAX][NMAX], *a, *b;
```

```
/* initialize matrix */
n = NMAX;
for (i=0;i<n;i++)
    for (j=0;j<n;j++)
        mat[i][j] = i+j+1;
/* calculate the product sum of row 6 and column 3 */
a = &mat[6][0];
ia = 1;
b = &mat[0][3];
ib = NMAX;
c_dsum(a, b, n, ia, ib, &sum);
/* check sum */
eps = 1e-6;
sum2 = 0;
for (i=0;i<n;i++)
    sum2 = sum2 + a[i*ia]*b[i*ib];
err = fabs((sum2-sum)/sum);
if (err > eps) {
    printf("WARNING: result inaccurate\n");
    exit(1);
}
printf("Result OK\n");
return(0);
}
```

c_iradix

浮動小数点演算の基底

<code>iradix = c_iradix();</code>

1. 機能

関数の値は、浮動小数点演算の基底の値が設定される。

2. 引数

呼出し形式:

```
iradix = c_iradix();
```

3. 使用上の注意

本関数の復帰値は整数型である。復帰値には以下のような値が設定される。

演算方法	iradix	適用例
2進法	iradix = 2	VPP シリーズ FM シリーズ SX/G シリーズ
16進法	iradix = 16	M シリーズ S シリーズ SX/G 200 シリーズ

付録 B 参考文献一覧

SSL II で使用している理論において重要な文献を示す. 本文の中で参照する場合には[10]のように文献の番号を記述している.

- [1] Ahlberg, J. H., Nilson, E. N. and Walsh, J. L. *The Theory of Splines and Their Applications*, Academic Press, 1967.
- [2] Akima, H. "A New Method of Interpolation and Smooth Curve Fitting Based on Local Procedure", *Journal of the ACM*, Vol.17, No.41, pp.589-602, 1970.
- [3] Akima, H. "Bivariate Interpolation and Smooth Surface Fitting Based on Local Procedures", *Comm. ACM*, Vol.17, No.1, pp.26-31, 1974.
- [4] "Algorithm 334, Normal Random Deviates", *Comm. ACM*, Vol.11, p.498, (July 1968).
- [5] Amestoy, P., Dayde, M. and Duff, I. "Use of computational kernels in the solution of full and sparse linear equations", *Parallel and Distributed Algorithms*, (M. Cosnard, Y. Robert, Q. Quinton and M. Raynal eds.), pp.13-19, North-Holland, 1989.
- [6] Anderson, S. L. "Random number generators on vector supercomputers and other advanced architectures", *SIAM Rev*, Vol.32 (1990), pp.221-251.
- [7] Bowdler, H. J., Martin, R. S. and Wilkinson, J. H. "Solution of Real and Complex Systems of Linear Equations", *Linear Algebra Handbook for Automatic Computation*, Vol.2, pp.93-110, Springer-Verlag, 1971.
- [8] Brent, R. P. A Fast Vectorised Implementation of Wallace's Normal Random Number Generator, Technical Report, Computer Sciences Laboratory, Australian National University, To appear.
- [9] Brent, R. P. *Algorithms for Minimization without Derivatives*, Prentice-Hall, pp.47-60, 1973.
- [10] Brent, R. P. "Fast normal random number generators on vector processors", Technical Report TR-CS-93-04, Computer Sciences Laboratory, Australian National University, Canberra, March 1993.
- [11] Brent, R. P. "Uniform random number generators for supercomputers", *Proc. Fifth Australian Supercomputer Conference*, Melbourne, Dec. 1992, pp.95-104.
- [12] Brent, R. P. "Uniform random number generators for vector and parallel computers", Report TR-CS-92-02, Computer Sciences Laboratory, Australian National University, Canberra, March 1992.
- [13] Brezinski, C. *Acceleration de la Convergence en Analyse Numerique*, Lecture Notes in Mathematics 584, Springer-Verlag, pp.136-159, 1977.
- [14] Bromwich, T. J. *Introduction to the Theory of Infinite Series*, Macmillan, 1926.
- [15] Bunch, J. R. and Kaufman, L. "Some Stable Methods for Calculation Inertia and Solving Symmetric. Linear Systems", *Math. Comp.*, Vol.13, No.137, January 1977, pp.163-179.
- [16] Burden, R.L. and Faires, J. D. *Numerical Analysis*, Fifth Edition, PWS Publishing Company, 1993.
- [17] Burrus, C. S., and Eschenbacher, P. W. "An In Place, In-Order Prime Factor FFT Algorithm", *IEEE Trans. on Acous, Speech, Signal Processing*, Vol.ASSP-29, No.4, pp.806-817, August 1981.
- [18] Businger, P. and Golub, G. H. "Linear Least Squares Solutions by Householder Transformations", *Linear Algebra Handbook for Automatic Computation*, Vol.2, pp.111-118, Springer-Verlag, 1971.
