FUJITSU

FUJITSU Software

FUJITSU

C-SSL II スレッド並列機能使用手引書

(科学用関数ライブラリ)

J2UL-2573-01Z0(00)
2020年2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 スレッド並列機能 (C-Scientific Subroutine Library II Thread-Parallel Capabilities) の使用方法について記述しています.

C-SSL II スレッド並列機能は, 共用メモリ型スカラ並列計算機システム上で大規模な問題を効率良く計算するための並列アルゴリズムを使用したライブラリです. このため, 多くのルーチンでは呼び出しインタフェースは従来の C-SSL II のインタフェースとは異なります. 本マニュアルでは, それらのルーチンの使用方法について解説します.

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 スレッド並列機能一覧表

C-SSL II スレッド並列機能で提供される関数を機能ごとに分類した一覧表です. 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 スレッド並列機能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 スレッド並列機能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.

- 概要
- C-SSL II スレッド並列機能の一般規約
- C-SSL II スレッド並列機能の利用方法
- データの格納方法

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 スレッド並列機能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 参考文献一覧

C-SSL II スレッド並列機能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 スレッド並列機能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 スレッド並列機能の使用手引書を参照してください.

“FUJITSU SSL II スレッド並列機能使用手引書”

本書では C 言語用の SSL II スレッド並列機能を C-SSL II スレッド並列機能と呼び, Fortran 言語用の SSL II スレッド並列機能は区別のため Fortran SSL II スレッド並列機能と呼んでいます.

C-SSL II スレッド並列機能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 スレッド並列機能は, VPP シリーズ向け並列数値計算ライブラリ SSL II/VPP 用に開発されたコード及びアルゴリズムを全面的もしくは部分的に利用または改造して開発された機能が含まれています. なお, SSL II/VPP はオーストラリア国立大学(The Australian National University: ANU)と富士通株式会社の共同開発によるものです. ANU での開発は, Mike Osborne 教授と Richard Brent 教授により指導され, オーストラリア国立大学スーパーコンピュータセンター長の Bob Gingold 博士が幹事を勤められました. 下記の ANU の研究者の方々が SSL II/VPP の設計と実現に従事されました. 富士通株式会社はこれらの人々の御協力に対して感謝致します.

Professor Richard Peirce Brent
Dr Andrew James Cleary
Dr Murray Leslie Dow
Mr Christopher Robert Dun
Dr Lutz Grosz
Dr David Lawrence Harrar II
Dr Markus Hegland
Ms Judith Helen Jenkinson
Dr Margaret Helen Kahn
Dr Zbigniew Leyk
Mr David John Miron
Professor Michael Robert Osborne
Dr Peter Frederick Price
Dr Stephen Gwyn Roberts
Dr David Barry Singleton
Dr David Edward Stewart
Dr Bing Bing Zhou

本書は旧マニュアル(前版)を元に作成しました. 新しく追加された箇所及び記載内容が旧マニュアルより修正された箇所には, 目次および C-SSL II スレッド並列機能一覧表に＊印を付けてあります.

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
2020 年　2 月 初版	J2UL-2573-01Z0(00)

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.	`a[i][j]`
Times	一般の英字. 数学の関数名.	C-SSL II, Fortran exp, max
Times italic	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.	**Ax=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 **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 $z = 5 + i\,10$ 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 スレッド並列機能一覧表

C-SSL II スレッド並列機能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) 行列演算

b) 連立 1 次方程式 (直接法)

c) 連立１次方程式 (反復法)

関数名	項　目	ページ
c_dm_vcgd	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)	49
c_dm_vcge	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)	54
c_dm_vbcscc	非対称または不定値のスパース行列の連立 1 次方程式 (BICGSTAB(*l*)法, 圧縮列格納法)	28
c_dm_vbcsd	非対称また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)	34
c_dm_vbcse	非対称また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)	37
c_dm_vtfqd	非対称また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)	426
c_dm_vtfqe	非対称また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)	429
c_dm_vamlid	スパースな M-行列の連立 1 次方程式 (代数的マルチレベル反復法[AMLI 法], 対角形式格納法)	20
c_dm_vmlbife	スパース行列の連立 1 次方程式 (不完全ブロック分解に基づくマルチレベル前処理付き反復法, ELLPACK 形式格納法)	141
c_dm_vlcspsxcr1	非エルミート複素対称スパース行列の連立 1 次方程式 (不完全 LDL^T 分解に基づく前処理付き共役直交共役残差法 (*COCR* 法), 対称行列用圧縮行格納法)	104
c_dm_vlspaxcr2	実非対称スパース行列の連立 1 次方程式 (近似逆行列に基づく前処理付きの演繹的次元縮小法 (IDR 法), 圧縮行格納法)	120

d) 微分方程式

関数名	項　目	ページ
c_ dm_vradau5	スティフ連立 1 階常微分方程式/微分代数方程式 (陰的ルンゲ・クッタ法)	177

e) 偏微分方程式の離散化

関数名	項　目	ページ
c_dm_vpde2d	2 次元 2 階偏微分方程式の有限差分法による離散化によるスパース行列の連立 1 次方程式の生成	166
c_dm_vpde3d	3 次元 2 階偏微分方程式の有限差分法による離散化によるスパース行列の連立 1 次方程式の生成	171

f) 逆行列

関数名	項　目	ページ
c_dm_vminv	実行列の逆行列 (ブロック化された Gauss-Jordan 法)	139
c_dm_vcminv	複素行列の逆行列(ブロック化された Gauss-Jordan 法)	64

g) 固有値問題

関数名	項　目	ページ
c_dm_vsevph	実対称行列の固有値・固有ベクトル (三重対角化, マルチセクション法, 逆反復法)	300
c_dm_vhevp	エルミート行列の固有値・固有ベクトル	71
c_dm_vtdevc	実 3 重対角行列の固有値・固有ベクトル	421
c_dm_vgevph	実対称行列の一般化固有値問題 (固有値および固有ベクトル) (三重対角化, マルチセクション法, 逆反復法)	66
c_dm_vtrid	実対称行列の実対称三重対角行列への変換 (ハウスホルダー法)	432
c_dm_vhtrid	エルミート行列の実対称三重対角行列への変換 (ハウスホルダー法)	75
c_dm_vjdhecr	エルミートスパース行列の固有値問題 (Jacobi-Davidson 法，圧縮行格納法)	78
c_ dm_vjdnhcr	複素スパース行列の固有値問題 (Jacobi-Davidson 法，圧縮行格納法)	87

h) フーリエ変換

関数名	項　目	ページ
c_dm_v1dcft	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)	435
c_dm_v1dcft2	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)	438
c_dm_v1dmcft	1 次元多重離散型複素フーリエ変換 (2, 3, 5 及び 7 の混合基底)	440
c_dm_v2dcft	2 次元離散型複素フーリエ変換 (2, 3, 5 及び 7 の混合基底)	450
c_dm_v3dcft	3 次元離散型複素フーリエ変換 (2, 3, 5 及び 7 の混合基底)	457
c_dm_v3dcft2	3 次元離散型複素フーリエ変換 (2, 3, 5 及び 7 の混合基底)	460
c_dm_v1drcf	1 次元離散型実フーリエ変換 (2, 3, 5 及び 7 の混合基底)	443
c_dm_v1drcf2	1 次元離散型実フーリエ変換 (2, 3, 5 及び 7 の混合基底)	447
c_dm_v2drcf	2 次元離散型実フーリエ変換 (2, 3, 5 及び 7 の混合基底)	453
c_dm_v3drcf	3 次元離散型実フーリエ変換 (2, 3, 5 及び 7 の混合基底)	466
c_dm_v3drcf2	3 次元離散型実フーリエ変換 (2, 3, 5 及び 7 の混合基底)	470
c_dm_v3dcpf	3 次元素因子離散型複素フーリエ変換	463

i) 乱数

関数名	項　目	ページ
c_dm_vranu4	一様乱数 [0, 1)の生成	215
c_dm_vranu5	一様乱数[0, 1)の生成 (MRG8)	218
c_dm_vrann3	正規乱数の生成	207
c_dm_vrann4	正規乱数の生成 (Wallace 法)	211

目次

第 1 章　概　説

1.1 概要

C-SSL II スレッド並列機能は, 共有メモリ型スカラ並列計算機上で並列動作する数値計算ライブラリであり, 単一 CPU では処理できないような大規模な問題を並列処理することで効率よく計算するための関数を提供しています.

“スレッド並列”とは, ひとつのプロセスの中でスレッドと呼ばれる“実行の流れ”を複数個発生させ, 各スレッドは共用メモリ内にあるひとつの CPU を使って計算の一部を担います. CPU の個数が発生させたスレッドの個数分だけあれば, 各スレッドは異なる CPU に割り当てられて並列に実行されます. このように, 一つの計算(ただし並列化できる計算)を, 複数スレッドに分担して並列に処理する方式をスレッド並列と呼びます.

C-SSL II スレッド並列機能の各関数は内部でスレッドを複数発生して並列化できる計算を, スレッドで並列に計算するライブラリであると言えます. C-SSL II スレッド並列機能の関数を利用するには, OpenMP C/C++ の実行環境が必要です.

関数内部で発生させるスレッドの個数は, 利用者が OpenMP C/C++の環境変数もしくは実行時ライブラリルーチンで指定できます. したがって, 関数は与えられた任意のスレッド数で動作することができます.

C-SSL II スレッド並列機能は, 倍精度実数型(double)の変数を扱う関数が用意されています. 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ます. また引数や関数の復帰値として使用する整数は int 型を採用しています.

なお, C-SSL II スレッド並列機能は従来の C-SSL II や, ベクトル並列計算機の単一プロセッサー用の C-SSL II/VP とは機能範囲, 関数名及び呼出し方法が異なります.

1.2 C-SSL II スレッド並列機能の一般規約

1.2.1 関数名

C-SSL II スレッド並列機能の各関数名は次のような規則に基づいてつけられています.

- 関数名の先頭の 6 文字は必ず c_dm_v です.
- 3 文字目には倍精度ルーチンであることを示す d の文字がつきます.

図 1.1 に関数名のつけ方の規則を示します.

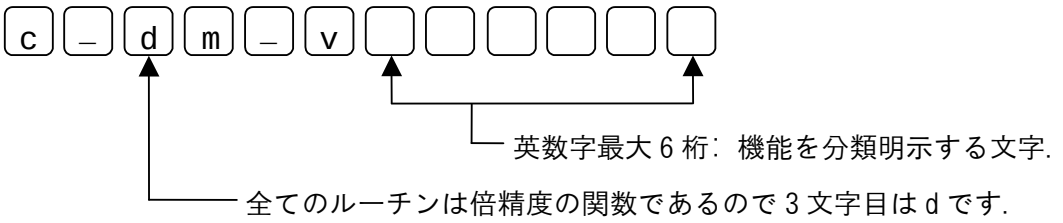


図 1.1 関数名

例えば, c_dm_vlax は実行列の連立 1 次方程式を行うルーチンです.

1.2.2 引数

C-SSL II スレッド並列機能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います.

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らっています.

- 入力のみのスカラ変数は値渡しにより渡します.
- 入出力又は出力を行うスカラ変数は, 変数のアドレスを渡します.

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ます. 基本的には Fortran の変数名の規則に従ってつけられています. 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 で始まります. z は特別に dcomplex 型の変数名の最初に使用しています.

引数は利用者側からみたときの入出力関係に応じて, 次の種類があります.

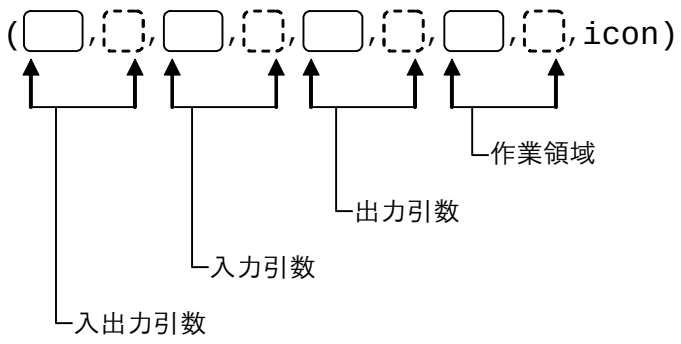
入出力引数 : 引数に値を入力します. また, 演算後, 結果が同一領域上に出力されます.
入力引数 : 引数に値を入力します. 演算後内容は保存されます. なお, 保存されない場合には, “内容は保存されない”と各関数の説明の項で明記されます.
出力引数 : 引数に値が出力されます.
作業領域 : 作業領域として利用されます. 演算後, 内容は原則として意味を持ちません. 連続呼び出しが可能な一部のルーチンでは, 2 回目以降に呼び出す場合に前回の作業領域の内容をそのままにしておく必要があります.

さらに, 引数の内容に応じて, 次の種類を定義します.

主部 : 数値計算の実質的な対象となるデータを格納します. 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します. 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` があります. `icon` とは実行後の状態を示す引数です. 各引数の中では主部が先頭にあり, 制御部が続きます. C-SSL II スレッド並列機能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 スレッド並列機能と同じ並び順となっています.

図 1.2 に引数の並び順を示します.



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

図 1.2 引数の並び順

C-SSL II スレッド並列機能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ます. 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ります.

1.2.3 戻り値

すべての C-SSL II スレッド並列機能の関数は int 型の値を返します. 戻り値は以下のような意味を持ちます.

0 : 計算が正常に終了しました.
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ませんでした.

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ます.

1.2.4 標準ヘッダファイル

C-SSL II スレッド並列機能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ます. 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ます. C-SSL II スレッド並列機能を使用する 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りません

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 スレッド並列機能は内部で Fortran SSL II スレッド並列機能を呼び出しており, 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 スレッド並列機能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です. C-SSL II スレッド並列機能を利用するプログラムでは行列は一般に 2 次元配列に格納します.

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ります.

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

C-SSL II スレッド並列機能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ます. 格納方法の詳細については “1.4 データの格納方法”を参照してください.

1.2.7 複素数

ANSI C には複素数型はサポートされていません. C-SSL II スレッド並列機能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ます.

```
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ます. dcomplex は倍精度実数型を実部(re)と虚部(im)に持ちます.

複素数を利用したプログラムの例を示します.

```
/* include C-SSL II header file */
#include "cssl.h"
#define N1 4000
#define N2 3000
#define KX (N1+1)
#define KY (N2+1)

MAIN__()
{
  int      isn, i, j, icon, ierr;
  dcomplex x[N2][KX], y[N1][KY];

  /* Set up the input data arrays */
#pragma omp parallel for shared(x) private(i,j)
  for(i=0; i<N2; i++) {
    for(j=0; j<N1; j++) {
      x[i][j].re = N1*i+j+1;
      x[i][j].im = 0.0;
    }
  }

  /* Do the forward transform */
  isn = 1;
  ierr = c_dm_v1dcft((dcomplex*)x, KX, (dcomplex*)y, KY, N1, N2, isn, &icon);
  ...
}
```

1.2.8 コンディションコード

C-SSL II スレッド並列機能のルーチンには, 実行後の状態を示す引数 icon が用意されています. 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ります.

コードは 0 から 39999 の値をとりますが, 結果が保証されるか否かに応じて, 表 1.1 のように区分されています.

表 1.1 コンディションコード

コード	意味	結果の保証	区分
0	正常に処理が終了しています.	結果は保証されます.	正常
1～9999	正常に処理が終了していますが, なんらかの補助的な情報を含んでいます.		
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ます.	制限付きで結果は保証されます.	警告
20000～29999	処理の過程で異常が生じ, 処理が打ち切られました.	結果は保証されません.	異常
30000～39999	入力引数にエラーがあったため, 処理が打ち切られました.		

1.3 C-SSL II スレッド並列機能の利用方法

1.3.1 関数の呼出し位置

C-SSL II スレッド並列機能は OpenMP C/C++の関数であり, 利用者プログラムでは OpenMP C/C++のパラレルリージョン(Parallel Region)の外および内から呼び出すことができます. また, OpenMP C/C++文のない逐次プログラムからも呼び出すことができます. さらに自動並列化されたプログラムから呼び出すこともできます.

パラレルリージョンの中から呼び出すときは , パラレルリージョンを実行する各スレッドに対して異なる領域を入出力, 出力および作業域として利用する実引数に指定する必要があります.

上記のいずれの場合も C-SSL II スレッド並列機能を利用する利用者プログラムを翻訳した後結合するとき, fcc コマンドに-Kopenmp オプションを指定します. このオプションにより OpenMP C/C++の実行環境で実行するロードモジュールとなります. 詳細は“C 言語 使用手引書”を参照してください.

1.3.2 スレッド数の指定方法

C-SSL II スレッド並列機能の関数の実行は, 関数の内部のパラレルリージョンで多重スレッドによって並列に行われます. 生成されるスレッド数は, OpenMP C/C++仕様で規定された環境変数 OMP_NUM_THREADS もしくは実行時ライブラリルーチン(以下ライブラリルーチンと略す.) omp_set_num_threads()関数で指定することができます. 普通は, 前者の方法でスレッド数を指定します.

ライブラリルーチンでスレッド数を指定するのは, 直後のパラレルリージョンに対して, 特定のスレッド数で実行させる場合に使います. この方法によって, C-SSL II スレッド並列機能の関数を呼び出す前に指定すれば, 特定のスレッド数で関数を実行させることもできます.

OpenMP C/C++の環境変数および実行時ライブラリルーチンの詳細は “C 言語 使用手引書” および “OpenMP Application Program Interface Version2.5 (May 2005)” を参照してください.

1.3.3 各スレッドのスタック領域の大きさ

関数内部では各スレッドの作業領域をスタック領域に確保しています.生成するスレッドの数を NT, 利用できるメモリ量を M としたとき,各スレッドのスタック領域の大きさとして環境変数 OMP_STACKSIZE に M/(5*NT) 程度を指定して実行することを勧めます.-Nfjomplib オプションが指定されている場合, 環境変数 THREAD_STACK_SIZE にスタック領域の大きさを指定する事もできます. OpenMP C/C++の実行環境で動作するプログラムに対するスタックサイズの指定の詳細に関しては, “C 言語使用手引書” を参照してください.

1.3.4 利用例プログラム

パラレルリージョンの外側からの呼出し

環境変数 OMP_NUM_THREADS を 4 に設定して以下のプログラムを 4 プロセッサーのシステムで実行すると 4000×4000 の実係数行列の連立 1 次方程式を 4 つのスレッドで並列に解くことができます.

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

#define NMAX  (4000)
#define LDA   (NMAX+1)

MAIN__()
{
  int    ip[NMAX];
  int    n, is, isw, icon, ierr, i, j;
  double a[NMAX][LDA], b[NMAX];
  double epsz, c, t, s;

  n = NMAX;
  c = sqrt(2.0/(n+1));
  t = atan(1.0)*4.0/(n+1);

  for(i=1; i<=n; i++) {
    for(j=1; j<=n; j++) {
      a[i-1][j-1] = c*sin(t*i*j);
    }
  }

  for(i=1; i<=n; i++) {
    s = 0.0;
    for(j=1; j<=n; j++) {
      s = s+sin(t*i*j);
    }
    b[i-1] = s*c;
  }

  epsz = 0.0;
  isw  = 1;
  ierr = c_dm_vlax((double*)a, LDA, n, b, epsz, isw, &is, ip, &icon);

  printf("icon = %d, return code = %d\n", icon, ierr);
  printf("n = %d, b[0] = %f, b[n-1] = %f\n", n, b[0], b[n-1]);

}
```

例 2.1 C-SSL II スレッド並列機能のルーチンの利用例

パラレルリージョンの内側からの呼出し

以下の例は, 独立した 2 組の実係数行列の連立 1 次方程式を解きます. 環境変数 OMP_NUM_THREADS を 2 に, そして環境変数 OMP_NESTED を TRUE に設定して以下のプログラムを 4 プロセッサーのシステムで実行します. このとき 4000×4000 の実係数行列の連立 1 次方程式を 2 つのスレッドで , 4200×4200 の実係数行列の連立 1 次方程式を 2 つのスレッドで合計 4 つのスレッドを使って並列に解きます.