- [19] Byrne, G. D. and Hindmarsh, A. C. "A Polyalgorithm for the Numerical Solution of Ordinary Differential Equations", *ACM Trans. Math. Soft.*, Vol.1, No.1, pp.71-96, March 1975.
- [20] Chui, C.K. "Wavelets and Splines" in *Advances in Numerical. Analysis*, Vol. II, Oxford Scientific Publishers, 1992.
- [21] Clenshaw, C.W. and Curtis, A. R. "A method for numerical integration on an automatic computer", *Numer. Math.*, Vol.2, 1960, pp.197-205.
- [22] Cody, W. J. and Thacher Jr, C. H. "Chebyshev Approximations for the Exponential Integral $E_1(x)$ ", *Math. Comp.*, Vol.23, pp.289-303, Apr. 1969.
- [23] Cody, W. J. and Thacher Jr, C. H. "Rational Chebyshev Approximations for the Exponential Integral $E_1(x)$ ", *Math. Comp.*, Vol.22, pp.641-649, July 1968.

- [24] Cosnard, M. Y. "A Comparison of Four Methods for Solving Systems of Nonlinear Equations", Cornell University Computer Science Technical Report TR75-248, 1975.
- [25] Cullum, J.K. and Willoughby, R.A. *Lanczos algorithm for large symmetric eigenvalue computations*, Birkhauser, 1985.
- [26] Dantzig, G. *Linear Programming and Extensions*, Princeton University Press, 1963.
- [27] Daubechies, I. *Ten lectures on wavelets*, SIAM, 1992.
- [28] Davis, P. J. and Rabinowitz, P. *Methods of Numerical Integration*, Academic Press, 1975.
- [29] De Boor, C. "CADRE: An algorithm for numerical quadrature", in *Mathematical Software* (ed. J. R. Rice), Academic Press, 1971, pp.417-449.
- [30] De Boor, C. "On Calculating with B-splines", *J. Approx. Theory*, Vol.6, 1972, pp.50-62.
- [31] Demmel, J. and Kahan, W. "Accurate singular values of bidiagonal matrices", *SIAM J. Sci. Comput*, Vol.11, pp.873-912, 1990.
- [32] Ferrenberg, A. M., Landau, D. P. and Wong, Y. J. "Monte Carlo simulations: Hidden errors from "good" random number generators", *Phys. Rev. Lett*, Vol.69, (1992), pp.3382-3384.
- [33] Fletcher, R. "Fortran subroutines for minimization by quasi-Newton methods", Report R7125 AERE, Harwell, England, 1972.
- [34] Forsythe, G. E. and Moler, C. B. *Computer Solution of Linear Algebraic Systems*, Prentice-Hall Inc., 1967.
- [35] Fox, L. and Parker, A. B. *Chebyshev Polynomials in Numerical Analysis*, Oxford University Press, 1972.
- [36] Freund, R. "A transpose-free quasi-minimal residual algorithm for nonhermitian linear systems", *SIAM J. Sci. Comput*. Vol. 14, pp.470-482, 1993.
- [37] Freund, R. and Nachtigal, N., "QMR: a quasi minimal residual method for non-Hermitian linear systems", *Numerische Mathematik* Vol. 60, pp. 315-339, 1991.
- [38] Garside, G. R., Jarratt, P. and Mack, C. "A New Method for Solving Polynomial Equations", *Computer Journal*, Vol.11, pp.87-90, 1968.
- [39] Gear, C. W. *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice Hall, 1971.
- [40] Gershinsky, M. and Levine, D. "Aitken-Hermite Interpolation", *Journal of the ACM*, Vol.11, No.3, 1964, pp.352-356.
- [41] Golub, G. H. and Reinsch, C. "Singular Value Decomposition and Least Squares Solutions", *Linear Algebra Handbook for Automatic Computation*, Vol.2, pp.134-151, Springer-Verlag, 1971.
- [42] Golub, G. H. and Van Loan, C. *Matrix computations*, The Johns Hopkins University Press, Baltimore, 1989.
- [43] Graps, A. "An introduction to wavelets", *IEEE Computational Science and Engineering*, Vol. 9, No.2, 1995.
- [44] Greville, T. N. E. "Spline Function, Interpolation and Numerical Quadrature", *Mathematical Methods for Digital Computers*, Vol.2, John-Wiley & Sons, 1967, pp.156-168.
- [45] Gutknecht, M. H. Variants of BiCGStab for matrices with complex spectrum, IPS Research report No. 91-14, 1991.
- [46] Hamming, R. W. "Stable Predictor Corrector Methods for Ordinary Differential Equations", *Journal of the ACM*, Vol.6, 1956, pp.37-47.
- [47] Hamming, W. *Numerical Methods for Scientists and Engineers*, McGraw-Hill, 1973.
- [48] Hart, J. F. *Complete Elliptic Integrals*, John Wiley & Sons, 1968.
- [49] Hegland, M. "A self-sorting in-place fast Fourier transform algorithm suitable for vector and parallel processing", *Numerische Mathematik*, 1994.