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

#define NMAX1  (4000)
#define NMAX2  (4200)
#define LDA1   (NMAX1+1)
#define LDA2   (NMAX2+1)

MAIN__()
{
  int    ip1[NMAX1], ip2[NMAX2], i, j, num;
  int    n1, is1, isw1, icon1, ierr1;
  int    n2, is2, isw2, icon2, ierr2;
  double a1[NMAX1][LDA1], b1[NMAX1];
  double a2[NMAX2][LDA2], b2[NMAX2];
  double epsz1, epsz2, c, t, s;

  n1 = NMAX1;
  c  = sqrt(2.0/(n1+1));
  t  = atan(1.0)*4.0/(n1+1);

  for(i=1; i<=n1; i++) {
    for(j=1; j<=n1; j++) {
      a1[i-1][j-1] = c*sin(t*i*j);
    }
  }

  for(i=1; i<=n1; i++) {
    s = 0.0;
    for(j=1; j<=n1; j++) {
      s = s+sin(t*i*j);
    }
    b1[i-1] = s*c;
  }

  n2 = NMAX2;
  c  = sqrt(2.0/(n2+1));
  t  = atan(1.0)*4.0/(n2+1);

  for(i=1; i<=n2; i++) {
    for(j=1; j<=n2; j++) {
      a2[i-1][j-1] = c*sin(t*i*j);
    }
  }

  for(i=1; i<=n2; i++) {
    s = 0.0;
    for(j=1; j<=n2; j++) {
      s = s+sin(t*i*j);
    }
    b2[i-1] = s*c;
  }

#pragma omp parallel default(shared) private(num)
 {
  num = omp_get_thread_num();

  if(num == 0) {
    epsz1 = 0.0;
    isw1  = 1;
    ierr1 = c_dm_vlax((double*)a1, LDA1, n1, b1, epsz1, isw1, &is1, ip1, &icon1);
  } else {
    epsz2 = 0.0;
    isw2  = 1;
    ierr2 = c_dm_vlax((double*)a2, LDA2, n2, b2, epsz2, isw2, &is2, ip2, &icon2);
```

```
  }
 }

  printf("icon1 = %d, return code = %d\n", icon1, ierr1);
  printf("n1 = %d, b1[0] = %f, b1[n1-1] = %f\n", n1, b1[0], b1[n1-1]);
  printf("icon2 = %d, return code = %d\n", icon2, ierr2);
  printf("n2 = %d, b2[0] = %f, b2[n2-1] = %f\n", n2, b2[0], b2[n2-1]);
}
```

例 2.2 C-SSL II スレッド並列機能のルーチンの利用例

1.4 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説明します.

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ります.

1.4.1 一般行列の格納方法

一般の密行列は図 1.3 に示すように 2 次元配列に格納します.

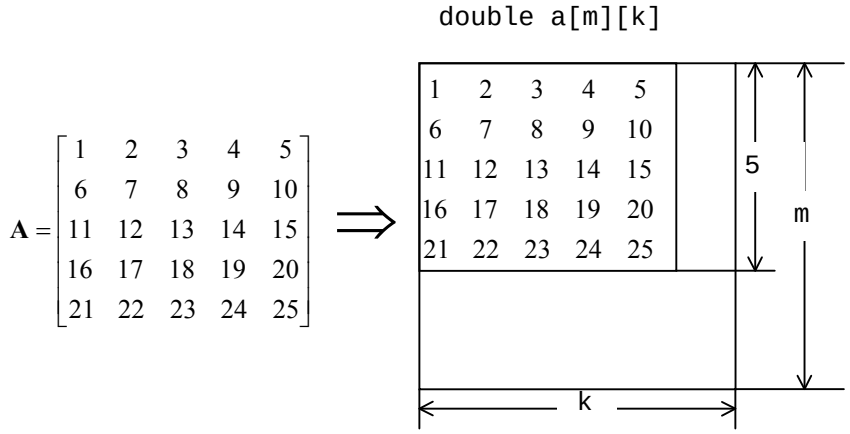


図 1.3 一般行列の格納方法

行列の要素 a_{ij} は配列の要素 a[i-1][j-1]に格納します. 行列の要素の添字は 1 から始まりますが, C 言語の配列の添字は 0 から始まることに注意してください. 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]に格納します.

C-SSL II スレッド並列機能のルーチンでは, 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ます.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ます. 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ます. C-SSL II スレッド並列機能で用いるところの “整合寸法” とは, 2 次元配列 a の列の数 k のことを意味します.

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ます. 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りません.

1.4.2 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ます.

ELLPACK 形式格納法は, 係数行列の各行ベクトルの非零要素のみを圧縮して格納します. 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します.

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ます.

ELLPACK 形式格納法

図 1.4 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[45], [62]参照)といいます.

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

図 1.4 一般行列の ELLPACK 形式格納法

係数行列 **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 列に格納します. 行ベクトルを列方向に格納することに注意してください. さらに, `coef` に格納された非零要素が行列 **A** の何番目の列にあるかを示す値を, 対応する `icol` の配列要素に格納します. `coef` に行列 **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ありません.

この他に, 入力の引数として行列 **A** の行ベクトルの非零要素の最大値 `iwidt` が必要です. **A** の第 i 行の非零要素の数が `iwidt` 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 の値を入れます.

図 1.4 の例では, 行列 **A** の 1 行目には 1 列目と 4 列目に非零要素があります. $a_{11}=1$ より, `coef[0][0]`に 1 を格納し, `icol[0][0]`には 1 を格納します. $a_{14}=2$ より, `coef[1][0]`に 2 を格納し, `icol[1][0]`には 4 を格納します. 行列 **A** の 3 行目の非零要素の数は `iwidt` よりも少ないです. このため `coef[0][2]=5`, `icol[0][2]=3` としますが, `coef[1][2]`には 0 を格納し, `icol[1][2]`には行番号の 3 を格納します.

対角形式格納法

図 1.5 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[52], [59]参照)といいます.

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

図 1.5 一般行列の対角形式格納法

行列 **A** の非零要素の存在する対角方向のベクトルを配列 diag の行方向に格納します. 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, 本マニュアルでは呼びます. 特に, 対角要素からなるベクトルを主対角ベクトルと呼びます.

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

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す. ただし, 次に述べる 1 次元配列 nofst で対応をとります.

nofst[i]には, diag[i][j], j=0, ... , 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します. 上記対角ベクトルで k が距離を表します. 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ります.

1.4.3 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, この順番に格納します.

正値対称な行列 **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ます.

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

ここで,

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

次数 n の連立 1 次方程式 $\mathbf{Ax}=\mathbf{b}$ 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ます.

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

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

正値対称スパース行列の **ELLPACK** 形式格納法

図 1.6 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 ELLPACK 形式の格納方法で格納したものを 1 つの配列 coef にまとめて格納します.

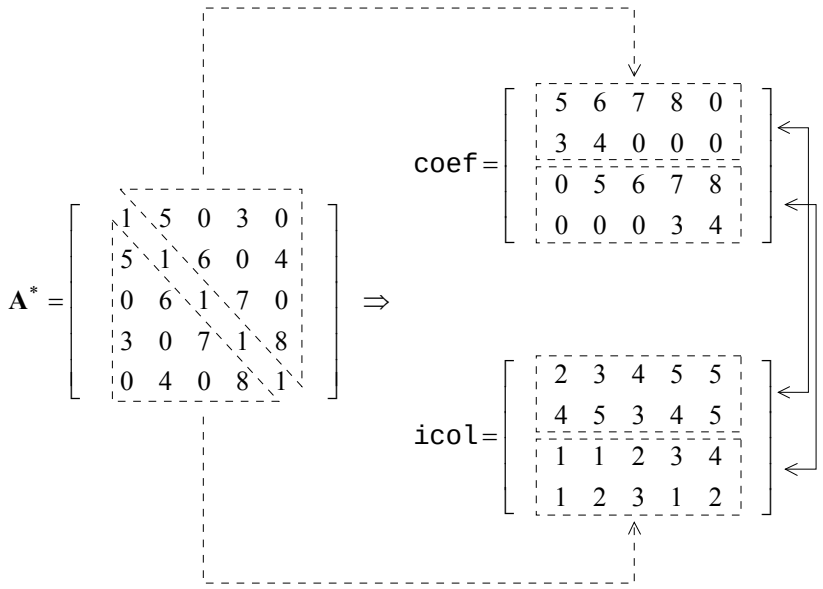


図 1.6 正値対称スパース行列の ELLPACK 形式格納法

上三角行列部分を最初に格納し, その後に下三角行列部分を格納します. 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ありません.

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 とします. 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～nsh–1 行に格納します.下三角行列部分の非零要素は nsh～2*nsh–1 行に格納します. 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します.

図 1.6 の例では, $a_{32} = 6$ より coef[2][2]に 6 を格納し icol[2][2]に 2 を格納します. 同様に, $a_{34} = 7$ より coef[0][2]に 7 を格納し icol[0][2]に 4 を格納します.

正値対称スパース行列の対角形式格納法

図 1.7 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を, 1 つの配列 diag にまとめて格納します.

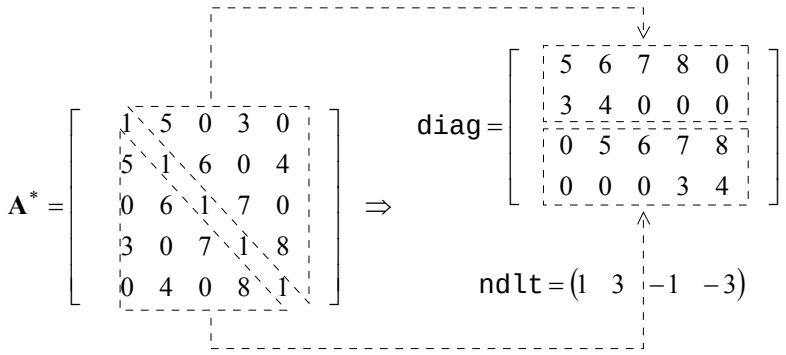


図 1.7 正値対称スパース行列の対角形式の格納法

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～ndt–1 行に上三角行列部分を, diag の ndt～2*nw–1 行に下三角行列部分を格納します. 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りません.

第 2 章　関数の使用方法

c_dm_valu

実行列の LU 分解 (ブロック化された LU 分解法)
`ierr = c_dm_valu(a, k, n, epsz, ip, &is, &icon);`

1. 機能

$n \times n$ の正則な実行列 **A** をブロック化された LU 分解法 (ガウスの消去法)により LU 分解します.

$$\mathbf{PA} = \mathbf{LU}$$

ただし, **P** は部分ピボッティング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す. ($n \geq 1$)

2. 引数

呼出し形式:

```
ierr = c_dm_valu((double*)a, k, n, epsz, ip, &is, &icon);
```

引数の説明:

a	double	入力	行列 **A**.
	a[n][k]	出力	行列 **L** と行列 **U**.
k	int	入力	配列 a の整合寸法 (≥ 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ます. ("使用上の注意" a)参照).
ip	int ip[n]	出力	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" 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ます.
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.
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ます. 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

$$\left|a_{kk}^{k}\right| \le \max\left|a_{ij}\right| \mathrm{epsz}$$

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ります. epsz の標準値は丸め誤差の単位をμとしたとき, epsz = 16μです.

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ません.

b) ip について

トランスポジションベクトルとは, 部分ピボッティングを行う LU 分解

$$\mathbf{PA} = \mathbf{LU}$$

における置換行列 **P** に相当します.

本関数では, 部分ピボッティングに伴い, 配列 a の内容を実際に交換しています. すなわち, 分解の J 段階目 (J = 1, ... , n)において第 I 行($I \ge J$)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ます. そして, その履歴を示すために ip[J-1]に I が格納されます.

c) 本関数の使用方法

本関数に続けて, 関数 c_dm_vlux 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ます. 通常は関数 c_dm_vlax を呼び出せば一度に解が求まります.

4. 使用例

1000×1000 の実行列の LU 分解を行います.

```
#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 NMAX      (1000)
#define LDA       (NMAX+1)

MAIN__()
{
  int    n, is, isw, i, j, icon, ierr;
  int    ip[NMAX];
  double a[NMAX][LDA], b[NMAX];
  double epsz, s, det;

  n    = NMAX;
  epsz = 0.0;
  isw  = 1;

#pragma omp parallel for shared(a,n) private(i,j)
  for(i=0; i<n; i++)
    for(j=0; j<n; j++) a[i][j] = min(i,j)+1;

#pragma omp parallel for shared(b,n) private(i)
  for(i=0; i<n; i++) b[i] = (i+1)*(i+2)/2+(i+1)*(n-i-1);

  ierr = c_dm_valu((double*)a, LDA, n, epsz, ip, &is, &icon);
```

```
  if (icon != 0) {
    printf("ERROR: c_dm_valu failed with icon = %d\n", icon);
    exit(1);
  }

  ierr = c_dm_vlux(b, (double*)a, LDA, n, ip, &icon);

  if (icon != 0) {
    printf("ERROR: c_dm_vlux failed with icon = %d\n", icon);
    exit(1);
  }

  s = 1.0;
#pragma omp parallel for shared(a,n) private(i) reduction(*:s)
  for(i=0; i<n; i++) s *= a[i][i];

  printf("solution vector:\n");
  for(i=0; i<10; i++) printf("    b[%d] = %e\n", i, b[i]);

  det = is*s;
  printf("\ndeterminant of the matrix = %e\n", det);
  return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VALU の項目及び[1], [30], [54], [55], [56], [70]を参照してください.

c_dm_vamlid

スパースな M-行列の連立 1 次方程式 (代数的マルチレベル反復法 [AMLI 法], 対角形式格納法)

`ierr = c_dm_vamlid(a, k, ndiag, n, nofst, b,`
`isw, iguss, info, epsot, epsin,`
`x, w, nw, iw, niw, &icon);`

1. 機能

$n \times n$ の M-行列のスパース行列を係数行列とする連立 1 次方程式を反復法で解きます. ("使用上の注意" a) 参照)

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

$n \times n$ の係数行列 **A** は, 対角形式の格納法で格納します. ベクトル **b** および **x** は n 次元ベクトルです.

反復法は行列 **A** が対称のときは, ORTHOMIN 法が, 非対称のときは, GMRES 法が使用できます. この反復解法(以下外部反復と呼びます)には代数的マルチレベル法(AMLI 法)による前処理が施されていて, 安定な解法です. 代数的マルチレベル法による前処理では縮小された連立 1 次方程式も反復法(以下内部反復と呼びます)で解かれます. (レベルが深くなるほど, 縮小された連立 1 次方程式の次数は小さくなります.)

2. 引数

呼出し形式:

```
ierr = c_dm_vamlid((double*)a, k, ndiag, n, nofst, b, isw, iguss, info,
            epsot, epsin, x, w, nw, iw, niw, &icon);
```

引数の説明:

a	double a[n][k]	入力	係数行列 **A** の非零要素を対角形式で格納します.
k	int	入力	配列 a の整合寸法 (≥ 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します. 上対角ベクトル列は正, 下対角ベクトル列は負の値で表します.
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します.
isw	int	入力	制御情報. 1 初回の呼び出し. 2 2 回目以降の呼び出し. このとき, a, iw, w の 値は, 初回の呼び出しの結果を変更せずに呼び出す必要があります. ("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行うかを指定する制御情報. iguss = 0 解ベクトルの近似値を指定しません. このときゼロベクトルから反復を開始します. iguss ≠ 0 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します.

`info` `int info[14]` 入出力 反復に関する制御情報.

[行列 **A** が対称な場合の指定例]

```
info[0] = –1;   info[1] = NTHRD*100; info[2] = 0;
info[4] = 1;    info[5] = 2000;      info[9] = 1;
info[10]= 1000;
```

[行列 **A** が非対称な場合の指定例]

```
info[0] = –1;   info[1] = NTHRD*100; info[2] = 0;
info[4] = 2;    info[5] = 2000;      info[6] = 5;
info[7] = 20;   info[9] = 2;         info[10]= 1000;
info[11]= 10;   info[12]= 0;
```

NTHRD は, 並列実行するスレッドの数.

`info[0]` 入力 MAXLVL.
代数的マルチレベル法のレベルの最大値.
MAXLVL < 0 のとき, 本関数が評価した最適レベルで行ないます.
MAXLVL = 0 のとき, マルチレベル法は使われません.
MAXLVL > 0 のとき, 指定された値以上の粗いレベルは使われません.
(“使用上の注意” f), g)参照)

`info[1]` 入力 MINUK.
もっとも深いレベルの縮小された連立 1 次方程式の未知数の最小個数.
MINUK は n よりずっと小さく, 並列処理するスレッドの数 NTHRD よりずっと大きな値を指定することを勧めます.
例えば, NTHRD × 100.

`info[2]` 入力 NORM.
行列の正規化の方法を指定します.
NORM < 1 のとき, 行列 **A** は **A** の対角要素の平方根の逆数を対角要素として持つ対角行列で左右から正規化されます. 対称行列を正規化した行列は対称行列です.
対称な正規化を勧めます. しかし, ある場合は, 非対称な正規化の方が速く収束することもあり得ます.
NORM ≥ 1 のとき, 行列 **A** は **A** の対角要素の逆数からなる対角行列で左から正規化します. **A** が対称行列でも正規化されたあとは対称行列ではありません.
(“使用上の注意” d), e)参照)

`info[3]` 出力 使用されたレベル数.

`info[4]` 入力 METHOT.
外部反復で使われる反復法.
METHOT = 1 のとき, 前処理付き ORTHOMIN 法が使われます. 行列 **A** が対称で対称な正規化を指定した場合に指定してください.
METHOT ≠ 1 のとき, 切り捨て再起動する

GMRES 法が使われます. 行列 **A** が非対称であるか, 非対称な正規化が使われた場合に指定してください.

`info[5]`	入力	ITMXOT. 外部反復の最大回数. 例えば, 2000. 外部反復回数がこの値に達したとき, 処理を打ち切ります. このとき返却された解は, 保証されません.
`info[6]`	入力	NRESOT. 外部反復で使われる反復法に GMRES 法を指定したとき, 外部反復の直交手続きで使う残差ベクトルの本数を指定します. 例えば 5. つまり GMRES 法で NRESOT 個のベクトルを使い, 解を更新し, 残りのベクトルによる更新はされません.
`info[7]`	入力	NRSTOT. 外部反復で使われる反復法に GMRES 法を指定したとき, 再起動(restart)する反復回数を指定します. 例えば 20 回. NRSTOT < 1 のとき, 再起動は行いません. (“使用上の注意” e)参照)
`info[8]`	出力	ITEROT. 外部反復の回数.
`info[9]`	入力	METHIN. 内部反復で使われる反復法. METHIN = 1 のとき, 前処理付き ORTHOMIN 法が使われます. 行列 **A** が対称で対称な正規化を指定した場合に指定してください. METHIN ≠ 1 のとき, 切り捨て再起動する GMRES 法が使われます. 行列 **A** が非対称であるか, 非対称な正規化が使われた場合に指定してください.
`info[10]`	入力	ITMXIN. 内部反復の最大回数. 例えば, 1000. 内部反復回数がこの値に達すると, 外部反復に処理は引き継がれます.
`info[11]`	入力	NRESIN. 内部反復で使われる反復法に GMRES 法を指定したとき, 内部反復の直交手続きで計算する残差ベクトルの数を指定します. 例えば 10. つまり GMRES 法で `info[6]`個のベクトルを使い, 解を更新し, 残りのベクトルによる更新はされません. (“使用上の注意” e)参照)
`info[12]`	入力	NRSTIN. 内部反復で使われる反復法に GMRES 法を指定したとき, 再起動(restart)する反復回数を指定します.

			NRSTIN < 1 のとき, 再起動は行いません. ("使用上の注意" e)参照)
			info[13]　出力　内部反復の平均回数.
epsot	double	入力	解の収束判定値. 外部反復の k ステップで求められた解ベクトルが正規化された残差ベクトル $\hat{\mathbf{r}}_k = \hat{\mathbf{A}}\mathbf{x}_k - \hat{\mathbf{b}}_k$ に対して以下の条件を満たしたときベクトルは収束したと見なします. $\|\hat{\mathbf{r}}_k\| \le \mathtt{epsot}\|\hat{\mathbf{b}}\|$, $\|\mathbf{y}\|^2 = \mathbf{y}^{\mathrm{T}}\mathbf{y}$ なるユークリッドノルムであり, $\hat{\mathbf{A}}$ および $\hat{\mathbf{b}}$ は正規化された係数行列および右辺ベクトルです.
epsin	double	入力	内部反復での解の収束判定値. たいていの場合 10^{-3} が最適です.
x	double x[n]	出力	解ベクトルが格納されます.
w	double w[nw]	作業領域	
nw	int	入力	作業用配列 w の大きさ. $\mathtt{nw} \ge \mathtt{NT} \times (3 \times \mathtt{NAMAX} + 5) + 3 \times (\mathtt{NLVL} + 1) \times \mathtt{NBAND} \times \mathtt{MAXT} + \max(\mathtt{NAMAX} \times \mathtt{NT}, 7 \times \mathtt{NT} + \mathtt{LR0})$ MAXT は, 本関数を並列実行する最大スレッド数. $\mathtt{NT} = \mathtt{n} + \mathtt{MAXT}$. NBAND は, 下バンド巾と上バンド巾の最大値. NLVL は代数的マルチレベル法のレベルの数. MAXLVL < 0 のときは 10. $\mathtt{NAMAX} \ge$ ndiag. ("使用上の注意" c)参照) ORTHOMIN 法を使用したとき: $\mathtt{LR0} = 4 \times \mathtt{NT}$. GMRES 法を使ったとき: $\mathtt{NRES} = \max(\mathrm{NRESOT}, \mathrm{NRESIN})$. $\mathtt{LR0} = (2 \times \mathtt{NRES} + 1) \times \mathtt{NT}$.
iw	int iw[niw]	作業領域	
niw	int	入力	作業用配列 iw の大きさ. $\mathtt{niw} \ge \mathtt{MAXT} \times ((6 \times \mathtt{MAXT} + 12 \times \mathtt{NAMAX}) \times (\mathtt{NLVL} + 1) + 8 \times \mathtt{NBAND} + 3000) + 4 \times (\mathtt{n} + \mathtt{MAXT})$ MAXT は, 本関数を並列実行する最大スレッド数. NBAND は, 下バンド巾と上バンド巾の最大値. NLVL は代数的マルチレベル法のレベルの数. MAXLVL < 0 のときは 10. $\mathtt{NAMAX} \ge$ ndiag. ("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.
10700	ベクトル $\mathbf{v}^{pos}$ が見つからなかった.	$\mathbf{v}^{pos} = (1, 1, \ldots, 1)$として処理を続ける.
10800	GMRES 法で回復可能な break down を起こした.	処理を続ける.
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.

コード	意味	処理内容
20003	GMRES 法で回復不可能な break down を起こした.	処理を打ち切る.
20005	ORTHOMIN 法で $\mathbf{p}^T \mathbf{A}\,\mathbf{p} = 0$ となり回復不可能な break down を起こした.	
20006	ORTHOMIN 法で $\mathbf{p}^T \mathbf{r} = 0$ となり回復不可能な break down を起こした.	
30000	次のいずれかであった. • n < 1 • n > k • ndiag < 1 • isw ≠ 1, 2	
30104	|nofst[i]| > n–1	
30105	主対角要素がない.	処理を打ち切る.
30200	係数行列が M-行列でない.	
30210	行列の圧縮が非正値であるためできない.	
30212	対角要素にゼロ要素がある.	
30310	niw が小さ過ぎる.	
30320	nw が小さ過ぎる.	

3. 使用上の注意

a) M-行列について

楕円型の境界値問題を 2 次または 1 次の有限差分離散化を行って生成した係数行列は M-行列です. これらは, 2 階偏微分方程式を離散化するルーチン(c_dm_vpde2d, c_dm_vpde3d)で生成することができます.

M-行列とは, 以下の条件を意味します.

- すべての対角要素は正 ($a_{i,i} > 0$, $i = 1, \ldots, n$), かつ他の要素は非正値 ($a_{i,j} \le 0$, $i, j = 1, \ldots, n$, $i \ne j$).
- 正値のベクトル $\mathbf{v}^{pos}$ があって $\mathbf{A}\mathbf{v}^{pos}$ が正値.

最初の条件を満たさないとき, また本ルーチンが $\mathbf{v}^{pos}$ を見つけることができなかった場合(icon = 10700), 本ルーチンは $\mathbf{v}^{pos} = (1, \ldots, 1)$として breakdown のリスクをもって処理を続けます. この場合, 外部反復または内部反復で収束が遅かったり, breakdown を起こしたりする可能性(icon = 30212, 30210)があります.

粗いレベルを定義するために, 係数行列を組み立てるために使われた矩形のグリッドを再構成します. 再構成に失敗したときは, breakdown するか, 粗いレベルの行列の対角ベクトルが過度に増えて収束が遅くなったり, 外部または内部反復で breakdown を起こしたりする可能性があります.

b) isw について

同じ係数行列を持つ右辺の異なる多重組みの連立 1 次方程式を解く場合は, 最初の呼び出しで isw = 1 で呼び出して解き, 2 回目以降の呼び出しでは isw = 2 として呼び出すことで解けます. 最初の呼び出しで組み立てられた粗いレベルに対応する行列が 2 回目以降の呼び出しで再利用されます.

c) NAMAX について

たいていの場合は, NAMAX = ndiag で十分です. 与えられた係数行列の対角ベクトルの数より粗いレベルの行列の対角ベクトルの数が大きい場合が起こり得ます. このとき NAMAX は増やさなければなりません.

d) ORTHOMIN 法について

行列が対称のとき, ORTHOMIN 法を使うことを勧めます. 本関数は計算のできる未知数を反復が始まる前に行列から除きます. 与えられた行列が対称でなくても実際の反復計算では行列が対称であることが起こ

り得ます. そのため, 行列が対称となる可能性がある場合は, 最初に対称な正規化を行う ORTHOMIN 法で解くことを試みることを勧めます.

e) GMRES 法について

行列が対称でない場合は, 非対称な正規化を行って GMRES 法を使うことを勧めます. たいていの場合, 外部反復で NRESOT = 5 として切り捨て 20 ステップ後に再起動すれば十分です. 内部反復では, NRESIN として切り捨てる数をもっと大きくし, もっと大きなステップの後に再起動するか再起動を禁じることが必要です. NRESIN を大きくすると作業域を大きくする必要があります. このため作業域の大きさの計算式の中の NRES を大きくしなければなりません. しかし NRES は NRESOT より小さくできます.

一般に, NRESOT および NRESIN を大きくすると GMRES 法の精度はよくなりますが, 計算およびメモリ使用量も増えます.

f) レベル数について

本関数は最適なレベルを見つけようとします. 問題によっては, レベル数を指定することで計算時間が減ることもあります. しかし, たいていの場合改善はさほど大きくありません.

g) 前処理について

各レベルの行列を $\mathbf{A}_n(n= 1, \ldots , \mathrm{MAXLVL}-1)$としたとき,

$$\mathbf{A}_n = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}$$

$\mathbf{S} = \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12}$ を Schur complement と呼びます. 擬似格子による変数の並びを利用して, 消去する適当な変数の集合を選ぶことでこのようなブロックを形成します. $\mathbf{A}_{11}$ の逆行列を対角行列で近似します. この近似的逆行列を使って Schur complement を計算し, レベル n+1 の行列 $\mathbf{A}_{n+1}$ とします.

Schur complement を使い $\mathbf{A}_n$ は次のように分解できます.

$$\mathbf{A}_n = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{0} \\ \mathbf{A}_{21} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{A}_{11}^{-1}\mathbf{A}_{12} \\ \mathbf{0} & \mathbf{S} \end{bmatrix}$$

$\mathbf{A}_{11}^{-1}$ の近似を利用して, 各レベルをこのように近似的に分解し前処理として利用します.

4. 使用例

偏微分方程式を領域 $[0, 1]^2$ で離散化して解きます.

$$-\left(\frac{\partial^2 u}{\partial^2 x_1} + \frac{\partial^2 u}{\partial^2 x_2}\right) + cu = 1$$

Dirichlet 境界条件 $u = 0$ を設定します.

```
#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 MAXT       4
#define N1      1281
#define N2      1537
#define NLVL      10
#define L1      (N1)
```

```
#define L2      (N2)
#define KA   (N1*N2)
#define NA        5
#define NW     ((3*NA+5)*(KA+MAXT)+3*(NLVL+1)*N1*MAXT+11*(KA+MAXT))
#define NIW  (((6*MAXT+12*NA)*(NLVL+1)+8*N1+2000)*MAXT+4*(KA+MAXT))

int MAIN__()
{
  double a[NA][KA], b[KA], u[KA], sol[3*N1*N2], rhs[N1*N2], rhsc[N1*N2];
  double x1[L1], x2[L2], a1[L2][L1], a2[L2][L1], b1[L2][L1], b2[L2][L1];
  double c[L2][L1], f[L2][L1], w[NW], epsin, epsot, tmp;
  int    nofst[NA], info[100], iw[NIW];
  int    z1, z2, ndiag, n, isw, iguss, nband, i, z, icon;

  /* CREATE NODE COORDINATES */
  for (z1=0; z1<N1; z1++) {
    x1[z1] = (double)(z1)/(double)(N1-1);
  }

  for (z2=0; z2<N2; z2++) {
    x2[z2] = (double)(z2)/(double)(N2-1);
  }

  /* COEFFICIENTS IN THE PARTIAL DIFFERENTIAL EQUATION : */
  for (z2=0; z2<N2; z2++) {
    for (z1=0; z1<N1; z1++) {
      a1[z2][z1] = 1.0;
      a2[z2][z1] = 1.0;
      b1[z2][z1] = 0.0;
      b2[z2][z1] = 0.0;
      c[z2][z1]  = 1.0;
      f[z2][z1]  = 1.0;
    }

    /* DIRICHLET BOUNDARY CONDITIONS: */
    c[z2][0]    = 1.0;
    f[z2][0]    = 0.0;
    c[z2][N1-1] = 1.0;
    f[z2][N1-1] = 0.0;

    if (z2 == 0) {
      for (z1=0; z1<N1; z1++) {
        c[0][z1] = 1.0;
        f[0][z1] = 0.0;
      }
    }

    if (z2 == N2-1) {
      for (z1=0; z1<N1; z1++) {
        c[N2-1][z1] = 1.0;
        f[N2-1][z1] = 0.0;
      }
    }
  }

  n = N1*N2;
  c_dm_vpde2d((double*)a1, L1, N1, N2, (double*)a2, x1, x2, (double*)b1,
              (double*)b2, (double*)c, (double*)f, (double*)a, KA, NA, n,
              &ndiag, nofst, b, &icon);
  printf("icon of c_dm_vpde2d = %d\n", icon);

  for (z=0; z<n; z++) {
    rhs[z] = b[z];
  }

  nband = 0;
  for (i=0; i<ndiag; i++) {
    nband = max(nband,fabs(nofst[i]));
  }

  /* CALL DAMLI: */
  isw   = 1;
  iguss = 0;

  info[0]  = -1;
  info[1]  = MAXT*100;
  info[2]  = 0;
```

```
  info[4]  = 1;
  info[5]  = 2000;
  info[9]  = 1;
  info[10] = 1000;

  epsot = 1e-6;
  epsin = 1e-3;

  c_dm_vamlid((double*)a, KA, ndiag, n, nofst, b, isw, iguss, info, epsot, epsin, u,
              w, NW, iw, NIW, &icon);
  printf("icon of c_dm_vamlid = %d\n", icon);

  for (i=0; i<nband; i++) {
    sol[i]          = 0.0;
    sol[nband+n+i-1] = 0.0;
  }

  for (z=0; z<n; z++) {
    sol[nband+z] = u[z];
  }

  c_dm_vmvsd((double*)a, KA, ndiag, n, nofst, nband, sol, rhsc, &icon);

  tmp = 0.0;
  for (z=0; z<n; z++) {
    tmp = max(tmp,fabs((rhs[z]-rhsc[z])/(rhs[z]+1.0)));
  }

  printf("error = %e\n", tmp);
  return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VAMLID の項目を参照してください.

c_dm_vbcscc

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, 圧縮列格納法)

`ierr = c_dm_vbcscc(a, nz, nrow, nfcnz, n, b,`
`itmax, eps, iguss, l, x, &iter,`
`w, iw, &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)で解きます.

$$\mathbf{Ax} = \mathbf{b} \tag{1}$$

$n \times n$ の係数行列 **A** は, 圧縮列格納法で格納します.
ベクトル **b** および **x** は n 次元ベクトルです.
反復解法の収束および利用指針に関しては, “SSL II 拡張機能使用手引書 II 第 I 部 概説 第 4 章 連立 1 次方程式の反復解法と収束性”を参照してください.

2. 引数

呼出し形式:

```
ierr = c_dm_vbcscc(a, nz, nrow, nfcnz, n, b, itmax, eps, iguss, l, x, &iter,
            w, (int*)iw, &icon);
```

引数の説明:

a	double a[nz]	入力	係数行列の非零要素を格納します. a[*i*], *i*=0,…,nz-1 にスパース行列の非零要素を格納します. 圧縮列格納法については, c_dm_vmvscc の図 c_dm_vmvscc-1 参照.
nz	int	入力	係数行列 **A** の非零要素の総数.
nrow	int nrow[nz]	入力	圧縮列格納法で使用される行指標で, **A** に格納される要素が何番目の行ベクトルに属するかを示します.
nfcnz	int nfcnz[n+1]	入力	行列 **A** の各列の非零要素を列方向に圧縮して順次配列 **A** に格納するとき, 対応する列の最初の非零要素が格納される位置を表します. nfcnz[n]=nz+1
n	int	入力	行列 **A** の次数 n.
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します.
itmax	int	入力	BICGSTAB(l)法の反復回数の上限値. (> 0). 2000 程度で十分です.
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} が設定されます. (“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ません. 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します.

l	int	入力	BICGSTAB(*l*)法で安定化を行うときの安定化のステップ数 *l* ($1 \le l \le 8$). ほとんどの場合 1 か 2 で十分です. ("使用上の注意" b)参照)
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ます. (引数"iguss"の項を参照)
		出力	解ベクトル **x** が格納されます.
iter	int	出力	BICGSTAB(*l*)法での実際の反復回数.
w	double w[nz]	作業領域	
iw	int iw[nz][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.
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 • nz < 0 • nfcnz[n] ≠ nz+1 • itmax ≤ 0 • l < 1 • l > 8	処理を打ち切る.

3. 使用上の注意

a) 収束判定について

本ルーチンは, 残差のユークリッドノルムが最初の残差のユークリッドノルムと eps の積以下になったとき解が収束したと見なします. 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 の積に等しくなります.

b) l の値について

l = 1 のとき, BICGSTAB 法と同じアルゴリズムになります. l の値を大きくすると, 反復 1 回あたりのコストは増加しますが, 反復回数が減り, よい収束が得られる場合があります.

4. 使用例

連立 1 次方程式 **Ax** = **f** を解き, 得られた結果を正しい値と比較し確認します. 行列 **A** は境界条件として立方体の境界で 0 となる楕円型の偏微分方程式に有限差分法を適用して生成されます.

$$-\Delta u + a\nabla u + u = f$$

ここで $a = (a_1, a_2, a_3)$, a_1, a_2 および a_3 はある定数です. 行列 **A** はルーチン init_mat_diag によって生成されます. これを圧縮列格納法に変換します.

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

```
#define NORD   (60)
#define NX     (NORD)
#define NY     (NORD)
#define NZ     (NORD)
#define N      (NX*NY*NZ)
#define K      (N+1)
#define NDIAG  (7)
#define L      (4)

MAIN__()
{
  int    ierr, icon, iguss, iter, itmax;
  int    nord, n, l, i, j, k;
  int    nx, ny, nz, nnz;
  int    length, nbase, ndiag;
  int    numnz, ntopcfg, ncol;
  int    nofst[NDIAG];
  int    nrow[K*NDIAG];
  int    nfcnz[N+1];
  int    iw[K*NDIAG][2];

  double eps;
  double va1, va2, va3, vc;
  double err1, err2, err3, err4;
  double xl, yl, zl;
  double diag[NDIAG][K];
  double diag2[NDIAG][K];
  double a[K*NDIAG];
  double b[N];
  double w[K*NDIAG];
  double x[N];
  double solex[N];
  double y[N];

  void init_mat_diag(double va1, double va2, double va3, double vc,
              double d_l[], int offset[], int nx, int ny, int nz,
              double xl, double yl,double zl, int ndiag, int len, int ndivp);

  double errnrm(double *x1, double *x2, int len);

  nord=NORD, nx=NX, ny=NY, nz=NZ, n=N, k=K, ndiag=NDIAG, l=L;

  printf("    BICGSTAB(L) METHOD\n");
  printf("    COMPRESSED COLUMN STORAGE\n");
  printf("\n");

  for (i=1; i<=n; i++){
    solex[i-1]=1.0;
  }
  printf("    EXPECTED SOLUTIONS\n");
  printf("    X(1) = %f  X(N) = %f\n", solex[0], solex[n-1]);
  printf("\n");

  va1 = 3.0;
  va2 = 1.0/3.0;
  va3 = 5.0;
  vc = 1.0;
  xl = 1.0;
  yl = 1.0;
  zl = 1.0;
  init_mat_diag(va1, va2, va3, vc, (double*)diag, (int*)nofst,
                      nx, ny, nz, xl, yl, zl, ndiag, n, k);

  for (i=1; i<=ndiag; i++){
    if (nofst[i-1] < 0){
      nbase=-nofst[i-1];
      length=n-nbase;
      for (j=1; j<=length; j++){
        diag2[i-1][j-1]=diag[i-1][nbase+j-1];
      }
    }
    else{
      nbase=nofst[i-1];
      length=n-nbase;
      for (j=nbase+1; j<=n; j++){
        diag2[i-1][j-1]=diag[i-1][j-nbase-1];
      }
```

```
    }
  }

  numnz=1;
  for (j=1; j<=n; j++){
    ntopcfg = 1;
    for (i=ndiag; i>=1; i--){
      if (diag2[i-1][j-1]!=0.0){
        ncol=j-nofst[i-1];
        a[numnz-1]=diag2[i-1][j-1];
        nrow[numnz-1]=ncol;
        if (ntopcfg==1){
          nfcnz[j-1]=numnz;
          ntopcfg=0;
        }
        numnz=numnz+1;
      }
    }
  }
  nfcnz[n]=numnz;
  nnz=numnz-1;

  for (i=1; i<=n; i++){
    x[i-1]=0.0;
  }

  ierr = c_dm_vmvscc(a, nnz, nrow, nfcnz, n, solex, b, w, (int*)iw, &icon);
  err1 = errnrm(solex,x,n);

  ierr = c_dm_vmvscc(a, nnz, nrow, nfcnz, n, x, y, w, (int*)iw, &icon);
  err2 = errnrm(y,b,n);

  iguss = 0;
  itmax = 2000;
  eps   = 1.0e-8;

  ierr = c_dm_vbcscc(a, nnz, nrow, nfcnz, n, b, itmax, eps, iguss, l,
                          x, &iter, w, (int*)iw, &icon);
  err3 = errnrm(solex,x,n);

  ierr = c_dm_vmvscc(a, nnz, nrow, nfcnz, n, x, y, w, (int*)iw, &icon);
  err4 = errnrm(y,b,n);

  printf("    COMPUTED VALUES\n");
  printf("    X(1) = %f  X(N) = %f\n", x[0], x[n-1]);
  printf("\n");
  printf("    c_dm_vbcscc ICON = %d\n", icon);
  printf("\n");
  printf("    N = %d   :: NX = %d  NY = %d  NZ = %d\n",n,nx,ny,nz);
  printf("    ITER MAX = %d\n",itmax);
  printf("    ITER     = %d\n",iter);
  printf("\n");
  printf("    EPS      = %e\n",eps);
  printf("\n");
  printf("    INITIAL ERROR = %f\n",err1);
  printf("    INITIAL RESIDUAL ERROR  = %f\n",err2);
  printf("    CRITERIA RESIDUAL ERROR = %e\n",err2 * eps);
  printf("\n");
  printf("    ERROR  = %e\n",err3);
  printf("    RESIDUAL ERROR = %e\n",err4);
  printf("\n");
  printf("\n");

  if (err4<=(err2*eps*1.1) && icon==0){
    printf("********** OK **********\n");
  }
  else{
    printf("********** NG **********\n");
  }
}

void init_mat_diag(double va1, double va2, double va3, double vc,
         double d_l[], int offset[], int nx, int ny, int nz,
         double xl, double yl, double zl, int ndiag, int len, int ndivp)
{
  int i, l, j;
  int length, numnz, js;
```