- [50] Hegland, M. "An Implementation of Multiple and Multivariate Fourier Transforms on Vector Processors", *SIAM J. Sci. Comput*, Vol.16, No.2, pp.271-288, March 1995.
- [51] Hegland, M. "On the parallel solution of tridiagonal systems by wrap-around partitioning and incomplete LU factorization", *Numerische Mathematik*, Vol. 59, No.5, pp.453-472, 1991.
- [52] Heringa, J.R, Blöte, H.W. J. and Compagner, A. "New primitive trinomials of Mersenne-exponent degrees for random-number generation", *International J. of Modern Physics C* 3(1992), pp.561-564.
- [53] Hestenes, M. R. and Stiefel, E. Methods of conjugate gradients for solving linear systems. J. Res. Nat Bur. Standards, 49:409-433, 1952.
- [54] Hildebrand, F. B. *Introduction to Numerical Analysis, Second Edition*. McGraw-Hill, 1974.

-
- [55] 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, April 1977.
 - [56] Hosono, T. "Numerical inversion of Laplace Transform and some applications to wave optics", *International U.R.S.I. – Symposium 1980 on Electromagnetic Waves*, Munich, 1980.
 - [57] Jackson, K. R. Enright, W. H. and Hull, T. E. "A theoretical criterion for comparing Runge-Kutta formulas", *SIAM J. Numer. Anal.*, Vol.15, No.3, 1978, pp.618-641.
 - [58] James, F. "A review of pseudo-random number generators", *Computer Physics Communication*, Vol.60 (1990), pp.329-344.
 - [59] Jenkins, M. A. "Algorithm 493 Zeros of a real polynomial", *ACM Trans. Math. Soft.*, Vol.1, No.2, pp.178-187, 1975.
 - [60] Jenkins, M. A. and Traub, J. F. "A three-stage algorithm for real polynomials using quadratic iteration", *SIAM J. Numer. Anal.* Vol.17, pp.545-566, 1970.
 - [61] Jennings, A. *Matrix Computation for Engineers and Scientists*, J.Wiley, pp.301-310, 1977.
 - [62] Kahaner, D. K. "Comparison of numerical quadrature formulas", in *Mathematical Software* (J. R. Rice ed.), Academic Press, 1971, pp.229-259.
 - [63] Kincaid, D. and Oppe, T. "ITPACK on supercomputers", *Numerical methods*, Lecture Notes in Mathematics 1005, Springer-Verlag, (1982).
 - [64] Knuth, D.E. *The Art of Computer Programming*, Volume 2: Seminumerical Algorithms (second edition). Addison-Wesley, Menlo Park, 1981, Sec. 3.4.1, Algorithm P.
 - [65] Lambert, J. D. *Computational Methods in Ordinary Differential Equations*, Wiley, 1973.
 - [66] Leyk, Z. "Modified generalized conjugate residuals for nonsymmetric systems of linear equations", *Proceedings of the 6th Biennial Conference on Computational Techniques and Applications: CTAC93*, D. Siewart, H. Gardner and D. Singleton. eds., World Scientific, 1994, pp.338-344. Also published as CMA Research Report CMA-MR33-93, Australian National University, 1993.
 - [67] Lyness, J. N. "Notes on the Adaptive Simpson Quadrature Routine", *Journal of the ACM*, Vol.16, No.3, 1969, pp.483-495.
 - [68] Madsen, N. K., Rodrigue, G. H. and Karush, J.I. "Matrix multiplication by diagonals on a vector/parallel processor", *Information Processing Letters*, Vol.5, 1976, pp.41-45.
 - [69] Marquardt, D.W. "An algorithm for least squares estimation of nonlinear parameters", *SIAM J. Appl. Math.*, Vol.11, pp.431-441, 1963.
 - [70] Marsaglia, G. "A current view of random number generators", *Computer Science and Statistics: The Interface* (L. Billard ed.), Elsevier Science Publishers B.V., 1985, pp.3-10.
 - [71] Martin, R. S., Peters, G. and Wilkinson, J. H. "Symmetric Decomposition of a Positive Definite Matrix", *Linear Algebra Handbook for Automatic Computation*, Vol.2, pp.9-30, Springer-Verlag, 1971.
 - [72] Martin, R. S. and Wilkinson, J. H. "Solution of Symmetric and Unsymmetric Band Equations and the Calculations of Eigenvectors of Band Matrices", *Linear Algebra Handbook for Automatic Computation*, Vol.2, pp 70-92, Springer-Verlag, 1971.
 - [73] Martin, R. S. and Wilkinson, J. H. "Symmetric Decomposition of Positive Definite Band Matrices", *Linear Algebra Handbook for Automatic Computation*, Vol.2, pp 50-56, Springer-Verlag, 1971.
 - [74] Mueller, D. J. "Householder's Method for Complex Matrices and Eigensystems of Hermitian Matrices", *Numer.Math.* 8, pp.72-92, 1966.