```
int i0, j0, k0;
int ndiag_loc;
int nxy;

double hx, hy, hz;
double x1, x2;
double base;
double ret, remark;

if (ndiag<1){
  printf("FUNCTION INIT_MAT_DIAG:\n");
  printf("NDIAG SHOULD BE GREATER THAN OR EQUAL TO 1\n");
  return;
}
ndiag_loc = ndiag;
if (ndiag>7){
  ndiag_loc=7;
}

hx = xl / (nx + 1);
hy = yl / (ny + 1);
hz = zl / (nz + 1);

for (i=1; i<=ndivp; i++){
  for (j=1; j<=ndiag; j++){
    d_l[i-1+(j-1)*ndivp]= 0.;
  }
}

nxy = nx * ny;
l = 1;
if (ndiag_loc >= 7) {
  offset[l-1] = -nxy;
  ++l;
}
if (ndiag_loc >= 5) {
  offset[l-1] = -nx;
  ++l;
}
if (ndiag_loc >= 3) {
  offset[l-1] = -1;
  ++l;
}
offset[l-1] = 0;
++l;
if (ndiag_loc >= 2) {
  offset[l-1] = 1;
  ++l;
}
if (ndiag_loc >= 4) {
  offset[l-1] = nx;
  ++l;
}
if (ndiag_loc >= 6) {
  offset[l-1] = nxy;
}

for (j = 1; j <= len; ++j) {
  js=j;
  k0 = (js - 1) / nxy + 1;
  if (k0 > nz) {
    printf("ERROR; K0.GH.NZ\n");
    return;
  }
  j0 = (js - 1 - nxy * (k0 - 1)) / nx + 1;
  i0 = js - nxy * (k0 - 1) - nx * (j0 - 1);

  l = 1;
  if (ndiag_loc >= 7) {
    if (k0 > 1) {
      d_l[j-1+(l-1)*ndivp] = -(1.0/hz+va3*0.5)/hz;
    }
    ++l;
  }

  if (ndiag_loc >= 5) {
    if (j0 > 1) {
```

```
        d_l[j-1+(l-1)*ndivp] = -(1.0/hy+va2*0.5)/hy;
      }
      ++l;
    }

    if (ndiag_loc >= 3) {
      if (i0 > 1) {
        d_l[j-1+(l-1)*ndivp] = -(1.0/hx+va1*0.5)/hx;
      }
      ++l;
    }

    d_l[j-1+(l-1)*ndivp] = 2.0/(hx*hx)+vc;
    if (ndiag_loc >= 5) {
      d_l[j-1+(l-1)*ndivp] += 2.0/(hy*hy);
      if (ndiag_loc >= 7) {
        d_l[j-1+(l-1)*ndivp] += 2.0/(hz*hz);
      }
    }
    ++l;
    if (ndiag_loc >= 2) {
      if (i0 < nx) {
        d_l[j-1+(l-1)*ndivp] = -(1.0/hx-va1*0.5)/hx;
      }
      ++l;
    }

    if (ndiag_loc >= 4) {
      if (j0 < ny) {
        d_l[j-1+(l-1)*ndivp] = -(1.0/hy-va2*0.5)/hy;
      }
      ++l;
    }

    if (ndiag_loc >= 6) {
      if (k0 < nz) {
        d_l[j-1+(l-1)*ndivp] = -(1.0/hz-va3*0.5)/hz;
      }
    }
  }
  return;
}

double errnrm(double *x1, double *x2, int len)
{
  double ret_val;

  int i;
  double s, ss;

  s = 0.;
  for (i = 1; i <= len; ++i) {
    ss = x1[i-1] - x2[i-1];
    s += ss * ss;
  }
  ret_val = sqrt(s);
  return ret_val;
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VBCSCC の項目及び[32], [72] , [77]を参照してください.

c_dm_vbcsd

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_dm_vbcsd(a, k, ndiag, n, nofst, b, itmax, eps, iguss, l, x, &iter, &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)で解きます.

$$\mathbf{Ax} = \mathbf{b} \tag{1}$$

係数行列は, 対角形式の格納法で格納します. ベクトル **b** 及び **x** は n 次元ベクトル. 反復解法の収束および利用指針に関しては, “SSL II 拡張機能使用手引書 II 第 I 部 概説 第 4 章 連立 1 次方程式の反復解法と収束性”を参照してください.

2. 引数

呼出し形式:

```
ierr = c_dm_vbcsd((double*)a, k, ndiag, n, nofst, b, itmax, eps, iguss, l, x,
            &iter, &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します.
k	int	入力	配列 a の整合寸法 (≥ 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します. 上対角ベクトル列は正, 下対角ベクトル列は負の値で表します.
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します.
itmax	int	入力	BICGSTAB(l)法の反復回数の上限値. (> 0). 2000 程度で十分です.
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} が設定されます. (“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ません. 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します.
l	int	入力	BICGSTAB(l)法で安定化を行うときの安定化のステップ数 l ($1 \le l \le 8$). ほとんどの場合 1 か 2 で十分です. (“使用上の注意” b)参照)

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ます. (引数“iguss”の項を参照)
		出力	解ベクトル **x** が格納されます.
iter	int	出力	BICGSTAB(*l*)法での実際の反復回数.
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.
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 ≤ 0	処理を打ち切る.
32001	|nofst[i]| > n–1 (0 ≤ 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します. 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 の積に等しくなります.

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ります.

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ます. l=1 のとき, BICGSTAB 法と同じアルゴリズムになります. l の値が大きいと, 反復一回あたりのコストは増加しますが, 反復回数が減り, よい収束が得られる場合があります.

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ります. 対角ベクトル列を配列 a に格納する順序に制限は特にありません.

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します.

```
#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)
#define NDIAG   (UBANDW + LBANDW + 1)
#define L       (2)

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[NDIAG];
  double a[NDIAG][NMAX], b[NMAX], x[NMAX];

  nub   = UBANDW;
  nlb   = LBANDW;
  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;
  ierr = c_dm_vbcsd ((double*)a, k, NDIAG, n, nofst, b, itmax, eps,
                 iguss, L, x, &iter, &icon);

  if (icon != 0) {
    printf("ERROR: c_dm_vbcsd 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 スレッド並列機能使用手引書”の DM_VBCSD の項目及び[32], [72], [77]を参照してください.

c_dm_vbcse

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_dm_vbcse(a, k, iwidt, n, icol, b,`
`itmax, eps, iguss, l, x, &iter,`
`&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)で解きます.

$$\mathbf{Ax} = \mathbf{b} \tag{1}$$

係数行列は, ELLPACK 形式の格納法で格納します. ベクトル **b** 及び **x** は n 次元ベクトル. 反復解法の収束および利用指針に関しては, "SSL II 拡張機能使用手引書 II 第 I 部 概説 第 4 章 連立 1 次方程式の反復解法と収束性"を参照してください.

2. 引数

呼出し形式:

```
ierr = c_dm_vbcse((double*)a, k, iwidt, n, (int*)icol, b, itmax, eps, iguss,
            l, x, &iter, &icon);
```

引数の説明:

a	double a[iwidt][k]	入力	係数行列の非零要素を ELLPACK 形式格納法で格納します.
k	int	入力	a 及び icol の整合寸法 (≥ 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. 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[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かを示します.
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します.
itmax	int	入力	BICGSTAB(*l*)法の反復回数の上限値. (> 0). 2000 程度で十分です.
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} が設定されます. ("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開始するかどうか示す制御情報. 0 のとき　解ベクトルの近似値を指定しません. 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します.
l	int	入力	BICGSTAB(*l*)法で安定化を行うときの安定化のステップ数 l (1 ≤ l ≤ 8). ほとんどの場合, 1 か 2 で十分です. ("使用上の注意" b)参照)
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

			きます. (引数 iguss の項を参照)
		出力	解ベクトルが格納されます.
iter	int	出力	BICGSTAB(*l*)法での実際の反復回数.
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.
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 • iwidt < 1 • iwidt > k • itmax ≤ 0	処理を打ち切る.
30001	バンド巾が零であった.	

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します. 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 の積に等しくなります.

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ります.

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ます. l=1 のとき, BICGSTAB 法と同じアルゴリズムになります. l の値が大きいと, 反復一回あたりのコストは増加しますが, 反復回数が減り, よい収束が得られる場合があります.

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します.

```
#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)
#define IWIDT   (UBANDW + 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, n, k, itmax, iguss, iter, i, j, ix;
  int    icol[IWIDT][NMAX];
  double a[IWIDT][NMAX], b[NMAX], x[NMAX];

  nub   = UBANDW;
  nlb   = LBANDW;
  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;
  ierr = c_dm_vbcse ((double*)a, k, IWIDT, n, (int*)icol, b, itmax,
                 eps, iguss, L, x, &iter, &icon);

  if (icon != 0) {
    printf("ERROR: c_dm_vbcse 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 スレッド並列機能使用手引書”の DM_VBCSE の項目及び[32], [72], [77]を参照してください.

c_dm_vblu

実バンド行列の LU 分解 (ガウスの消去法)
`ierr = c_dm_vblu(a, k, n, nh1, nh2, epsz,` `&is, ip, &icon);`

1. 機能

$n \times n$, 下バンド幅 h_1, 上バンド幅 h_2 のバンド行列 **A** をガウスの消去法により LU 分解します.

$$\mathbf{PA} = \mathbf{LU}$$

ただし, **P** は行ベクトルの置換行列, **L** は単位下バンド行列, **U** は上バンド行列です. $n > h_1 \geq 0, n > h_2 \geq 0$.

2. 引数

呼出し形式:

```
ierr = c_dm_vblu((double*)a, k, n, nh1, nh2, epsz, &is, ip, &icon);
```

引数の説明:

a	double a[n][k]	入力	バンド係数行列 **A** を格納します. 詳細については, 図 c_dm_vblu-1.を参照してください.
		出力	LU 分解された行列 ***L*** および ***U*** が格納されます. 詳細については, 図 c_dm_vblu-2.を参照してください.
k	int	入力	配列 a の整合寸法 ($\geq 2 \times$ nh1 + nh2 + 1).
n	int	入力	行列 **A** の次数 *n*.
nh1	int	入力	下バンド巾の大きさ h_1.
nh2	int	入力	上バンド巾の大きさ h_2.
epsz	double	入力	ピボットの零判定値 (≥ 0.0). 0.0 のときは, 標準値が設定されます. (“使用上の注意” a)参照)
is	int	出力	行ベクトルの入換えの回数を示します. (“使用上の注意” d)参照) 1　偶数回の入換え. –1　奇数回の入換え.
ip	int ip[n]	出力	行の入れ換えの履歴情報を格納するトランスポジションベクトルが格納されます. (“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.

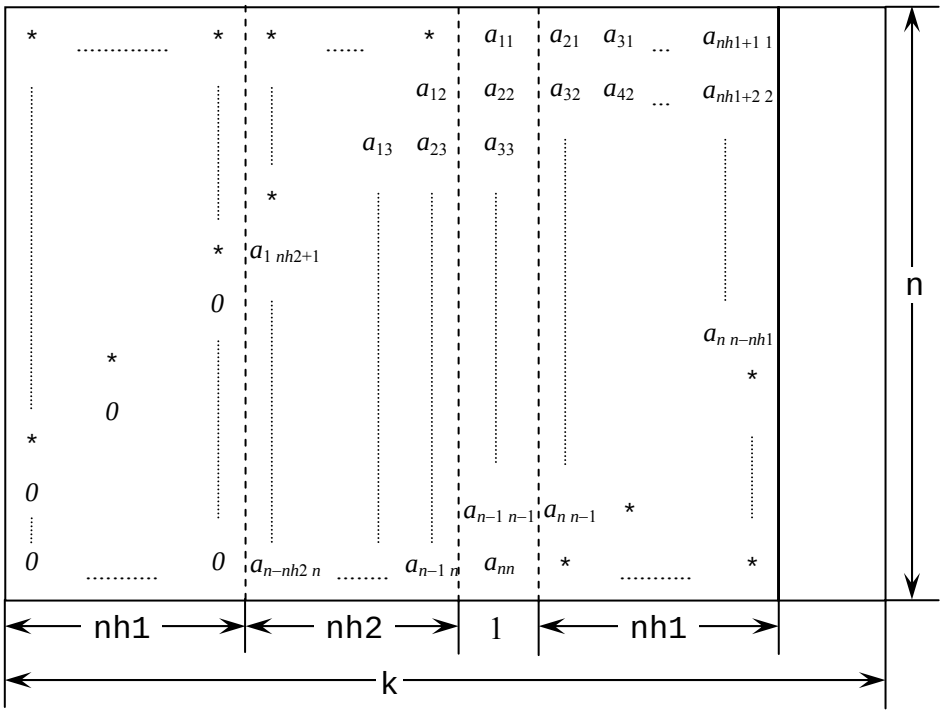


図 c_dm_vblu-1. 配列 a に行列 **A** を格納する方法

バンド行列 **A** の対角要素 a_{ii}, $i = 1, \ldots, n$ が a[i–1][h_1+h_2]に格納されるように, 行列 **A** の列ベクトルを配列 a に連続に格納します.

上バンド行列部分, $a_{j-i,\,j}$, $i = 1, \ldots, h_2$, $j = 1, \ldots, n$, $j - i \geq 1$ を, a[i][j], i = 0, ... , n–1, j = h_1, ... , h_1+h_2–1 に格納します. 下バンド行列部分, $a_{j+i,\,j}$, $i = 1, \ldots, h_1$, $j = 1, \ldots, n$, $j + i \leq n$ を, a[i][j], i = 0, ... , n–1, j = h_1+h_2+1, ... , 2×h_1+h_2 に格納します.

配列 a[i][j], i = 0, ... , n – 1, j = 0, ... , h_1–1, で, バンドの外側の行列 **A** の要素はゼロを設定してください.

*は不定値を示します.

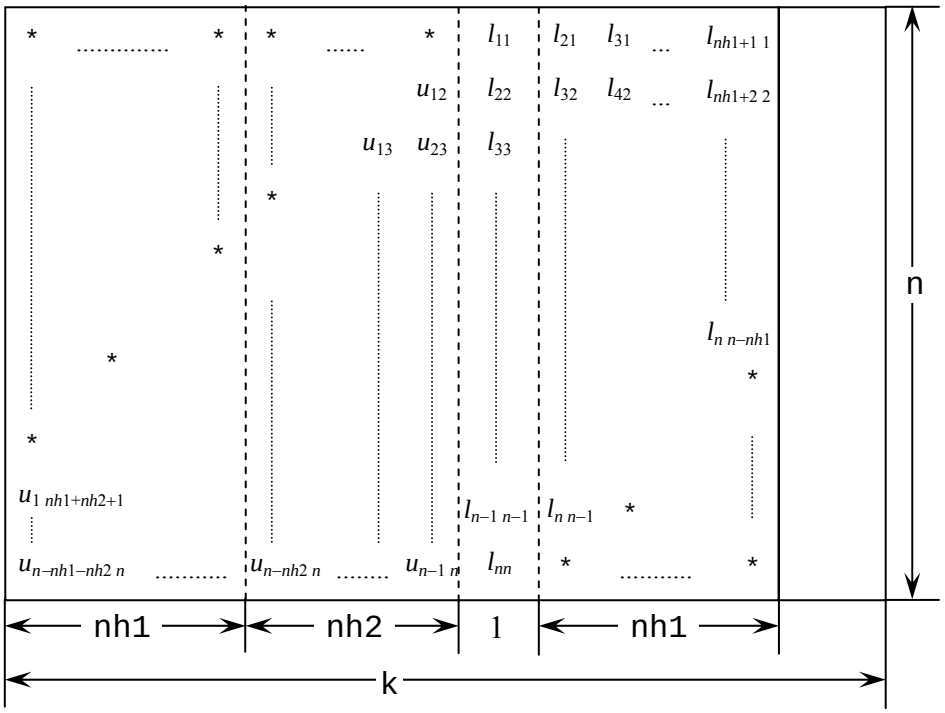


図 c_dm_vblu-2. LU 分解された行列 **L** および **U** の配列 a に格納される方法

LU 分解された単位上バンド行列の対角要素を除いた部分, $u_{j-i+1,\,j}$, $i = 1, \ldots, h_1+h_2$, $j = 1, \ldots, n$, $j - i + 1 \geq 1$ が a[i][j], i = 0, ... , n–1, j = 0 , ... , h_1+h_2,に格納されます.

下バンド行列, $l_{j+i,\,j}$, $i = 0, \ldots, h_2$, $j = 1, \ldots, n$, $j + i \leq n$ が a[i][j], i = 0, ... , n–1, j = h_1+h_2, ... , $2\times h_1+h_2$ に格納されます.

*は不定値を示します.

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 < 1 • nh1 ≥ n • nh1 < 0 • nh2 ≥ n • nh2 < 0 • k < 2 × nh1 + nh2 + 1 • epsz < 0	

3. 使用上の注意

a) epsz について

epsz が設定されると, LU 分解の過程でピボットが epsz 以下のときは相対的に零と見なし, icon = 20000 で処理を打ち切ります. epsz の標準値は丸め誤差の単位を u としたとき $16\times u$ です. なおピボットが小さくても処理を続行させたいときには, epsz に極小の値を設定すればよいのですが, その結果は保証されません.

b) ip について

本関数では, 部分ピボッティングに伴い, 行ベクトルの交換を行っています. すなわち, 分解の J 段階目($J = 1, \ldots, n$)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, 第 I 行と第 J 行の内容が交換されます. そして, その履歴を示すために ip[$J-1$]に I が格納されます.

c) 本関数の使用方法

本関数に続けて, 関数 c_dm_vblux 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ます. 通常は関数 c_dm_vlbx を呼び出せば一度に解が求まります.

d) 行列式の値について

行列式の値は, is および a[i][h_1+h_2], i = 0, ... , $n-1$ を掛け合わせることで求めることができます.

4. 使用例

$n = 10000, nh_1 = 2000, nh_2 = 3000$ の実バンド行列を入力して, バンド行列の連立 1 次方程式を解きます.

```
#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 NH1   2000
#define NH2   3000
#define N     10000
#define KA    (2*NH1+NH2+1)
#define NWORK 4500

int MAIN__()
{
  double a[N][KA], b[N], dwork[NWORK];
  double tt1, tt2, tmp, epsz;
  int    ip[N], i, j, is, ix, icon, nptr, nbase, nn;

  ix = 123;
  nn = NH1+NH2+1;
  for (i=0; i<N; i++) {
    c_dvrau4(&ix,&a[i][NH1],nn,dwork,NWORK,&icon);
  }

  printf("nh1 = %d, nh2 = %d, n = %d\n", NH1, NH2, N);

  /* zero clear */
  for (j=0; j<N; j++) {
    for (i=0; i<NH1; i++) {
      a[j][i] = 0.0;
    }
  }

  /* left upper triangular part */
  for (j=0; j<NH2; j++) {
    for (i=0; i<NH2-j; i++) {
      a[j][i+NH1] = 0.0;
    }
```

```
  }

  /* right rower triangular part */
  nbase = 2*NH1+NH2+1;
  for (j=0; j<NH1; j++) {
    for (i=0; i<j; i++) {
      a[N-NH1+j][nbase-i-1] = 0.0;
    }
  }

  /* set right hand constant vector */
  for (i=0; i<N; i++) {
    b[i] = 0.0;
  }

  for (i=0; i<N; i++) {
    nptr = i;
    for (j=max(nptr-NH2,0); j<min(N,nptr+NH1+1); j++) {
      b[j] += a[i][j-i+NH1+NH2];
    }
  }

  epsz = 0.0;
  c_dm_vblu((double*)a, KA, N, NH1, NH2, epsz, &is, ip, &icon);
  c_dm_vblux(b, (double*)a, KA, N, NH1, NH2, ip, &icon);

  tmp = 0.0;
  for (i=0; i<N; i++) {
    tmp = max(tmp,fabs(b[i]-1));
  }

  printf("maximum error = %e\n", tmp);
  return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VBLU の項目を参照してください.

c_dm_vblux

LU 分解された実バンド行列の連立 1 次方程式
`ierr = c_dm_vblux(b, fa, k, n, nh1, nh2, ip, &icon);`

1. 機能

LU 分解されたバンド行列を係数に持つ連立 1 次方程式を解きます.

$$\mathbf{LUx} = \mathbf{b}$$

ただし, **L** は下バンド幅 h_1 の単位下バンド行列, **U** は上バンド幅 $h(= \min(h_1+h_2, n-1))$の上バンド行列, **b** は n 次元の実定数ベクトルです. LU 分解された元の行列 **A** の次数, 上バンド幅, 下バンド幅は n, h_1 および h_2 です. $n > h_1 \geq 0$, $n > h_2 \geq 0$.

2. 引数

呼出し形式:

```
ierr = c_dm_vblux(b,(double*)fa, k, n, nh1, nh2, ip, &icon);
```

引数の説明:

`b`	`double b[n]`	入力	定数ベクトル **b**.
		出力	解ベクトル **x**.
`fa`	`double fa[n][k]`	入力	LU 分解された行列 **L** および **U** を格納します. 詳細については, 図 c_dm_vblux-1.を参照してください. 演算後, `fa[i][j]`, `i` = 0, ... , `n`–1, `j` = 2×`nh1`+`nh2`+1, ... , `k`–1, の値は保証されません.
`k`	`int`	入力	配列 `fa` の整合寸法 (≥ 2 × `nh1` + `nh2` + 1).
`n`	`int`	入力	行列 **A** の次数 n.
`nh1`	`int`	入力	下バンド巾の大きさ h_1.
`nh2`	`int`	入力	上バンド巾の大きさ h_2.
`ip`	`int ip[n]`	出力	行の入れ換えの履歴情報を格納するトランスポジションベクトルが格納されます.
`icon`	`int`	出力	コンディションコード. 下の表を参照.

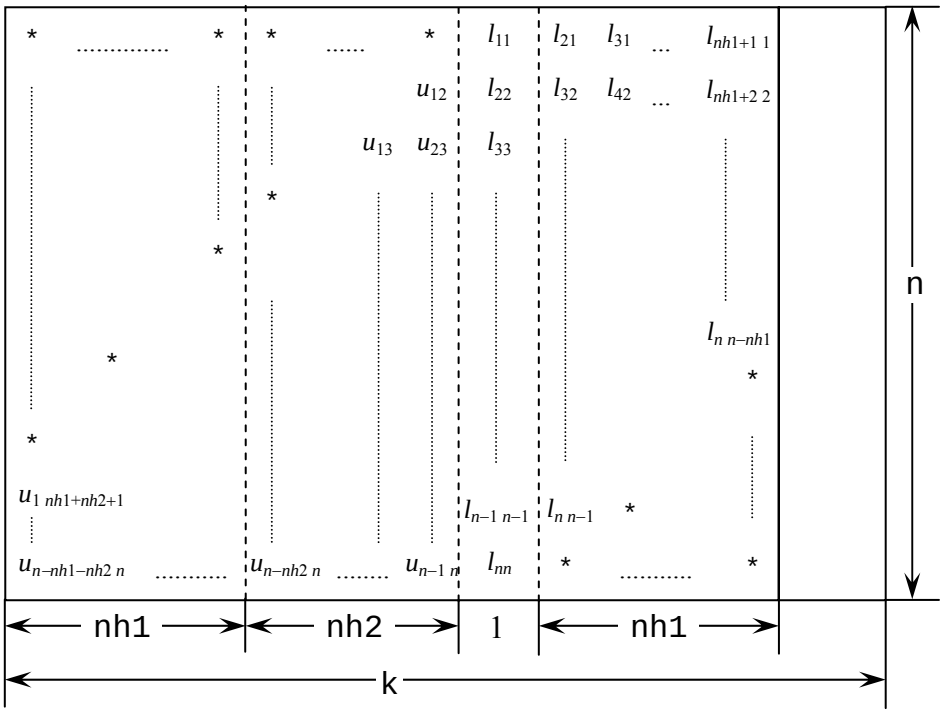


図 c_dm_vblux-1. LU 分解された行列 **L** および **U** の配列 fa に格納する方法

LU 分解された単位上バンド行列の対角要素を除いた部分, $u_{j-i+1,\,j}$, $i = 1, \ldots, h_1+h_2$, $j = 1, \ldots, n$, $j - i + 1 \geq 1$ が fa[i][j], i = 0, ... , n–1, j = 0 , ... , h_1+h_2,に格納されます.

下バンド行列, $l_{j+i,\,j}$, $i = 0, \ldots, h_2$, $j = 1, \ldots, n$, $j + i \leq n$ が fa[i][j], i = 0, ... , n–1, j = h_1+h_2, ... , 2×h_1+h_2 に格納されます.

*は不定値を示します.

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 • nh1 ≥ n • nh1 < 0 • nh2 ≥ n • nh2 < 0 • k < 2 × nh1 + nh2 + 1 • 下バンド行列の対角要素が零であった. • ip の内容が正しくない.	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_dm_vblux 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ます. 通常は関数 c_dm_vlbx を呼び出せば一度に解が求まります.

4. 使用例

$n = 10000, nh_1 = 2000, nh_2 = 3000$ の実バンド行列を入力して, バンド行列の連立 1 次方程式を解きます.

```
#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 NH1   2000
#define NH2   3000
#define N     10000
#define KA    (2*NH1+NH2+1)
#define NWORK 4500

int MAIN__()
{
  double a[N][KA], b[N], dwork[NWORK];
  double tt1, tt2, tmp, epsz;
  int    ip[N], i, j, is, ix, icon, nptr, nbase, nn;

  ix = 123;
  nn = NH1+NH2+1;
  for (i=0; i<N; i++) {
    c_dvrau4(&ix,&a[i][NH1],nn,dwork,NWORK,&icon);
  }

  printf("nh1 = %d, nh2 = %d, n = %d\n", NH1, NH2, N);

  /* zero clear */
  for (j=0; j<N; j++) {
    for (i=0; i<NH1; i++) {
      a[j][i] = 0.0;
    }
  }

  /* left upper triangular part */
  for (j=0; j<NH2; j++) {
    for (i=0; i<NH2-j; i++) {
      a[j][i+NH1] = 0.0;
    }
  }

  /* right rower triangular part */
  nbase = 2*NH1+NH2+1;
  for (j=0; j<NH1; j++) {
    for (i=0; i<j; i++) {
      a[N-NH1+j][nbase-i-1] = 0.0;
    }
  }

  /* set right hand constant vector */
  for (i=0; i<N; i++) {
    b[i] = 0.0;
  }

  for (i=0; i<N; i++) {
    nptr = i;
    for (j=max(nptr-NH2,0); j<min(N,nptr+NH1+1); j++) {
      b[j] += a[i][j-i+NH1+NH2];
    }
  }

  epsz = 0.0;
  c_dm_vblu((double*)a, KA, N, NH1, NH2, epsz, &is, ip, &icon);
  c_dm_vblux(b, (double*)a, KA, N, NH1, NH2, ip, &icon);

  tmp = 0.0;
  for (i=0; i<N; i++) {
    tmp = max(tmp,fabs(b[i]-1));
  }
```

```
    printf("maximum error = %e\n", tmp);
    return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VBLUX の項目を参照してください.

c_dm_vcgd

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_dm_vcgd(a, k, nw, n, ndlt, b, ipc,`
`itmax, isw, omega, eps, iguss, x,`
`&iter, &rz, w, iw, &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 法で解きます.

$$\mathbf{Ax} = \mathbf{b} \tag{1}$$

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します. ベクトル **b** 及び **x** は n 次元ベクトル.

2. 引数

呼出し形式:

```
ierr = c_dm_vcgd((double*)a, k, nw, n, ndlt, b, ipc, itmax, isw, omega, eps,
            iguss, x, &iter, &rz, (double*)w, (int*)iw, &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します. 演算後, 内容は保存されません.
k	int	入力	配列 a の整合寸法 (≥ 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 **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 **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します.
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します.
ipc	int	入力	前処理の制御情報. (“使用上の注意” c)参照). 1 前処理なし. 2 Neumann の前処理. 3 Block 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ります.
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” a)参照). 1 初回の呼出し. 2 2 回目以降の呼出し. ただし a, ndlt, w, iw 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りません.
omega	double	入力	不完全コレスキー分解のための修正値. 0 ≤ omega ≤ 1. ipc = 3 のとき使用されます. (“使用上の注意” c)参照).
eps	double	入力	収束判定に用いられる判定値. 0.0 が設定されたときは,

			$\varepsilon\cdot\|\mathbf{b}\|$ が eps として設定されます. (“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: 解ベクトルの近似を指定しません. 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します.
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ます. (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ます.
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” b)参照).
w	double w[*Wlen1*][*Wlen2*]	作業領域	1)ipc = 3 のとき *Wlen1* = nw+8 *Wlen2* = n+*maxt* 2)その他のとき *Wlen1* = 7 *Wlen2* = n+*maxt* *maxt* は並列実行する最大スレッド数.
iw	int iw[*Iwlen1*][*Iwlen2*]	作業領域	1)ipc = 3 のとき *Iwlen1* = 4 *Iwlen2* = n+2×*maxt* 2)その他のとき *Iwlen1* = 2 *Iwlen2* = *maxt* *maxt* は並列実行する最大スレッド数.
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	**A** の対角ベクトルを U/L の順に対角ベクトルの距離の昇順に格納し直した.	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. 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.
20003	ブレークダウンを起こした.	
30003	itmax ≤ 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 ≤ 0	
30093	k ≤ 0, n ≤ 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.	

コード	意味	処理内容
30103	下三角部分が正しく格納されていない.	処理を打ち切る.
30104	上三角の本数が, 下三角の本数と等しくない.	
30105	isw ≠ 1 , 2	
30200	|ndlt[i]| > n–1 又は ndlt[i] = 0 (0 ≤ i < nw)	

3. 使用上の注意

a) 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 で解きます. 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きます.

b) 収束判定

n 回目の反復で求められた解が収束したかどうかは, 次のように判定します.

r を残差ベクトル $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}_n$, **M** を前処理行列, $\mathtt{rz} = \mathbf{r}^{\mathrm{T}}\mathbf{M}^{-1}\mathbf{r}$ として,

$$\mathtt{rz} = \sqrt{rz} < \mathtt{eps}$$

を満たすとき, 収束したと判断します.

c) 前処理について

2 種類の前処理及び前処理なしの機能が提供されています. 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す.

$\mathbf{A} = \mathbf{I} - \mathbf{N}$ 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 の値により以下のようになります.

ipc=1　前処理なし, $\mathbf{M} = \mathbf{I}$.
ipc=2　Neumann, $\mathbf{M}^{-1} = (\mathbf{I}+\mathbf{N})$.
ipc=3　不完全コレスキー分解, $\mathbf{M} = \mathbf{L}\mathbf{L}^{\mathrm{T}}$.

ipc = 2 のときは, 前処理行列も正定値であることが必要です. 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ります. 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ります.

ipc = 3 のときは, **A** を並列実行するスレッド数でブロックに分割し, 各ブロックを不完全コレスキー分解したものを前処理 **M** とします. ipc = 3 のとき, omega ($0 \le \mathtt{omega} \le 1$)に値を設定する必要があります. omega=0 のときは, 不完全コレスキー分解, omega=1 のときは修正不完全コレスキー分解です.

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ます.

ipc = 3 のときは, 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ます.

4. 使用例

正値対称スパース行列の連立 1 次方程式を解きます.

```
#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 MAXT (4)
#define ND   (20)
#define N    (ND*ND*ND)
#define K    (N)
#define NW   (6)

MAIN__()
{
  double a[NW][K], b[N], x[N], w[7][N+MAXT];
  double omega, eps, rz;
  int    ndlt[NW], iw[2][MAXT];
  int    k, nw, n, ipc, itmax, isw, iguss, iter, icon;
  int    i, j, nx, ny, iy, iz, l;
  int    rhs(double*, int, int, int, double*, int*, double*);

  for(j=0; j<NW; j++) {
    for(i=0; i<N; i++) {
      a[j][i] = 0.0;
    }
  }

  for(i=0; i<NW; i++) {
    ndlt[i] = 0;
  }

  nx = ND;
  ny = ND;
  for(i=0; i<N; i++) {
    l  = i+1;
    iz = (l-1)/(nx*ny);
    iy = (l-1-iz*nx*ny)/nx;

    if ((l/nx)*nx != l && l <= N-1) {
      a[0][i] = -1.0/6.0;
    }
    if (l <= N-nx && iy != ny-1) {
      a[1][i] = -1.0/6.0;
    }
    if (l <= N-nx*ny) {
      a[2][i] = -1.0/6.0;
    }
    if (((l-1)/nx)*nx != l-1 && l >= 2 && l <= N) {
      a[3][i] = -1.0/6.0;
    }
    if (l >= nx+1 && l <= N && iy != 0) {
      a[4][i] = -1.0/6.0;
    }
    if (l >= nx*ny+1 && l <= N) {
      a[5][i] = -1.0/6.0;
    }
  }

  ndlt[0] =  1, ndlt[1] =  nx, ndlt[2] =  nx*ny;
  ndlt[3] = -1, ndlt[4] = -nx, ndlt[5] = -nx*ny;

  rhs((double*)a, N, K, NW, (double*)w, ndlt, b);

  eps   = 1e-6;
  itmax = 2000;
  isw   = 1;
  iguss = 0;
  ipc   = 2;

  c_dm_vcgd((double*)a, K, NW, N, ndlt, b, ipc, itmax, isw, omega, eps, iguss, x,
            &iter, &rz, (double*)w, (int*)iw, &icon);

  printf("icon = %d\n", icon);
  printf("x[0] = %e, x[n-1]= %e\n", x[0], x[N-1]);

  return(0);
}
```

```
int rhs(double *a, int n, int k, int ndiag, double *dp, int *ndlt, double *b)
{
  int i, nlb, icon;

  nlb = 0;
  for (i=0; i < ndiag; i++) {
    nlb = max(fabs(ndlt[i]), nlb);
  }

  for (i=0; i < n*3; i++) {
    dp[i] = 0.0;
  }

  for (i=0; i < n; i++) {
    dp[i + nlb] = 1.0;
    b[i] = 0.0;
  }

  c_dm_vmvsd((double*)a, k, ndiag, n, ndlt, nlb, dp, b, &icon);

  for (i = 0; i < n; i++) {
    b[i] += dp[i+nlb];
  }

  return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VCGD の項目及び[25], [45], [52], [53], [59]を参照してください.

c_dm_vcge

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_dm_vcge(a, k, nw, n, icol, b, ipc,`
`itmax, isw, omega, eps, iguss, x,`
`&iter, &rz, w, iw, &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 法で解きます.

$$\mathbf{Ax} = \mathbf{b} \tag{1}$$

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します. ベクトル **b** 及び **x** は n 次元ベクトル.

2. 引数

呼出し形式:

```
ierr = c_dm_vcge((double*)a, k, nw, n, (int*)icol, b, ipc, itmax, isw, omega,
            eps, iguss, x, &iter, &rz, (double*)w, (int*)iw, &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を ELLPACK 形式で格納します. ("使用上の注意" a)参照).
k	int	入力	配列 a, icol の整合寸法(≥ 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 *NSU* とし, 下三角行列の行ベクトルの非零要素の最大値を *NSL* とするとき, nw = 2·max(*NSU*, *NSL*).
n	int	入力	行列 **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の情報を格納します.
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します.
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 Block 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ります.
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, w, iw の値は 1 回目に設定された値を使用するため, 変更してはなりません.
omega	double	入力	不完全コレスキー分解のための修正値. 0 ≤ omega ≤ 1.
eps	double	入力	収束判定に用いられる判定値. eps=0.0 のとき, eps = $\varepsilon \cdot \|\mathbf{b}\|$ として設定されます. ε には 10^{-6} が設定されます. ("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ません. 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します.
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ます. (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ます.
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” b) 参照).
w	double w[*Wlen1*][*Wlen2*]	作業領域	1)ipc = 3 のとき *Wlen1* = nw+8 *Wlen2* = n+*maxt* 2)その他のとき *Wlen1* = 7 *Wlen2* = n+*maxt* *maxt* は並列実行する最大スレッド数.
iw	int iw[*Iwlen1*][*Iwlen2*]	作業領域	1) ipc=3 のとき *Iwlen1* = nw+5 *Iwlen2* = n+2×*maxt* 2)その他のとき *Iwlen1* = 2 *Iwlen2* = *maxt* *maxt* は並列実行する最大スレッド数.
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	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. 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.
20003	ブレークダウンを起こした.	
30003	itmax ≤ 0	処理を打ち切る.
30005	k < n	
30006	不完全 LL^T 分解が出来なかった.	
30007	ピボットが負になった.	
30092	nw ≤ 0	
30093	k ≤ 0, n ≤ 0	
30096	omega < 0 , omega > 1	
30097	ipc < 1 , ipc > 3	
30098	isw ≠ 1 , 2	
30100	nw ≠ 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.	

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します.

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ます. このとき, nw = 2·max(*NSU*, *NSL*)である必要はありません.

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 で解きます. 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ます.

c) 収束判定

n 回目の反復で求められた解が収束したかどうかは, 次のように判定します.

r を残差ベクトル $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}_n$, **M** を前処理行列, $\mathrm{rz} = \mathbf{r}^{\mathrm{T}}\mathbf{M}^{-1}\mathbf{r}$ として,

$$\mathrm{rz} = \sqrt{rz} < \mathrm{eps}$$

を満たすとき, 収束したと判断します.

d) 前処理について

2 種類の前処理及び前処理なしの機能が提供されています. 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す.

$\mathbf{A} = \mathbf{I} - \mathbf{N}$ 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 の値により以下のようになります.

ipc=1	前処理なし, $\mathbf{M} = \mathbf{I}$.
ipc=2	Neumann, $\mathbf{M}^{-1} = (\mathbf{I}+\mathbf{N})$.
ipc=3	不完全コレスキー分解, $\mathbf{M} = \mathbf{L}\mathbf{L}^{\mathrm{T}}$.

ipc = 2 のときは, 前処理行列も正定値であることが必要です. 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ります. 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ります.

ipc = 3 のときは, **A** を並列実行するスレッド数でブロックに分割し, 各ブロックを不完全コレスキー分解したものを前処理 **M** とします. ipc = 3 のとき, omega ($0 \le$ omega ≤ 1)に値を設定する必要があります. omega=0 のときは, 不完全コレスキー分解, omega=1 のときは修正不完全コレスキー分解です.

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ます.

ipc = 3 のときは, 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ます.

4. 使用例

正値対称スパース行列の連立 1 次方程式を解きます.