 - [75] Nayler, T R. *Computer Simulation Techniques*, John Wiley & Sons, 1966, pp.43-67.
 - [76] Ninomiya, I. "Improvements of Adaptive Newton-Cotes Quadrature Methods, *Journal of Information Processing*, Vol.3, No.3, 1980, pp.162-170.
 - [77] Oppe, T., Joubert, W. and Kincaid, D. "An overview of NSPCG: a nonsymmetric preconditioned conjugate gradient package", *Computer physics communications*, Vol.53, p283, (1989).
 - [78] Oppe, T. C. and Kincaid, D. R. "Are there interactive BLAS?", *Int. J. Sci. Comput. Modeling*, (to appear).
 - [79] Ortega, J. *Introduction to parallel and vector solution of linear systems*, Plenum Press, 1988.
 - [80] Ortega, J. "The Givens-Householder Method for Symmetric Matrix", *Mathematical Methods for Digital Computers*, Vol.2, John Wiley & Sons, pp.94-115, 1967.
-

- [81] Osborne, M. "Computing the eigenvalues of tridiagonal matrices on parallel vector processors", Mathematics Research Report No. MRR 044-94, Australian National University, 1994.
- [82] Osborne, M. "Nonlinear least squares-the Levenberg algorithm revisited", *J. of the Australian Mathematical Society*, Vol.19, pp.343-357, 1976.
- [83] Parlett, B. N. and Wang, Y. "The Influence of The Compiler on The Cost of Mathematical Software-- in Particular on The Cost of Triangular Factorization, *ACM Trans. Math. Soft.*, Vol.1, No.1, pp.35-46, March, 1975.
- [84] Peters, G. and Wilkinson, J.H. "Eigenvalues of $Ax=\lambda Bx$ with Band Symmetric A and B", *Comp. J.* Vol.12, pp.398-404, 1969.
- [85] Pike, M. C. "Algorithm 267, Random Normal Deviate", *Comm. ACM*, Vol.8 (Oct.1965), p.606.
- [86] Powell, M. J. D. "A fast algorithm for nonlinearly constrained optimization calculations", *Proceedings of the 1977 Dundee Conference on Numerical Analysis*, Lecture Notes in Mathematics, Springer-Verlag, 1978.
- [87] Powell, M. J. D. et al. "The Watchdog technique for forcing convergence in algorithms for constrained optimization", presented at the *Tenth International Symposium on Mathematical Programming*, Montreal, 1979.
- [88] Press, W. H., Teukolsky, S. A., Vetterling, W. T. and Flannery, B. P. *Numerical Recipes in C*, Second Edition, Cambridge University Press, 1995.
- [89] Ralston, A. *A First Course in Numerical Analysis*, McGraw-Hill, 1965.
- [90] Rice, J.R. and Boisvert, R.F. *Solving Elliptic Problems Using ELLPACK*, Springer-Verlag, 1985.
- [91] Romanelli, M. J. "Runge-Kutta Methods for Solution of Ordinary Differential Equations", *Mathematical Methods for Digital Computers*, Vol.2, pp.110-120, John Wiley & Sons, 1967.
- [92] Saad, Y. and Schultz, M. H. "GMRES. A generalized minimal residual algorithm for solving non-symmetric linear systems", *SIAM J. Sci. Stat. Comp.*, Vol.7, 1986, pp.856-869.
- [93] Salesin, D.H., Stollnitz, E.J. and DeRose, T.D. "Wavelets for computer graphics: A primer, part 1 and 2", *IEEE Computer Graphics and Applications*, Vol.15, 1995.
- [94] Shampine, L. F. and Gordon, M. K. *Computer Solution of Ordinary Differential Equations*, Freeman, 1975.
- [95] Shanno, D. F. and Phua, K. H. "Numerical comparison of several variable metric algorithms", *J. of Optimization Theory and Applications*, Vol.25, No.4, 1978.
- [96] Simon, H. "Bisection is not optimal on vector processors", *SIAM J. Sci. Stat. Comp.*, Vol.10, pp.205-209, 1989.
- [97] Simonnard, M. (translated by W. S. Jeweli) *Linear Programming*, Prentice -Hall, 1966.
- [98] Singleton, C. "An Algorithm for Computing The Mixed Radix Fast Fourier Transform", *IEEE Transactions on Audio and Electroacoustics*, Vol.AU-17, No.2, pp.93-103, June 1969.
- [99] Singleton, R. C. "An ALGOL Convolution Procedure Based On The Fast Fourier Transform", *Comm. ACM*, Vol.12, No.3, pp.179-184, March 1969.
- [100] Singleton, R. C. "On Computing The Fast Fourier Transform", *Comm. ACM*, Vol.10, No.10, pp.647-654, October 1967.
- [101] Sleijpen, G. and Fokkema, D. BICG_STAB(L) for linear equations involving unsymmetric matrices with complex spectrum, *Electronic Transactions on Numerical Analysis*, Vol 1, p11-32, 1993.
- [102] Sleijpen, G. L. G. , van der Vorst, H. A.. and Fokkema, D. R. BiCGSTAB(l) and other hybrid Bi-CG methods. *Numerical Algorithms*, 7:75-109, 1994.
- [103] Smith, B. T., Boyle, J. M., Garbow, B. S., Ikebe, Y., Kiema, V. C. and Moler, C. B. *Matrix Eigensystem Routine-EISPACK Guide 2nd edition*, Lecture Notes in Computer Science 6, Springer-Verlag, 1976.
- [104] Stone, H. S. "Parallel Tridiagonal Equation Solvers", *ACM Trans. Math. Soft.*, Vol.1, No.4, pp.289-307.
- [105] Strang, G. and Nguyen, T. *Wavelets and filter banks*, Wellesley-Cambridge Press, 1996.
- [106] Streck, A. J. "On the Calculation of the Inverse of Error Function", *Math. Comp.*, Vol.22, 1968.
- [107] Swarztrauber, P. N., "Bluestein's FFTs for arbitrary N on the hypercube", *Parallel Comput.* Vol.17, (1991), pp.607-617, 1975.
- [108] Swarztrauber, P.N. "Vectorizing the FFTs", *Parallel Comput.*, Academic Press, 1982, pp.51-83.
- [109] Takahashi, H. and Mori, M. *Double Exponential Formulas for Numerical Integration*, Publications of R. I. M. S, Kyoto Univ.
- [110] Temperton, C. "Fast Fourier Transforms and Poisson solvers on CRAY-I", *INFOTECH*, 1979.

-
- [111] Traub, J. F. "The Solution of Transcendental Equations", *Mathematical Methods for Digital Computers*, Vol.2, pp.171-184, 1967.
 - [112] Van Der Vorst, H. A. "BCG: A fast and smoothly converging variant of BI-CG for the solution of non-symmetric linear systems", *SIAM J. Sci Statist. Comput.*, 13 p631 1992.
 - [113] Vande Panne, C. *Linear Programming and Related Techniques*, North-Holland, 1971.
 - [114] Verner, J. H. "Explicit Runge-Kutta methods with estimate of the local truncation error", *SIAM J. Numer. Anal.* Vol.15, No.4, 1978, pp.772-790.
 - [115] Wallace, C. S. "Fast Pseudo-Random Generators for Normal and Exponential Variates", *ACM Trans. on Mathematical Software* 22 (1996), 119-127.
 - [116] Weiss, R. *Parameter-Free Iterative Linear Solvers*. Mathematical Research, vol. 97. Akademie Verlag, Berlin, 1996.
 - [117] Wilkinson, J. H. *Rounding Errors in Algebraic Process*, Her Britannic Majesty's Stationary Office, London, 1963.
 - [118] Wilkinson, J. H. *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1965.
 - [119] Wilkinson, J. H. and Reinsch, C. *Linear Algebra Handbook for Automatic Computation*, Vol.2, Springer-Verlag, 1971.
 - [120] Winograd, S. "On computing the discrete Fourier transform", *Math. Comp.*, Vol.32, pp.175-199, Jan 1978.
 - [121] Wynn, P. "Acceleration Techniques for Iterated Vector and Matrix Problems", *Math. Comp.*, Vol.16, pp.301-322, 1962.
 - [122] Yamashita, S. "On the Error Estimation in Floating-point Arithmetic." *Information Processing in Japan* Vol.15, pp.935-939, 1974.
 - [123] 宇野利雄. Bessel 関数. 培風館, 電子計算機のための数値計算法 III, pp.166-186, 1972.
 - [124] 清野武, 松井照子. 低次元代数方程式の解法について, 京都大学大型計算機センター広報, Vol.4, No.9, pp.9-32, 1971.
 - [125] 古林隆. 線形計画法入門. 産業図書, 1980.
 - [126] 島崎眞昭. スーパーコンピュータとプログラミング, 共立出版株式会社, 1989.
 - [127] 戸川隼人. マトリクスの数値計算 3 章 pp.146-210. オーム社, 東京, 1971.
 - [128] 戸川隼人. 有限要素法による振動解析 4 章 pp.156-162, サイエンス社, 1970
 - [129] 鳥居達生. フーリエ変換サブルーチンパッケージの作成 (その 1), (その 2), 名古屋大学大型計算機センターニュース, Vol.10, No.2, 1979
 - [130] 鳥居達生, 長谷川武光, 二宮市三. 等差数列的に標本点を増す補間的自動積分法, 情報処理 Vol.19, No.3, pp.248-255, 1978
 - [131] 鳥居達生. 高速 sine 変換, cosine 変換とその数値積分への応用, 情報処理, Vol.15, No.9, pp.670-679, 1974
 - [132] 二宮市三. 代数方程式の GJM 解法について, 情報処理学会予稿集, p.33, 1969.
 - [133] 二宮市三. 漸化式による Bessel 関数の計算, 培風館, 電子計算機のための数値計算法 II, pp.103-120, 1967.
 - [134] 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, No.5, 1980, pp.504-512.
 - [135] 平岩健三. ビットベクトルとインダイレクトベクトルを持つパイプライン計算機上の FFT のための新しい並列算法, 情報処理学会論文誌, Vol.21, No.2, 1980.
 - [136] 松浦俊彦, 三浦謙一. スーパーコンピュータのデータ編集機能と FFT の高速化, 情報処理学会第 28 回全国大会, 1984.
 - [137] 三上次郎, 伊奈博, 秋田典伸, 山下真一郎. ベクトル計算機における FFT アルゴリズム, 情報処理学会第 28 回全国大会, 1984.
 - [138] 宮竹修, 脇本和昌. 乱数とモンテカルロ法 (森北出版, 1978).
 - [139] 森正武. 曲線と曲面. 教育出版, 1974.
 - [140] 森口繁一, 宇田川銈久, 一松信. 数学公式 III — 特殊関数 —, 岩波全書, 1967.
 - [141] 吉田年雄, 浅野道雄, 梅野正義, 三木七郎. 漸化式を用いる複素変数のベッセル関数 $I_n(z)$ の数値計算, 情報処理, Vol.14, No.1, pp.23-29, Jan.1973.
 - [142] 吉田年雄, 二宮市三. τ -method による複素変数のベッセル関数 $K_n(z)$ の数値計算, 情報処理, Vol.14, No.8, pp.569-575, Aug. 1973.
 - [143] 吉田年雄, 二宮市三. x が小さい場合の変形ベッセル関数 $K_\nu(x)$ の計算, 情報処理学会論文誌, Vol.21, No.3, pp.238-245, May. 1980.
-

索引

あ

アダムス法.....469, 475
安定化基本相似変換.....32, 264, 267

い

1 次元.....63
1 次元 FFT.....577, 654
一様分布.....691
一様乱数.....691
一般逆行列.....28, 29, 338
一般行列の格納方法.....8, 261, 294
一般固有値.....331
一般固有値問題.....38, 341, 343, 603
一般スパース行列の格納方法.....13

う

ウェーブレットフィルター.....744
ウェーブレット変換.....535, 538

え

エイトケン・エルミート補間.....86
エイトケン・ラグランジュ補間.....89
エルミート行列.....349, 606
エルミート行列の格納方法.....10
エルミート行列の固有値固有ベクトル.....35
エルミート行列用圧縮格納法.....10

お

オイラー定数.....181, 183, 185, 228, 230, 232
重み関数値.....195, 217, 394

か

階数.....29
改訂準ニュートン法.....44, 432
改訂シンプレックス法.....48, 405
改訂マルカート法.....459, 462
ガウス消去法.....22, 419, 555, 618
格納方法の変換.....261
可能基底解.....405
関数二乗和.....459
関数二乗和の極小化.....462
関数入力.....100, 104, 107, 110, 120
関数名.....2
観測値.....195, 217, 394
ガンマ関数.....79

き

ギア法.....475
キザミ制御.....74
擬似乱数.....81
逆 2 次元補間法.....530
逆行列.....21, 279, 648
逆誤差関数.....364
逆正規分布関数.....372
逆反復法 211, 224, 271, 349, 357, 515, 603, 606, 711, 714
既約優対角.....636, 639, 642, 645
既約優対角性.....24
逆余誤差関数.....366
逆余正規分布関数.....374
級数.....50, 59
行の均衡化.....23
行列格納方法の変換.....19, 259, 290, 292, 294, 297
行列式.....21
行列操作.....20
行列の差.....498, 511
行列の積.....430, 440, 442, 662
行列の和.....84, 123
極小化.....397, 432, 459
極値問題.....43
近似.....49, 58
近似的に多重.....608, 715

く

クラウト法.....22, 276, 391
クレンショー・カーチス型積分法.....97
クレンショー・カーチス型積分法.....110

け

計算機定数.....16
原関数.....64

こ

格子点.....146
誤差関数.....79
固定節点.....195
固有値.....513, 603
固有値固有ベクトル.....30
固有ベクトル.....513
混合基底.....655
コンディションコード.....6
根の公式.....40, 289, 403, 489

さ

サイクリック・リダクション法..25, 636, 639, 642, 645
最小二乗解.....27, 385

最小二乗近似	49
最小二乗近似多項式	58, 394, 507, 509
最小二乗最小ノルム解	28, 388
最適解	405
作業領域	3
差分商	52
三角関数の引数の上限値	16
三角関数表	64
残差二乗和	199, 203
残差ベクトル	426
残差ベクトルの計算	424
三重対角化	606, 714

し

ジェニングス法	36, 214, 331
ジェンキンス・トラウプの方法	486
指数積分	78, 311
指数分布	748
指数乱数	748
実3項行列の固有値固有ベクトル	37
実行列と実ベクトルの積	424
実行列の固有値固有ベクトル	31
実係数2次方程式	488
実係数高次代数方程式	486
実係数低次代数方程式	403
実対称行列	714
実対称行列と実ベクトルの積	444
実対称行列の一般固有値固有ベクトル	38
実対称行列の固有値固有ベクトル	33
実対称バンド行列	224, 331
実対称バンド行列と実ベクトルの積	438
実対称バンド行列の一般固有値固有ベクトル	39
実対称バンド行列の固有値固有ベクトル	36
実超越方程式	530, 532
実バンド行列と実ベクトルの積	650
実ヘッセンベルグ行列	346, 351, 355, 357
自動積分ルーチン	71
弱スティフ	471
周期型	645
修正一般化共役残差法	597, 600
出力引数	3
主部	3
準エルミート補間	58
準エルミート補間式	95
準ニュートン法	44, 435
条件数	28
常微分方程式	73
初期可能基底	405
振動型関数	112
シンプソン則	500
シンプレックス法	48

す

数値積分	70, 155, 165, 217
数値的に多重	607, 715
数値微分	69, 155, 165, 217
スケールリング	23

スティフ	75, 475
スパース行列とベクトルの積	675, 677
スパース行列の連立1次方程式	25

せ

正規化	14, 64, 284, 466
正規分布	685
正規分布関数	80, 446
制御部	3
正規乱数	685, 688
正弦・余弦積分	78
正弦積分	502
正弦フレネル積分	496
整合寸法	4, 9
正値行列	214
正値対称行列用圧縮格納法	10
正値対称スパース行列	579, 583
正値対称スパース行列の格納方法	14
正値対称バンド行列用圧縮格納法	12
正の最小値	16, 757
正の最大値	16, 756
制約条件	450
制約付き多変数関数の極小化	49
制約なし関数二乗和の極小化	46
制約なし多変数関数の極小化	44
積和計算	754, 759
節点追加方式	199, 203
線形計画問題	47, 405
線型計算	19
線形微分方程式	75
線型補間法	530

そ

素因子	707
素因子フーリエ変換	593
像関数	64

た

第1種0次ベッセル関数	173
第1種0次変形ベッセル関数	128
第1種1次ベッセル関数	175
第1種1次変形ベッセル関数	130
第1種完全楕円積分	253
第1種実数次ベッセル関数	179, 240
第1種実数次変形ベッセル関数	171
第1種整数次ベッセル関数	177, 238
第1種整数次変形ベッセル関数	169, 236
第1種不完全ガンマ関数	368
第2種0次ベッセル関数	228
第2種0次変形ベッセル関数	181
第2種1次ベッセル関数	230
第2種1次変形ベッセル関数	183
第2種完全楕円積分	255
第2種実数次ベッセル関数	234
第2種実数次変形ベッセル関数	187
第2種整数次ベッセル関数	232, 249

第2種整数変形ベッセル関数	185, 243
第2種不完全ガンマ関数	370
対角形式格納法	13, 15, 544, 579, 597, 675
対角相似変換	189, 246
対角ユニタリ変換	526
台形公式	60
台形則	517
対称行列の格納方法	9
対称行列用圧縮格納法	9, 261, 292, 294, 297
対称スパース行列の固有値固有ベクトル	37
対称バンド行列用圧縮格納法	12, 259, 290, 292, 297
代数的手法	40
対数特異点	101, 121
代数特異点	101, 121
代数方程式	40
楕円積分	78
多次元積分	115
多次元配列	4
多次元フーリエ変換	655
多次元有限領域積分	110
多重固有値	607, 715
多重変換	655
多変数関数	432
多変数関数の極小化	435

ち

チェビシェフ級数	59, 302, 313, 335, 361
逐次代入法	403
中点公式	61
超越方程式	41
直接法	21
直和分解	34
直交相似変換	226

て

ディリクレ型	639
データの格納方法	8
適応型ニュートン・コーツ9点則	120

と

等間隔離散点	500, 507
等式制約	450
特異値	30
特異値分解	28, 125
特異値分解法	338, 388
特殊関数	77
トランスポジションベクトル	23, 276, 279, 541, 648

に

二項分布	746
二項乱数	746
二重指数関数型積分公式	100, 104, 107, 115
入出力引数	3
ニュートン法	42, 44, 403
入力引数	3

の

ノイマン型	642

は

バイセクション法	33, 208, 211, 349, 515
パウエル法	450
ハウスホルダー変換	385
ハウスホルダー法	31, 33, 349, 351, 524, 526
バンド行列の格納方法	11
バンド行列用圧縮格納法	11
反復的手法	40
反復法	25

ひ

ピーク型の関数	121
引数	3
非スティフ	471
非線形計画問題	49, 450
非線形計算	40
非線形最小二乗解	46
非線形微分方程式	76
非対称又は不定値のスパース行列	544, 547, 597, 600
微分方程式	73
ピボッティング	23
標準固有値問題	343
標準ヘッダファイル	4
標本数	63

ふ

フェラリー法	403
不完全コレスキー分解による前処理	579, 583
複素QR法	32
複素行列	276, 279
複素行列と複素ベクトルの積	426
複素行列の固有値固有ベクトル	32
複素係数2次方程式	288
複素係数高次代数方程式	274
複素数	6
複素超越方程式	299
複素ヘッセンベルグ行列	32, 264, 267, 269
不自然な屈曲点	93, 95
不等間隔離散点	509, 517
不等式制約	450
部分ピボッティング	276, 282, 541
フレネル積分	78
不連続点	121
ブレント法	455, 530
ブロッキングLU分解法	541, 615
ブロック対角ピボッティング手法	410, 413, 490, 504

へ

ベアストウ法	403
平滑化	49, 58, 217, 507, 509
平滑化係数	199, 203, 217

平衡化..... 31, 189, 246, 252
 並列バイセクション法..... 603, 711
 ベクトル計算機..... 7
 ヘシアン行列..... 44, 432
 ベッセル関数..... 79
 ヘッセンベルグ行列..... 31
 変換..... 60
 変形コレスキー法..... 23, 416, 550, 625, 631, 721

ほ

ポアソン分布..... 750
 ポアソン乱数..... 750
 補間..... 49, 57, 155, 165
 補間係数..... 138, 146

ま

前処理付き CG 法..... 579, 583
 マラー法..... 299, 532
 マルチセクション法..... 606, 714
 丸め誤差の単位..... 16, 758

め

メインプログラム..... 4

も

目的関数..... 405
 戻り値..... 4

や

ヤコビアン..... 450
 ヤコビアン行列..... 42
 ヤラット法..... 274

ゆ

ユーザ関数..... 5

よ

余弦積分..... 286

ら

ラプラス変換..... 64, 376, 379, 382

り

離散型 cosine 変換..... 60, 321, 587
 離散型 sine 変換..... 61, 328, 718
 離散型実フーリエ変換..... 60, 668, 699, 703, 707, 730
 離散型複素フーリエ変換..... 62, 569, 573, 576, 593, 655
 離散点..... 95, 138, 195, 217, 394

る

ルティスハウザー・シュワルツ法..... 211, 226
 ルンゲ・クッタ・ヴァーナー法..... 482

れ

連立 1 階常微分方程式..... 469, 475, 482
 連立 1 次方程式..... 21
 連立非線型方程式..... 42, 455

B

BICGSTAB(*l*)法..... 544, 547
 break down..... 679, 682, 738, 741
 B-spline..... 51, 69
 B-spline 2 次元平滑化式..... 203, 220
 B-spline 2 次元補間式..... 142, 146, 161, 165
 B-spline による補間..... 57
 B-spline 平滑化式..... 195, 199, 217
 B-spline 補間式..... 132, 135, 138, 140, 149, 152, 155, 158

C

cosine 級数..... 304, 317
 cssl.h..... 4

E

ELLPACK 形式格納法..... 13, 14, 547, 583, 600, 677
 Euler 変換..... 65

F

FFT..... 569, 573, 587, 699, 703, 718
 $f_{l_{max}}$ 16, 756
 $f_{l_{min}}$ 16, 757

H

Householder 変換..... 714
 Hurwitz 多項式..... 67, 353

I

icon..... 6

L

Lanczos 法..... 37, 610
 LDL^T 分解..... 550, 553, 621, 623, 721
 Levenberg-Marquardt 法..... 46
 LU 分解..... 276, 279, 541, 555, 648
l 次安定化双対共役勾配法..... 544, 547

M

main..... 4
 MAIN__..... 4
 Marquardt 数..... 46

MDM ^T 分解	192, 428, 490, 504
MGCR 法	597, 600

N

Neumann の前処理	579, 583

Q

QL 法	33, 494, 513, 528
QMR 法	679, 682
QR 分解	24, 634
QR 法	251, 269

S

sine 級数	309, 324
Spline 関数	50
Spline 関数の節点	195, 217
Spline 関数	69
Sturm カウント	37

T

TFQMR 法	738, 741
t_{max}	16

W

Wallace 法	688
余弦フレネル積分	257
余正規分布関数	448

その他

1 次元全無限区間積分	107
1 次元半無限区間積分	104
1 次元有限区間積分	97, 100, 120, 500, 517
1 変数関数	397
1 変数関数の極小化	43, 400
2 基底 FFT	569, 573, 576, 587, 699, 703, 718
2 次元準エルミート補間式	92
2 次補間法	397
2 段 QR 法	31, 306, 355
2 分法	530
2 変数 Spline 関数	55
3 項行列の連立 1 次方程式	24
3 次補間法	400
3 段アルゴリズム	486