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

```
#define MAXT (4)
#define ND   (80)
#define N    (ND*ND*ND)
#define K    (N)
#define NW   (6)

MAIN__()
{
  double a[NW][K], b[N], x[N], xx[N], w[7][N+MAXT];
  double omega, eps, rz;
  int    icol[NW][K], iw[2][MAXT];
  int    ipc, itmax, isw, iguss, iter, icon;
  int    i, j, nx, ny, iy, iz, l;

  for(j=0; j<NW; j++) {
    for(i=0; i<N; i++) {
      a[j][i]    = 0.0;
      icol[j][i] = j+1;
    }
  }

  nx = ND;
  ny = ND;
  for(i=0; i<N; i++) {
    l  = i+1;
    iz = i/(nx*ny);
    iy = (i-iz*nx*ny)/nx;

    if ((l/nx)*nx != l && l <= N-1) {
      a[0][i]    = -1.0/6.0;
      icol[0][i] = l+1;
    }
    if (l <= N-nx && iy != ny-1) {
      a[1][i]    = -1.0/6.0;
      icol[1][i] = l+nx;
    }
    if (l <= N-nx*ny) {
      a[2][i]    = -1.0/6.0;
      icol[2][i] = l+nx*ny;
    }
    if (((l-1)/nx)*nx != l-1 && l >= 2 && l <= N) {
      a[3][i]    = -1.0/6.0;
      icol[3][i] = l-1;
    }
    if (l >= nx+1 && l <= N && iy != 0) {
      a[4][i]    = -1.0/6.0;
      icol[4][i] = l-nx;
    }
    if (l >= nx*ny+1 && l <= N) {
      a[5][i]    = -1.0/6.0;
      icol[5][i] = l-nx*ny;
    }
  }

  for (i=0; i<N; i++) {
    xx[i] = 1.0;
  }

  c_dm_vmvse((double*)a, K, NW, N, (int*)icol, xx, b, &icon);

  for (i=0; i<N; i++) {
    b[i] += 1.0;
  }

  itmax = 2000;
  eps   = 1e-6;
  isw   = 1;
  ipc   = 2;
  iguss = 0;

  c_dm_vcge((double*)a, K, NW, N, (int*)icol, b, ipc, itmax, isw, omega, eps, iguss,
            x, &iter, &rz, (double*)w, (int*)iw, &icon);

  printf("icon = %d\n", icon);
  printf("x[0] = %e, x[n-1]= %e\n", x[0], x[N-1]);
```

```
  return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VCGE の項目及び[25], [45], [53]を参照してください.

c_dm_vclu

複素行列の LU 分解 (ブロック化された LU 分解)
`ierr = c_dm_vclu(za, k, n, epsz, ip, &is, &icon);`

1. 機能

$n \times n$ の正則な複素数行列をブロック化した外積形ガウスの消去法により LU 分解します.

$$\mathbf{PA} = \mathbf{LU}$$

ただし, **P** は部分ピボッティング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す. ($n \geq 1$)

2. 引数

呼出し形式:

```
ierr = c_dm_vclu((dcomplex*)za, k, n, epsz, ip, &is, &icon);
```

引数の説明:

za	dcomplex	入力	行列 **A**.
	za[n][k]	出力	行列 **L** と行列 **U**.
k	int	入力	配列 za の整合寸法 (≥ 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ます. ("使用上の注意" a)参照).
ip	int ip[n]	出力	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" 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ます.
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.
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 < 0.0	

3. 使用上の注意

a) epsz について

epsz に値を設定した場合 , ピボットが epsz 以下なら零と見なし, icon = 20000 として処理を打ち切ります. epsz の標準値は丸め誤差の単位μとしたとき, epsz = 16μです. 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のですが, その結果は保証されません.

b) ip について

トランスポジションベクトルとは, 部分ピボッティングを行う LU 分解

$$\mathbf{PA} = \mathbf{LU}$$

における置換行列 **P** に相当します.

本関数では, 部分ピボッティングに伴い, 配列 za の内容を実際に交換しています. すなわち, 分解の J 段階目($J = 1, \ldots, n$)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 za の第 I 行と第 J 行の内容が交換されます. そして, その履歴を示すために ip[J –1]に I が格納されます.

c) 本関数の使用方法

本関数に続けて, 関数 c_dm_vclux 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ます. 通常は関数 c_dm_vlcx を呼び出せば一度に解が求まります.

4. 使用例

複素行列の連立 1 次方程式を LU 分解して解きます.

```
#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 N  (2000)
#define K  (N+1)

MAIN__()
{
  dcomplex za[N][K], zb[N];
  double   epsz, c, t, s, error;
  int      ip[N];
  int      is, icon, i, j;

  c = sqrt(1.0/(double)(N+1));
  t = atan(1.0)*8.0/(N+1);

  for (j=0; j<N; j++) {
    for (i=0; i<N; i++) {
      za[j][i].re = c*cos(t*(i+1)*(j+1));
      za[j][i].im = c*sin(t*(i+1)*(j+1));
    }
  }

  for (i=0; i<N; i++) {
    s = 0.0;
    for (j=0; j<N; j++) {
      s += cos(t*(i+1)*(j+1));
      zb[i].re = s*c;
      zb[i].im = 0.0;
    }
  }

  epsz = 0.0;
  c_dm_vclu((dcomplex*)za, K, N, epsz, ip, &is, &icon);
  c_dm_vclux(zb, (dcomplex*)za, K, N, ip, &icon);
```

```
  printf("icon    = %d\n", icon);

  error = 0.0;

  for (i=0; i<N; i++) {
    error = max(fabs(1.0-zb[i].re), error);
  }

  printf("error   = %f\n", error);
  printf("ORDER   = %d\n", N);
  printf("zb[0]   = %e\n", zb[0].re);
  printf("zb[n-1] = %e\n", zb[N-1].re);

  return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VCLU の項目及び[1], [30], [54], [55], [56], [70]を参照してください.

c_dm_vclux

LU 分解された複素行列の連立 1 次方程式
`ierr = c_dm_vclux(zb, zfa, kfa, n, ip, &icon);`

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

$$\mathbf{LUx} = \mathbf{Pb} \tag{1}$$

を解きます. 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, **P** は置換行列 (係数行列を LU 分解したときの部分ピボッティングによる行の入れ換えを行います), **b** は n 次元の複素定数ベクトル, **x** は n 次元の解ベクトル. ($n \geq 1$)

2. 引数

呼出し形式:

```
ierr = c_dm_vclux(zb, (dcomplex*)zfa, kfa, n, ip, &icon);
```

引数の説明:

`zb`	`dcomplex zb[n]`	入力 出力	定数ベクトル **b**. 解ベクトル **x**.
`zfa`	`dcomplex zfa[n][kfa]`	入力	行列 $\mathbf{L} + (\mathbf{U} - \mathbf{I})$.
`kfa`	`int`	入力	配列 `zfa` の整合寸法 ($\geq$ `n`).
`n`	`int`	入力	行列 **L**, **U** の次数 n.
`ip`	`int ip[n]`	出力	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.
`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 • `kfa` < `n` • `ip` に誤りがあった	

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_dm_vclu を呼び出して, 係数行列を LU 分解してから, 本関数を呼び出せば方程式を解くことができます. しかし, 通常は関数 c_dm_vlcx を呼び出せば一度に解が求まります.

4. 使用例

複素行列の連立 1 次方程式を LU 分解して解きます.

```
#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 N  (2000)
#define K  (N+1)

MAIN__()
{
  dcomplex za[N][K], zb[N];
  double   epsz, c, t, s, error;
  int      ip[N];
  int      is, icon, i, j;

  c = sqrt(1.0/(double)(N+1));
  t = atan(1.0)*8.0/(N+1);

  for (j=0; j<N; j++) {
    for (i=0; i<N; i++) {
      za[j][i].re = c*cos(t*(i+1)*(j+1));
      za[j][i].im = c*sin(t*(i+1)*(j+1));
    }
  }

  for (i=0; i<N; i++) {
    s = 0.0;
    for (j=0; j<N; j++) {
      s += cos(t*(i+1)*(j+1));
      zb[i].re = s*c;
      zb[i].im = 0.0;
    }
  }

  epsz = 0.0;
  c_dm_vclu((dcomplex*)za, K, N, epsz, ip, &is, &icon);
  c_dm_vclux(zb, (dcomplex*)za, K, N, ip, &icon);

  printf("icon    = %d\n", icon);

  error = 0.0;

  for (i=0; i<N; i++) {
    error = max(fabs(1.0-zb[i].re), error);
  }

  printf("error   = %f\n", error);
  printf("ORDER   = %d\n", N);
  printf("zb[0]   = %e\n", zb[0].re);
  printf("zb[n-1] = %e\n", zb[N-1].re);

  return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VCLUX の項目及び[54]を参照してください.

c_dm_vcminv

複素行列の逆行列 (ブロック化された Gauss-Jordan 法)
`ierr = c_dm_vcminv(za, k, n, epsz, &icon);`

1. 機能

Gauss-Jordan 法により正則な $n \times n$ 複素行列 **A** の逆行列 $\mathbf{A}^{-1}$ を求めます.

2. 引数

呼出し形式:

```
ierr = c_dm_vcminv((dcomplex*)za, k, n, epsz, &icon);
```

引数の説明:

za	dcomplex	入力	行列 **A**.
	za[n][k]	出力	行列 $\mathbf{A}^{-1}$.
k	int	入力	配列 za の整合寸法 (≥ 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 が入力されたときは, 標準値が採用されます. ("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.
20000	行列 **A** のある行の要素がすべて零であったか, またはピボットが相対的に零となった. 行列 **A** は非正則である可能性が強い.	処理を打ち切る.
30000	次のいずれかであった. • n < 1 • k < n • epsz < 0.0	

3. 使用上の注意

a) epsz について

部分ピボッティングで選択されたピボット要素が 0.0 か絶対値が epsz 以下なら, 相対的に零とみなして, icon = 20000 として処理を打ち切ります. epsz の標準値は丸め誤差の単位を u としたとき, $16u$ です. epsz に極小の値を設定すると, ピボットの絶対値が小さくなっても処理は続行されますが, 計算結果の精度は保証されません.

4. 使用例

逆行列を求めます.

```
#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 N 2000
#define K (N+1)

int MAIN__()
{
  dcomplex a[N][K], as[N][K], tmpz;
  double   c, t, error, epsz;
  int      i, j, icon;

  c = sqrt(1.0/(double)N);
  t = atan(1.0)*8.0/N;

  for (j=0; j<N; j++) {
    for (i=0; i<N; i++) {
      a[j][i].re  =  c*cos(t*i*j);
      a[j][i].im  =  c*sin(t*i*j);
      as[j][i].re =  a[j][i].re;
      as[j][i].im = -a[j][i].im;
    }
  }

  epsz = 0.0;
  c_dm_vcminv((dcomplex*)a, K, N, epsz, &icon);

  error = 0.0;
  for (j=0; j<N; j++) {
    for (i=0; i<N; i++) {
      tmpz.re = fabs(a[j][i].re-as[j][i].re);
      tmpz.im = fabs(a[j][i].im-as[j][i].im);
      error = max(error,tmpz.re+tmpz.im);
    }
  }

  printf("order = %d, error = %e\n", N, error);
  return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VCMINV の項目を参照してください.

c_dm_vgevph

実対称行列の一般化固有値問題 (固有値・固有ベクトル) (三重対角化, マルチセクション法, 逆反復法)

`ierr = c_dm_vgevph(a, k, n, b, epsz, nf, nl,`
`ivec, &etol, &ctol, nev, e, maxne,`
`m, ev, &icon);`

1. 機能

一般化固有値問題の指定された固有値を求めます. 指定に応じて, 固有ベクトルを求めます.

$$\mathbf{Ax} = \lambda\mathbf{Bx}$$

A は $n \times n$ の実対称行列, **B** は $n \times n$ の正値対称行列です.

2. 引数

呼出し形式:

```
ierr = c_dm_vgevph((double*)a, k, n, (double*)b, epsz, nf, nl, ivec, &etol,
            &ctol, nev, e, maxne, (int*)m, (double*)ev, &icon);
```

引数の説明:

a	double a[n][k]	入力	a の上三角部分{a[i−1][j−1], i ≤ j}に, 実対称行列 **A** の上三角部分$\{a_{ij} \mid i \le j\}$を格納します. 演算後の内容は保証されません.
k	int	入力	配列 a の整合寸法 (k ≥ n).
n	int	入力	実対称行列 **A** の次数 n.
b	double b[n][k]	入力	b の上三角部分{b[i−1][j−1], i ≤ j}に, 正値対称行列 **B** の上三角部分$\{b_{ij} \mid i \le j\}$を格納します.
		出力	**B** を LL^T 分解した結果が格納されます. b の上三角部分{b[i−1][j−1], i ≤ j}に, 上三角行列 **L**$\{l_{ij} \mid i \le j\}$が格納されます.
epsz	double	入力	**B** の LL^T 分解におけるピボットの零判定値 (≥ 0.0) 0.0 のときは標準値が採用されます. (“使用上の注意”a)参照)
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ます. ivec ≠ 1 のとき固有値のみ求めます.
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.
		出力	etol < 3×10^{-16} のときは, etol = 3×10^{-16} が標準値として設定されます. (“使用上の注意”b)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(≥ etol). 近似的多重固有値(cluster)に属する固有値の対応する固有ベクトルの 1 次独立性を保証するために使用されます. 5.0×10^{-12} が一般的に適当です. ただし非常に大きなクラスタに対しては, 大きな値が必要です.

			$10^{-6} \geq$ ctol $\geq$ etol.
		出力	ctol $> 10^{-6}$ のときは, ctol $= 10^{-6}$ と設定されます. ctol $<$ etol のときは, ctol $= 10 \times$ etol が標準値として設定されます. ("使用上の注意"b)参照)
nev	int nev[5]	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. nev[3]は, 求めた固有値の内最初の固有値の番号. nev[4]は, 求めた固有値の内最後の固有値の番号.
e	double e[maxne]	出力	固有値が格納されます. 求められた固有値は e[i–1], i = 1, ... , nev[2]に格納されます.
maxne	int	入力	計算できる固有値の最大個数. 配列 e の大きさ. 多重度 m の固有値がいくつかあると考えられるとき, n を上限として nl – nf + 1 + 2 × m を設定する必要があります. nev[2] > maxne のとき, 固有ベクトルの計算はできません. ("使用上の注意" c)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します. ("使用上の注意" b)参照)
ev	double ev[maxne][k]	出力	ivec = 1 のとき, 固有値に対応して固有ベクトルが格納されます. 求められた固有ベクトルは ev[i–1][j–1], i = 1,...,nev[2], j = 1, ... ,n に格納されます.
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.
20000	**B** の LL^T分解においてピボットが負となった. 行列 **B** は正値ではない.	処理を打ち切る.
20100	**B** の LL^T分解においてピボットが相対的に零となった. 行列 **B** は非正則の可能性が強い.	
20200	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. nev[2]に maxne に設定するべき大きさが返却される. ("使用上の注意" c)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 • epsz < 0	処理を打ち切る.

3. 使用上の注意

a) epsz

epsz が設定されると, LL^T 分解の過程でピボットが epsz 以下のときに相対的に零と見なし, icon = 20100 で処理を打ち切ります. epsz の標準値は, まるめ誤差単位を u としたとき $16u$ です. なおピボットが小さくても処理を続行させたいときには, epsz に極小の値を設定すればよいですが, その結果は保証されません.

b) etol 及び ctol

本ルーチンは, 求める固有値を交わりのない順序付けられた集合に分けて独立に計算することにより並列化しています.

ε = etol としたとき, 連続する固有値λ_j , $j = s-1, s, \ldots, s+k$, $(k \geq 0)$に対して,

$$\frac{|\lambda_i - \lambda_{i-1}|}{1 + \max(|\lambda_{i-1}|, |\lambda_i|)} \leq \varepsilon \ , \tag{1}$$

$i = s, s+1, \ldots, s+k$ 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, $j = s-1, s, \ldots, s+k$ は数値的に多重であると見なされます.

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ます.

(1)が ε =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ます.

ε =etol で異なる固有値と見なされた連続する固有値λ_m, $m = t-1, t, \ldots, t+k$, $(k \geq 0)$に対して, ε =ctol としたとき, $i = t, t+1, \ldots, t+k$ 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, $m = t-1, t, \ldots, t+k$ を近似的に多重(cluster)と見なします. 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ます.

c) maxne

maxne に, 計算できる固有値の最大個数を指定できます. ctol を大きくすると, cluster の大きさが大きくなり, 計算する固有値の総数が maxne を超えるかも知れません. この場合, ctol を小さくするか, maxne の大きくする必要があります.

計算する固有値の総数が maxne を超えた場合, icon＝20200 を返却します. 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ません. 固有値は求められていますが, 多重度分だけ繰り返して格納はされてはいません.

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ます.

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 は求める固有値の総数を nt (nt = nl – nf + 1)として nt で十分です. 多重な固有値がいくつかあり, 多重度が m と考えられるときは, 少なくとも maxne は $nt + 2 \times m$ とする必要があります.

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値が nev[2]に返却されます. この値で領域を確保して再度呼び出すことで計算を続けることができます.

4. 使用例

固有値・固有ベクトルが分かっている一般化固有値問題の指定された固有値・固有ベクトルを求めます.

```
#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 N  2000
#define K  (N+1)
#define NF 1
#define NL N
#define MAXNE (NL-NF+1)


int MAIN__()
{
    double a[N][K], b[N][K], b2[N][K], c[N][K], d[N][K];
    double e[MAXNE], ev[MAXNE][K];
    double pai, coef, ctol, etol, epsz, temp;
    int    nev[5], m[2][MAXNE];
    int    i, j, k, ivec, icon;

    pai  = atan(1.0) * 4.0;
    coef = sqrt(2.0/(N+1));

    for (j=0; j<N; j++) {
      for (i=0; i<N; i++) {
        d[j][i] = coef*sin(pai/(N+1)*(i+1)*(j+1));
      }
    }

    for (j=0; j<N; j++) {
      for (i=0; i<N; i++) {
        if (i==j) { c[j][i] = (double)(j+1); }
        else      { c[j][i] = 0.0; }
      }
    }

    c_dm_vmggm((double*)d, K, (double*)c, K, (double*)b, K, N, N, N, &icon);
    c_dm_vmggm((double*)b, K, (double*)d, K, (double*)a, K, N, N, N, &icon);

    /* B = LL^t , A <- LALt */
    for (i=0; i<N; i++) {
      for (j=0; j<N; j++) {
        b[i][j]  = 1.0/sqrt(1.0);
        b2[i][j] = min(i+1,j+1)/1.0;
      }
    }

    for (j=0; j<N; j++) {
      for (k=N-1; k>=0; k--) {
        temp = a[j][k];
        a[j][k] *= b[k][k];
        for (i=k+1; i<N; i++) {
          a[j][i] += temp*b[k][i];
        }
      }
    }

    for (j=N-1; j>=0; j--) {
      temp = b[j][j];
      for (i=0; i<N; i++) {
        a[j][i] *= temp;
      }
      for (k=0; k<j; k++) {
        temp=b[j][k];
        for (i=0; i<N; i++) {
          a[j][i] += temp*a[k][i];
        }
      }
    }

    ivec     = 1;
    etol     = 1.0e-15;
    ctol     = 1.0e-10;
    epsz     = 0;

    c_dm_vgevph((double*)a, K, N, (double*)b2, epsz, NF, NL, ivec, &etol, &ctol,
                nev, e, MAXNE, (int*)m, (double*)ev, &icon);

    for (i=0; i<nev[2]; i+=nev[2]/10) {
      printf("eigen value in e[%d] = %f\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 スレッド並列機能使用手引書”の DM_VGEVPH の項目を参照してください.

c_dm_vhevp

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)

`ierr = c_dm_vhevp(za, k, n, nf, nl, ivec,`
`&etol, &ctol, nev, eh, maxne, m,`
`zev, &icon);`

1. 機能

n 次のエルミート行列の指定されたいくつかの固有値を求めます. 指定に応じて対応する固有ベクトルを計算します.

$$\mathbf{Ax} = \lambda\mathbf{x}$$

2. 引数

呼出し形式:

```
ierr = c_dm_vhevp((dcomplex*)za, k, n, nf, nl, ivec, &etol, &ctol, nev, eh,
            maxne, (int*)m, (dcomplex*)zev, &icon);
```

引数の説明:

`za`	`dcomplex za[n][k]`	入力	`za` の上三角部分{`za[i-1][j-1]`, `i` ≤ `j`}に固有値・固有ベクトルを求めるエルミート行列 **A** の上三角部分$\{a_{ij} \mid i \le j\}$を格納します. 演算後の内容は保証されません.
`k`	`int`	入力	配列 `za` の整合寸法 (`k` ≥ `n`).
`n`	`int`	入力	エルミート行列 **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ます. `ivec` ≠ 1 のとき固有値のみ求めます.
`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.
		出力	`etol` < 3×10^{-16} のときは, `etol` = 3×10^{-16} が標準値として設定されます. ("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(≥ `etol`). 近似的多重固有値(cluster)に属する固有値の対応する固有ベクトルの 1 次独立性を保証するために使用されます. 5.0×10^{-12} が一般的に適当です. ただし非常に大きなクラスタに対しては, 大きな値が必要です. $10^{-6} \ge$ `ctol` ≥ `etol`.
		出力	`ctol` > 10^{-6} のときは, `ctol` = 10^{-6} と設定されます. `ctol` < `etol` のときは, `ctol` = 10 × `etol` が標準値として設定されます. ("使用上の注意"a)参照)

nev	int nev[5]	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. nev[3]は, 求めた固有値の内最初の固有値の番号. nev[4]は, 求めた固有値の内最後の固有値の番号.
eh	double eh[maxne]	出力	固有値が格納されます. 求められた固有値は eh[i–1], i = 1, ... , nev[2]に格納されます.
maxne	int	入力	計算できる固有値の最大個数. 配列 eh の大きさ. 多重度 m の固有値がいくつかあると考えられるとき, n を上限として $\mathtt{nl}-\mathtt{nf}+1+2\times m$ を設定する必要があります. nev[2] > maxne のとき, 固有ベクトルの計算はできません. ("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します. ("使用上の注意" b)参照)
zev	dcomplex zev[maxne][k]	出力	ivec = 1 のとき, 固有値に対応して固有ベクトルが格納されます. 求められた固有ベクトルは zev[i–1][j–1], i = 1,...,nev[2], j = 1, ... ,n に格納されます.
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.
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.

3. 使用上の注意

a) etol 及び ctol

本ルーチンは, 求める固有値を交わりのない順序付けられた集合に分けて独立に計算することにより並列化しています.

ε = etol としたとき, 連続する固有値λ_j, $j = s, s+1, \ldots, s+k$, $(k \geq 0)$に対して,

$$\frac{|\lambda_i - \lambda_{i-1}|}{1+\max(|\lambda_{i-1}|,|\lambda_i|)} \leq \varepsilon, \tag{1}$$

$i = s, s+1, \ldots, s+k$ 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, $j = s-1, s, \ldots, s+k$ は数値的に多重であると見なされます.

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ます.

(1)が ε =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ます.

ε =etol で異なる固有値と見なされた連続する固有値 λ_m, $m = t-1, t, \ldots, t+k$, $(k \geq 0)$ に対して, ε =ctol としたとき, $i = t, t+1, \ldots, t+k$ 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, $m = t-1, t, \ldots, t+k$ を近似的に多重(cluster)と見なします. 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ます.

b) maxne

maxne に, 計算できる固有値の最大個数を指定できます. ctol を大きくすると, cluster の大きさが大きくなり, 計算する固有値の総数が maxne を超えるかも知れません. この場合, ctol を小さくするか, maxne の大きくする必要があります.

計算する固有値の総数が maxne を超えた場合, icon＝20000 を返却します. 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ません. 固有値は求められていますが, 多重度分だけ繰り返して格納はされてはいません.

つまり, 固有値に関しては, 求められた異なる固有値が eh[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ます.

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 は求める固有値の総数を nt (nt = nl – nf + 1)として nt で十分です. 多重な固有値がいくつかあり, 多重度が m と考えられるときは, 少なくとも maxne は $nt + 2 \times m$ とする必要があります.

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値が nev[2]に返却されます. この値で領域を確保して再度呼び出すことで計算を続けることができます.

4. 使用例

エルミート行列の固有値を計算します.

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

#define N              512
#define K                N
#define NF               1
#define NL              28
#define MAXNE      NL-NF+1

MAIN__()
{
  dcomplex za[N][K], zev[MAXNE][K];
  double   eh[MAXNE];
  double   etol, ctol;
  int      nev[5], m[2][MAXNE];
  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++) {
      za[i][j].re = (double)(i+j+2)/(double)n;
      if(i==j) {
        za[i][j].im = 0.0;
        za[i][j].re = (double)(j+1);
      } else {
        za[i][j].im = (double)((i+1)*(j+1))/(double)(n*n);
      }
    }
  }
  for(i=0; i<n; i++) {
    for(j=0; j<n; j++) {
      if(i > j) za[i][j].im = -za[i][j].im;
    }
  }
  /* Call complex eigensolver */
  ierr = c_dm_vhevp ((dcomplex*)za, k, n, nf, nl, ivec, &etol, &ctol, nev, eh,
                  maxne, (int*)m, (dcomplex*)zev, &icon);
  if (icon > 20000) {
    printf("ERROR: c_dm_vhevp 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("  eh[%d] = %12.4e\n", i, eh[i]);
  }
  return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VHEVP の項目及び[61]を参照してください.

c_dm_vhtrid

エルミート行列の実対称三重対角行列への変換 (ハウスホルダー法)
`ierr = c_dm_vhtrid(za, k, n, d, sl, zs, &icon);`

1. 機能

エルミート行列を Householder 変換でエルミート三重対角行列 **H** に変換し, 更に対角ユニタリ変換で実三重対角行列 **T** に変換します.

$$\mathbf{H} = \mathbf{P}^{*}\mathbf{A}\mathbf{P}$$
$$\mathbf{T} = \mathbf{V}^{*}\mathbf{H}\mathbf{V}$$

A は $n \times n$ エルミート行列, **P** は $n \times n$ ユニタリ行列, **V** は $n \times n$ の対角ユニタリ行列, **T** は実三重対角行列です.

2. 引数

呼出し形式:

```
ierr = c_dm_vhtrid((dcomplex*)za, k, n, d, sl, zs, &icon);
```

引数の説明:

`za`	`dcomplex za[n][k]`	入力	`za` の上三角部分{`za[`i`-1][`j`-1]` , $i \le j$}にエルミート行列 **A** の上三角部分$\{a_{ij} \mid i \le j\}$を格納します.
		出力	`za[`i`-1][`j`-1]`, i=1, … ,n, j=1, … ,n-2 の上三角部分{`za[`i`-1][`j`-1]` , $i \le j$}にエルミート三重対角化を行うために利用した Householder 変換に関する情報が格納されます. 演算後, 下三角部分の値は保証されません. (“使用上の注意”a)参照)
`k`	`int`	入力	配列 `za` の整合寸法 ($k \ge n$).
`n`	`int`	入力	エルミート行列 **A** の次数 n.
`d`	`double d[n]`	出力	三重対角化された三重対角行列の対角要素を格納します.
`sl`	`double sl[n]`	出力	三重対角化された三重対角行列の下副対角要素を`sl[`i`-1]`, i = 2, ... , n に格納します. `sl[0]` = 0.
`zs`	`dcomplex zs[n]`	出力	`zs[`i`-1]`, i= 1, … , n に, 対角ユニタリ行列の対角要素が格納されます.
`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	k<n, n<2	処理を打ち切る.

3. 使用上の注意

a) za について

k = 1, ... , n-2 まで, 以下の変換を繰り返してエルミート三重対角化します.

$$\mathbf{A}^{k} = \mathbf{P}_{k}^{*}\mathbf{A}^{k-1}\mathbf{P}_{k}\ , \quad \mathbf{A}^{0} = \mathbf{A}$$

$\mathbf{b}^{\mathrm{T}} = (0, \ldots, 0, \mathbf{A}^{k}(k+1, k), \ldots, \mathbf{A}^{k}(n, k))$ とおきます.($\mathbf{A}^{k}(i, j)$は $\mathbf{A}^{k}$ の i, j 要素)

$\mathbf{b}^{\mathrm{T}} = (0, \ldots, 0, b_{k+1}, \ldots, b_{n})$

$\mathbf{b}^{*}\cdot\mathbf{b} = \mathrm{S}^2$ として, $\mathbf{w}^{\mathrm{T}} = (0, \ldots, 0, b_{k+1}\left(1+\frac{|S|}{|b_{k+1}|}\right), b_{k+2}, \ldots, b_n)$とおきます.

すると, $\mathbf{P}_k = \mathrm{I} - \alpha\mathbf{w}\cdot\mathbf{w}^{*}, \alpha = \frac{1}{\mathrm{S}^2 + |b_{k+1}\,\mathrm{S}|}$ と表せます.

za[*k*-1][*i*-1]に $\mathbf{w}(i\text{-}1)$ ($i=k+1, \ldots, n$) ,また za[*k*-1][*k*-1]にαを格納します.

エルミート三重対角行列は, ユニタリ対角行列で実三重対角化します.

$$\mathbf{T} = \mathbf{V}^{*}\mathbf{H}\mathbf{V}$$

4. 使用例

固有値の分かっているエルミート行列を三重対角化します.

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

#define N               2000
#define K               N
#define NE              N
#define MAX_NEV         NE

MAIN__()
{
    dcomplex a[N][K],b[N][K],c[N][K],d[N][K],dh[N][K];
    dcomplex alpha,beta,tr[N];
    double   eval[MAX_NEV],evec[MAX_NEV][K],dd[N],sld[N],sud[N];
    double   pai2, coef, part1, part2, eval_tol, clus_tol;
    int      nev[5],mult[2][MAX_NEV];
    int      i, j, k, n, nf, nl, ivec, icon ,in, im, ik;

    n    = N;
    k    = K;


    pai2 = 8.0 * atan(1.0);
    coef = sqrt(1.0/(N));
    for (j=0; j<N; j++) {
      for (i=0; i<N; i++) {
        part1 = coef * cos(pai2/N*i*j);
        part2 = coef * sin(pai2/N*i*j);
        d[i][j].re = part1;
        d[i][j].im = part2;
        dh[i][j].re = part1;
        dh[i][j].im = -part2;
      }
    }

    for (j=0; j<N; j++) {
      for (i=0; i<N; i++) {
        if (i == j) {
          c[i][j].re = (double)(i+1);
          c[i][j].im = 0.0;
        }
        else {
          c[i][j].re = 0.0;
          c[i][j].im = 0.0;
        }
      }
    }

    /* d x c -> b */
    for (im=0; im<N; im++) {
      for (in=0; in<N; in++) {
        b[im][in].re = 0.0;
        b[im][in].im = 0.0;
      }
      for (ik=0; ik<N; ik++) {
        for (in=0; in<N; in++) {
          b[im][in].re = b[im][in].re + d[im][ik].re * c[ik][in].re
- d[im][ik].im * c[ik][in].im;
```

```
          b[im][in].im = b[im][in].im + d[im][ik].re * c[ik][in].im
+ c[ik][in].re * d[im][ik].im;
        }
      }
    }

    /* b x dh -> a */
    for (im=0; im<N; im++) {
      for (in=0; in<N; in++) {
        a[im][in].re = 0.0;
        a[im][in].im = 0.0;
      }
      for (ik=0; ik<N; ik++) {
        for (in=0; in<N; in++) {
          a[im][in].re = a[im][in].re + b[im][ik].re * dh[ik][in].re
- b[im][ik].im * dh[ik][in].im;
          a[im][in].im = a[im][in].im + b[im][ik].re * dh[ik][in].im
+ dh[ik][in].re * b[im][ik].im;
        }
      }
    }

    c_dm_vhtrid((dcomplex*)a, K, N, dd, sld, tr, &icon);

    if (icon != 0) {
      printf(" icon of c_dm_vhtrid =%d\n", icon);
      exit(0);
    }

    for (i=1; i<N; i++) {
      sud[i-1]=sld[i];
    }
    sud[N-1]=0.0;

    nf=1;
    nl=N;
    ivec=0;
    eval_tol=1.0e-15;
    clus_tol=1.0e-10;
    c_dm_vtdevc(dd, sld, sud, N, nf, nl, ivec, &eval_tol, &clus_tol,
                nev, eval, MAX_NEV, (double*)evec, K, (int*)mult, &icon);

    for (i=0; i<NE; i=i+N/20) {
      printf("eigen value in eval(%d) = %f\n",i+1,eval[i]);
    }

    return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VHTRID の項目を参照してください.

c_dm_vjdhecr

エルミートスパース行列の固有値問題 (Jacobi-Davidson 法, 圧縮行格納法)

`ierr = c_dm_vjdhecr(zh, nz, ncol, nfrnz, n,`
`itrgt, dtrgt, nsel, &nev, itmax,`
`&iter, iflag, dprm, deval, zevec, kv, dhis,`
`kh, &icon);`

1. 機能

$n \times n$ のエルミートなスパース行列の指定されたいくつかの固有値および対応する固有ベクトルを Jacobi-Davidson 法で求めます.

$$\mathbf{Ax} = \lambda \mathbf{x}$$

エルミート行列 **A** は, 下三角部分を圧縮行格納法で格納します. ベクトル **x** は n 次元ベクトルです.

2. 引数

呼出し形式:

```
ierr = c_dm_vjdhecr(zh, nz, ncol, nfrnz, n, itrgt, dtrgt, nsel, &nev, itmax,
            &iter, iflag, dprm, deval, (dcomplex*)zevec, kv, (double*)dhis,
            kh, &icon);
```

引数の説明:

`zh`	`dcomplex zh[nz]`	入力	行列 **A** の下三角部分の非零要素を格納します. 圧縮行格納法については, 図 c_dm_vjdhecr-1 を参照してください.
`nz`	`int`	入力	行列 **A** の下三角部分にある非零要素の総数.
`ncol`	`int ncol[nz]`	入力	圧縮行格納法で使用される列指標で, `zh` に格納される要素が何番目の列ベクトルに属するかを示します.
`nfrnz`	`int nfrnz[n+1]`	入力	行列 **A** の各行の下三角部分の非零要素を行方向に圧縮して順次配列 `zh` に格納するとき, 対応する行の最初の非零要素が格納される位置を表します. `nfrnz[n]` = `nz+1` として指定します.
`n`	`int`	入力	行列 **A** の次数 n.
`itrgt`	`int`	入力	求める固有値の選択方法を指定します($0 \le$ `itrgt` ≤ 4). `itrgt` = 0: `dtrgt` に指定したターゲット値の近傍の固有値を優先的に選択します. `itrgt` = 1: 絶対値の大きい固有値を優先的に選択します. `itrgt` = 2: 絶対値の小さい固有値を優先的に選択します. `itrgt` = 3: 数値が大きい固有値を優先的に選択します. `itrgt` = 4: 数値が小さい固有値を優先的に選択します. ("使用上の注意"a),b)参照)

dtrgt	double	入力	itrgt = 0 のとき, 固有値を選択するターゲット値を指定します. また, itrgt ≠ 0 であっても以下の場合には, 求める固有値の近傍値を dtrgt に指定することで収束が改善することがあります. 1) dprm[2] = 1 で harmonic アルゴリズムを選択しているとき, テスト副空間<**W**> = <(**A** − τ **I**)**V**> を設定するためのシフト値 τ として dtrgt を使用します. ("使用上の注意"b)参照) 2) dprm[8] ≥ 1 のとき, 修正方程式で用いる近似固有値として,反復開始初期に dtrgt への指定値を使用します. ("使用上の注意"e)参照) 3) dprm[14] ≥ 1 のとき, 修正方程式の前処理行列を設定するためのシフト値 τ として dtrgt への指定値を利用します. ("使用上の注意"g)参照) それ以外の場合には, dtrgt の値は参照されません.
nsel	int	入力	求める固有値個数を指定します(1 ≤ nsel ≤ n). ("使用上の注意"a)参照)
nev	int	出力	収束した固有値個数.
itmax	int	入力	反復回数の上限を指定します(≥ 0).
iter	int	出力	Jacobi-Davidson 法での実際の反復回数.
iflag	int iflag[32]	入力	dprm に動作調整パラメタを明に指定するかどうかを示します. iflag[i] ≠ 0 のとき, dprm[i]に指定したパラメタを使用します($i \le 14$). iflag[i] = 0 のとき, dprm[i]は参照せずデフォルト値を使用します. iflag[15] ~ [31]は機能拡張用のリザーブ域のため, 0 を指定しておきます.
dprm	double dprm[32]	入力	iflagで指定された動作調整用パラメタについて, 値を指定します. アルゴリズムにおける各パラメタ変数の定義については"FUJITSU SSL II スレッド並列機能使用手引書"のDM_VJDHECRの"手法概要"を参照してください. iflag[0] ~ [31]が全て0であればdprmは参照されず, デフォルト値が用いられます. デフォルト値のままでは収束しなかった場合に, パラメタを変えて試みることを奨めます. dprm[0]: リスタート時に縮小する副空間次元m_{min}を指定します. ($1 \le m_{min} < n$). デフォルト値は$m_{min} = 50$です. dprm[1]: 副空間次元の最大次元m_{max}を指定します ($m_{min} < m_{max} \le n$). デフォルト値は$m_{max} = m_{min} + 30$です. ("使用上の注意"h)参照) dprm[2]: 射影するテスト副空間の設定方法による, アルゴリズムの種別を指定します. dprm[2] = 0のとき, 固有値選択ターゲットがスペクトル端の場合に適したstandardアルゴリズムを使用します. dprm[2] = 1のとき, 固有値選択ターゲットがスペクトル内部の場合に適したharmonicアルゴリズムを使用します. デフォルトでは, itrgt = 0および2のときに

harmonicアルゴリズムを使用し, それ以外のときにstandardアルゴリズムを使用します.

dprm[3]: 収束判定許容閾値を指定します. デフォルトは10^{-6}です.
("使用上の注意"d)参照)

dprm[4]: 反復中に得られた近似固有値θおよび近似固有ベクトル$\boldsymbol{u}$に対し, 残差ノルムを計算する方法を指定します.
dprm[4] = 0のとき, 近似固有値絶対値に対する相対残差ノルム$|\mathbf{Au}-\theta\mathbf{u}| / |\theta|$を使用します.
dprm[4] = 1のとき, 行列**A**の1-ノルムに対する相対残差ノルム$|\mathbf{Au}-\theta\mathbf{u}| / |\mathbf{A}|_1$を使用します.
dprm[4] = 2のとき, 行列**A**のFrobeniusノルムに対する相対残差$|\mathbf{Au}-\theta\mathbf{u}| / |\mathbf{A}|_{\mathrm{Fro}}$を使用します.
dprm[4] = 3のとき, 行列**A**の∞-ノルムに対する相対残差ノルム$|\mathbf{Au}-\theta\mathbf{u}| / |\mathbf{A}|_\infty$を使用します.
dprm[4] = 4のとき, 絶対残差ノルム$|\mathbf{Au}-\theta\mathbf{u}|$を使用します.
デフォルトは, dprm[4] = 0です.
("使用上の注意"c)参照)

dprm[5]: 遅延減次処理における閾値を指定します(≤1.0).
デフォルトはdprm[5] = 0.9です.
("使用上の注意"d)参照)

dprm[6]: 反復開始初期ベクトルをzevec[0][i-1], i = 1, ... , nに指定するかどうかを指定します.
dprm[6] = 0.0のとき, ルーチン内部で生成した乱数ベクトルを反復開始初期ベクトルとして使用します.
dprm[6] = 1.0のとき, zevec[0][i-1], i = 1, ... , nに反復開始初期ベクトルを指定してください.
デフォルトは乱数ベクトルを使用します.

dprm[7]: 乱数列生成のための種(*seed*)の値を指定します(≥ 1.0). デフォルトは1です.

dprm[8]: 修正方程式で用いる固有値近似値として, 反復毎に得られる近似固有値θではなく, 固定シフトτを反復開始初期に用います.
反復回数がdprm[8]までのときに, 固定シフトを用います.
dprm[2] = 0のとき, デフォルトはdprm[8] = 0です.
dprm[2] = 1のとき, デフォルトはdprm[8] = $m_{\max}$です. ("使用上の注意"e)参照)

dprm[9]: 修正方程式を解く解法種別を指定します.
dprm[9] = 0のとき, 修正方程式を用いずに**t** = **r**とします.
dprm[9] = 1のとき, GMRES法を使用します.
dprm[9] = 2のとき, BiCGstab(L)法を使用します.
dprm[9] = 11のとき, MINRES法を使用します.
デフォルトはMINRES法です. ("使用上の注意"g), h)参照)

dprm[10]: 修正方程式解法で使用するパラメタを指定します.

BiCGstab(L)法を用いる場合に, Lの値を指定します(≤ 10). デフォルト値は4です.

dprm[11]: 修正方程式を連立一次方程式反復解法を用いて解く場合の反復回数上限を指定します(≥ 1). デフォルトは30です.

dprm[12]: 修正方程式を連立一次方程式反復解法を用いて解く場合に, 収束判定閾値を決めるパラメタを指定します(> 0.0).
デフォルト値は0.7です. (“使用上の注意”f)参照).

dprm[13]: 修正方程式を連立一次方程式反復解法を用いて解く場合に, 収束判定閾値を決めるパラメタを指定します. (0.0 < dprm[13] ≤ 1.0).
減次処理毎にリセットされる外部反復に応じたカウンタをlとしたとき, 修正方程式求解の収束判定閾値はdprm[12] × dprm[13]^l に設定されます.
デフォルト値は0.7です. (“使用上の注意”f)参照).

dprm[14]: 修正方程式を連立一次方程式反復解法を用いて解く場合に, 前処理方式を指定します(≤ 1).
dprm[14] = 0のとき, 前処理を行いません.
dprm[14] = 1のとき, 対角要素スケーリングによる左側前処理を行います. (“使用上の注意”g)参照). デフォルトはdprm[14] = 0です.

dprm[15] ~ [31]: 機能拡張用のリザーブ域です.

deval	double deval[nsel]	出力	求まった固有値が deval[i-1] , i = 1, ... , nev に格納されます.
zevec	dcomplex zevec[nsel][kv]	出力	求まった固有ベクトルが zevec[i-1][j-1], i = 1, ... , nev, j = 1, ... , n に格納されます.
		入力	iflag[6] ≠ 0 かつ dprm[6] = 1.0 のとき, 反復開始初期ベクトルを zevec[0][i-1], i = 1, ... , n に指定します.
kv	int	入力	配列 zevec の整合寸法(≥ n).
dhis	double dhis[2][kh]	出力	dhis[0][i-1], i = 1, ... , *min*(kh, iter)に Jacobi-Davidson 法の残差ノルム収束履歴を出力します. dhis[1][i-1], i = 1, ... , *min*(kh, iter)に反復毎に解いた修正方程式の最終相対残差ノルムを出力します.
kh	int	入力	配列 dhis の整合寸法(≥ 0). kh = itmax であれば十分です. kh = 0 を指定すると, dhis への出力は抑止されます.
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.
1000	修正方程式の反復解法処理で breakdown が発生した.	それまでの近似解を用いて処理を続ける.
2000	直交化処理などで零ベクトルが発生した.	必要に応じて乱数ベクトルで空間拡大し処理を続ける.
3000	遅延減次処理における前反復情報回復などのリカバリー処理が動作した.	処理を継続する.
10000	反復回数上限までに固有値が nsel 個まで求まらなかった.	得られた nev 個までの固有値および固有ベクトルの結果は正常.

コード	意　味	処理内容
20000	射影により縮小された密行列固有値問題の求解処理に失敗した.	処理を打ち切る. nev > 0 であれば, nev 個までの固有値および固有ベクトルの結果は正常.
21000	固有値が一つも求まらずに反復回数の上限に達した.	処理を打ち切る. 配列 deval[0]および zevec[0][i-1], i = 1, … , n 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.
29000	内部エラーが発生した.	処理を打ち切る.
30000	次のいずれかであった. • n < 1 • itrgt < 0 • itrgt > 4 • nsel < 1 • nsel > n • itmax < 0 • kv < n • kh < 0	
30001～30032	iflag または dprm の値が正しくない.	
31000	nz, ncol, nfrnz の値が正しくない.	

$$A = \begin{bmatrix} 1 & 2+4i & 0 & 0 \\ 2-4i & 5 & 7-3i & 6+9i \\ 0 & 7+3i & 8 & 0 \\ 0 & 6-9i & 0 & 10 \end{bmatrix}$$

$$\mathtt{nfrnz} = \begin{bmatrix} 1 \\ 2 \\ 4 \\ 6 \\ 8 \end{bmatrix}, \quad \mathtt{za} = \begin{bmatrix} \frac{1}{2-4i} \\ \frac{5}{7+3i} \\ \frac{8}{6-9i} \\ 10 \end{bmatrix}, \quad \mathtt{ncol} = \begin{bmatrix} 1 \\ \frac{1}{2} \\ \frac{2}{3} \\ \frac{3}{2} \\ 4 \end{bmatrix}$$

図 c_dm_vjdhecr-1

3. 使用上の注意

a) Jacobi Davidson アルゴリズムについて

Jacobi-Davidson 法は確定的な手法ではなく, 一般に行列変形を経る密行列向けの解法ほど堅固な方法ではありません. Jacobi-Davidson 法で得られる結果は初期ベクトルの選択にも依存し, 優先的に選択する固有値の出力順も保証されません. また, 求める固有値・固有ベクトルの個数 nsel が, 行列次数 n に比べてごく少数の場合に適しています.

アルゴリズム内では種々の動作調整パラメタが使用されており, それらのパラメタを調整することで, 収束が改善することがあります. アルゴリズムにおける各パラメタ変数の説明については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VJDHECR の“手法概要を参照してください.

b) itrgt, dtrgt パラメタについて

本関数では, itrgt に設定する固有値選択方法に応じて, 使用する内部アルゴリズムを指定する dprm[2] のデフォルト値を自動で切り換えています. iflag[2] ≠ 0 として明に dprm[2]の設定をした場合にはその選択が優先されます. すなわちアルゴリズム適性に合う問題設定であれば, itrgt = 0, 2 で standard アルゴリズムを使用したり, itrgt = 1, 3, 4 で harmonic アルゴリズムを使用することもできます.

なお, 絶対値の小さい固有値を優先的に選択する itrgt = 2 でのデフォルトは harmonic アルゴリズムであるため, dtrgt パラメタの値が参照されることにご注意ください. 適切な値が不明な場合は dtrgt = 0.0 を指定してください.

c) 残差ノルム計算方法について

本関数では, 収束判定に用いる残差ノルムの値として近似固有値絶対値に対する相対残差ノルムを用いる計算方法をデフォルトとしています. しかし求める固有値が行列ノルムに比べてはるかに小さなゼロ近傍固有値の場合, 収束判定 $|\mathbf{Au} - \theta\mathbf{u}| / |\theta| <$ dprm[3]を満たすのが困難な場合があります. その場合には収束判定閾値 dprm[3]の値を調整するか dprm[4]で指定する残差ノルム計算方法を変更することで, 収束が得られる可能性があります.

d) 遅延減次処理方式について

本関数では解の計算精度をあげるために, 収束判定閾値よりも残差ノルムが小さくなった後も減次処理をしないで反復を続ける方法を採用しています. これを遅延減次処理と呼んでいます. この処理が有効な場合, 残差ノルムが収束とみなせる許容閾値 dprm[3]よりも小さくなった後, 残差ノルムが前反復と比較して dprm[5]で指定する比率以下に減少する間は反復を続けてから減次処理を行います. この反復継続中に残差ノルムが前の反復より悪化してしまった場合は, 前の反復の近似固有値および近似固有ベクトルを復元してから減次処理を行っています. dprm[5] = 0.0 と指定した場合, この遅延減次処理は行われません.

e) 修正方程式に用いる近似固有値について

反復開始初期段階では, 反復毎に得られる近似固有値 θ が, 求める固有値に遠い値である可能性があります. 探索副空間拡大の良否は修正方程式に用いる近似固有値に大きく影響されるため, 本関数では反復開始初期段階では θ ではなく, dtrgt に指定された固定シフト τ の値を修正方程式に用いる方法を採用しています. 何反復目までを反復開始初期段階とみなすかは, 動作調整用パラメタの dprm[8]で指定できます. スペクトル端の固有値を求めるのに適した standard アルゴリズムを用いる場合には, あらかじめ適当な固有値近似値 τ を設定するのが難しいであろうという配慮から, dprm[2] = 0 の場合のデフォルトは本方式を使用しない dprm[8] = 0 としています.

f) 修正方程式求解の収束判定閾値

本関数では修正方程式求解に, スパース行列を係数行列とする連立一次方程式の反復解法を用いています. 固有ベクトルを求める Jacobi-Davidson アルゴリズムのループを外側反復と呼び, 修正方程式の反復解法でのループを内側反復と呼びます. 外側反復が収束に近づいていない状況では, 内側反復で行われる修正方程式の求解の精度は高くなくても良いと考えられています. 内側反復処理の節約のため, 修正方程式求解の収束判定閾値は外側反復に応じて変化させることができます. 減次処理毎に 0 にリセットされる外部反復に応じたカウンタを l としたとき, 修正方程式求解の収束判定閾値は dprm[12] × dprm[13]^l に設定されます. なお, 内側反復回数の上限は dprm[11]で指定します.

g) 修正方程式反復解法の前処理について

修正方程式に対する連立一次方程式反復解法において, 前処理が有効な場合があります. 前処理種別は dprm[14]で指定します. このとき修正方程式の前処理行列として, 左右の射影操作を含まない部分の $\mathbf{M} \fallingdotseq (\mathbf{A} - \tau\mathbf{I})$を設定するためのシフト値 τ として dtrgt への指定値を使用します. τ が求める固有値と離れて

いたり, 前処理行列が修正方程式の係数行列をうまく近似できていないと収束悪化になる場合もありますので注意が必要です. また, 本関数で対応しているのは左側前処理であり修正方程式の係数行列がエルミート行列でなくなるため, その解法としては非対称行列向けのアルゴリズムを dprm[9]に指定する必要があります.

h) 使用メモリについて

本関数では, 内部で使用する作業域を動的確保しています. このため, メモリが不足すると異常終了する可能性があります. そのサイズはパラメタの設定方法によって異なりますが, standard アルゴリズムの場合は $n \times (2 \times m_{\max} + 2 \times \mathtt{nsel}) \times 16$byte 以上, harmonic アルゴリズムでは $n \times (3 \times m_{\max} + 2 \times \mathtt{nsel}) \times 16$byte 以上が基本アルゴリズム部分で必要であり, 修正方程式解法として GMRES 法を用いる場合にはそれに加えて $n \times \mathtt{dprm[11]} \times 16$byte が大規模問題での主要な作業域サイズになります.

4. 使用例

1 行あたりの非零要素数が 20 個程度の 10000 次元のエルミート行列の絶対値最大固有値を 10 個と, それに対応する固有ベクトルを求めます.

(並列実行を行うスレッド数は環境変数(OMP_NUM_THREADS)などで指定できます. 例えば, 4 プロセッサーのシステムで 4 つのスレッドで並列実行するときは, OMP_NUM_THREADS を 4 に設定して実行します.)

```
/* **EXAMPLE** */
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <malloc.h>
#include "cssl.h"

#define     NMAX 10000
#define     NZC 20
#define     NNZMAX NMAX*NZC
#define     LDK 10

int      mkspmat(int, int, dcomplex*, int*, int*);
dcomplex comp_add(dcomplex, dcomplex);
dcomplex comp_sub(dcomplex, dcomplex);
dcomplex comp_mult(dcomplex, dcomplex);
dcomplex d_c_mult(dcomplex, double);

int MAIN__() {

  static dcomplex  zh[NNZMAX], zevec[LDK][NMAX];
  dcomplex  rvec[NMAX], zw[NMAX], zh_w;
  double  dtrgt, deval[LDK], derr, dprm[32], dhis[2][NMAX];
  int  nz, ncol[NNZMAX], nfrnz[NMAX+1], n, itrgt;
  int  iflag[32], nsel, nev, itmax, iter, ldx, ldh, icon;
  int  i, j, k, ncolj;

  n = NMAX;
  mkspmat(n, NZC, zh, ncol, nfrnz);
  nz = nfrnz[n] - 1;

  itmax = 500;
  nsel = 10;
  for (i = 0; i < 32; i++) {
    iflag[i] = 0;
  }
  ldx = NMAX;
  ldh = NMAX;
  dtrgt = 0.0;
  itrgt = 1;

  c_dm_vjdhecr(zh, nz, ncol, nfrnz, n, itrgt, dtrgt, nsel,
              &nev, itmax, &iter, iflag, dprm,
              deval, (dcomplex *)zevec, ldx, (double *)dhis, ldh, &icon);

  printf(" C_DM_VJDHECR ICON= %d\n", icon);
```

```
  printf(" ITER= %d\n", iter);
  for (k = 0; k < nev; k++) {
#pragma omp parallel private(i, j, ncolj, zw, zh_w)
{
    for (i = 0; i < n; i++) {
      zw[i].re = 0.0;
      zw[i].im = 0.0;
    }
#pragma omp for
    for (i = 0; i < n; i++) {
      rvec[i].re = 0.0;
      rvec[i].im = 0.0;
      for (j = nfrnz[i]-1; j < nfrnz[i+1]-1; j++) {
        ncolj = ncol[j] - 1;
        rvec[i] = comp_add(rvec[i], comp_mult(zh[j], zevec[k][ncolj]));
        if (i != ncolj) {
          zh_w = zh[j];
          zh_w.im = -zh_w.im;
          zw[ncolj] = comp_add(zw[ncolj], comp_mult(zh_w, zevec[k][i]));
        }
      }
    }
#pragma omp critical
    for (i = 0; i < n; i++) {
      rvec[i] = comp_add(rvec[i], zw[i]);
    }
}
    derr = 0.0;
    for (i = 0; i < n; i++) {
      rvec[i] = comp_sub(rvec[i], d_c_mult(zevec[k][i], deval[k]));
      derr = derr + (rvec[i].re * rvec[i].re) + (rvec[i].im * rvec[i].im);
    }
    derr = sqrt(derr);
    printf(" EIGEN VALUE %d =%18.14lf\n", k+1, deval[k]);
    printf(" ERROR= %22.16le\n", derr/fabs(deval[k]));
  }
  return(0);
}

int mkspmat(int n, int nzc, dcomplex *zh, int *ncol, int *nfrnz) {
#define LDW 1350
  int  i, ic, ict, j, k, iseed, icon, nnz;
  double  *dwork, rndwork[LDW];

  dwork = (double *)malloc(nzc * sizeof(double));

  iseed = 1;
  nnz = 0;
  for (i = 1; i <= n; i++) {
    nfrnz[i-1] = nnz + 1;
label_10:   c_dvrau4(&iseed, dwork, nzc, rndwork, LDW, &icon);
    ic = 0;
    for (j = 1; j <= nzc; j++) {
      ict = n * fabs(dwork[j-1]) + 1;
      if (ict <= i) {
        for (k = 1; k <= ic; k++) {
          if (ict == ncol[nnz - k]) {
            nnz = nnz - ic;
            goto label_10;
          }
        }
        ic++;
        ncol[nnz] = ict;
        nnz++;
      }
    }
  }
  nfrnz[n] = nnz + 1;
  iseed = 1;
  c_dvran4(0.0, 1.0, &iseed, (double *)zh, 2 * nnz, rndwork, LDW,
      &icon);
  for (i = 0; i < n; i++) {
    for (j = nfrnz[i]-1; j < nfrnz[i+1]-1; j++) {
      if (i == ncol[j]-1) {
        zh[j].re = zh[j].re + zh[j].im;
        zh[j].im = 0.0;
      }
```

```
    }
  }
  free(dwork);
  return(0);
}

dcomplex comp_add(dcomplex so1, dcomplex so2) {

  dcomplex obj;

  obj.re = so1.re + so2.re;
  obj.im = so1.im + so2.im;
  return obj;
}

dcomplex comp_sub(dcomplex so1, dcomplex so2) {

  dcomplex obj;

  obj.re = so1.re - so2.re;
  obj.im = so1.im - so2.im;
  return obj;
}

dcomplex comp_mult(dcomplex so1, dcomplex so2) {

  dcomplex obj;

  obj.re = so1.re * so2.re - so1.im * so2.im;
  obj.im = so1.re * so2.im + so1.im * so2.re;
  return obj;
}

dcomplex d_c_mult(dcomplex so1, double so2) {

  dcomplex obj;

  obj.re = so1.re * so2;
  obj.im = so1.im * so2;
  return obj;
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VJDHECR の項目及び[7]を参照してください.

c_dm_vjdnhcr

複素スパース行列の固有値問題 (Jacobi-Davidson 法, 圧縮行格納法)
`ierr = c_dm_vjdnhcr(za, nz, ncol, nfrnz, n, itrgt, ztrgt, nsel, &nev, itmax, &iter, iflag, dprm, zeval, zevec, kv, dhis, kh, &icon);`

1. 機能

$n \times n$ の複素スパース行列の指定されたいくつかの固有値および対応する固有ベクトルを Jacobi-Davidson 法で求めます.

$$\mathbf{Ax} = \lambda \mathbf{x}$$

複素行列 **A** は, 圧縮行格納法で格納します. ベクトル **x** は n 次元ベクトルです.

2. 引数

呼出し形式:

```
ierr = c_dm_vjdnhcr (za, nz, ncol, nfrnz, n, itrgt, ztrgt, nsel, &nev, itmax,
            &iter, iflag, dprm, zeval, (dcomplex*)zevec, kv, (double*)dhis,
            kh, &icon);
```

引数の説明:

za	dcomplex za[nz]	入力	行列 **A** の非零要素を格納します. 圧縮行格納法については, 図 c_dm_vjdnhcr-1 を参照してください.
nz	int	入力	行列 **A** の非零要素の総数.
ncol	int ncol[nz]	入力	圧縮行格納法で使用される列指標で, za に格納される要素が何番目の列ベクトルに属するかを示します.
nfrnz	int nfrnz[n+1]	入力	行列 **A** の各行の非零要素を行方向に圧縮して順次配列 za に格納するとき, 対応する行の最初の非零要素が格納される位置を表します. nfrnz[n] = nz + 1 として指定します.
n	int	入力	行列 **A** の次数 n.
itrgt	int	入力	求める固有値の選択方法を指定します($0 \le$ itrgt ≤ 6). itrgt = 0: ztrgt に指定したターゲット値の近傍の固有値を優先的に選択します. itrgt = 1: 絶対値の大きい固有値を優先的に選択します. itrgt = 2: 絶対値の小さい固有値を優先的に選択します. itrgt = 3: 実数部が大きい固有値を優先的に選択します. itrgt = 4: 実数部が小さい固有値を優先的に選択します. itrgt = 5: 虚数部が大きい固有値を優先的に選択します. itrgt = 6: 虚数部が小さい固有値を優先的に選択します. ("使用上の注意"a),b)参照)

ztrgt	dcomplex	入力	itrgt = 0 のとき, 固有値を選択するターゲット値を複素変数で指定します. また, itrgt ≠ 0 であっても以下の場合には, 求める固有値の近傍値を ztrgt に指定することで収束が改善することがあります. 1) dprm[2] = 1 で harmonic アルゴリズムを選択しているとき, テスト副空間<**W**> = <(**A** − τ **I**)**V**> を設定するためのシフト値 τ として ztrgt を使用します. ("使用上の注意"b)参照) 2) dprm[8] ≥ 1 のとき, 修正方程式で用いる近似固有値として, 反復開始初期に ztrgt への指定値を使用します. ("使用上の注意"e)参照) 3) dprm[14] ≥ 1 のとき, 修正方程式の前処理行列を設定するためのシフト値 τ として ztrgt への指定値を利用します. ("使用上の注意"g)参照) それ以外の場合には, ztrgt の値は参照されません.
nsel	int	入力	求める固有値個数を指定します(1 ≤ nsel ≤ n). ("使用上の注意"a)参照)
nev	int	出力	収束した固有値個数.
itmax	int	入力	反復回数の上限を指定します(≥ 0).
iter	int	出力	Jacobi-Davidson 法での実際の反復回数.
iflag	int iflag[32]	入力	dprm に動作調整パラメタを明に指定するかどうかを示します.. iflag[i] ≠ 0 のとき, dprm[i]に指定したパラメタを使用します ($i \le 14$). iflag[i] = 0 のとき, dprm[i]は参照せずデフォルト値を使用します. iflag[15] ~ [31] は機能拡張用のリザーブ域のため, 0 を指定しておきます.
dprm	double dprm[32]	入力	iflagで指定された動作調整用パラメタについて, 値を指定します. アルゴリズムにおける各パラメタ変数の定義については"FUJITSU SSL II スレッド並列機能使用手引書"のDM_VJDNHCRの"手法概要"を参照してください. iflag[0] ~ [31] が全て0であればdprmは参照されず, デフォルト値が用いられます. デフォルト値のままでは収束しなかった場合に, パラメタを変えて試みることを奨めます. dprm[0]: リスタート時に縮小する副空間次元$m_{\min}$を指定します. ($1 \le m_{\min} < n$). デフォルト値は$m_{\min} = 50$です. dprm[1]: 副空間次元の最大次元$m_{\max}$を指定します ($m_{\min} < m_{\max} \le n$). デフォルト値は$m_{\max} = m_{\min} + 30$です. ("使用上の注意"h)参照) dprm[2]: 射影するテスト副空間の設定方法による, アルゴリズムの種別を指定します. dprm[2] = 0のとき, 固有値選択ターゲットがスペクトル端の場合に適したstandardアルゴリズムを使用します. dprm[2] = 1のとき, 固有値選択ターゲットがスペクトル内部の場合に適したharmonicアルゴリズムを使用します. デフォルトでは, itrgt = 0および2のときに

harmonicアルゴリズムを使用し, それ以外のときにstandardアルゴリズムを使用します.

dprm[3]: 収束判定許容閾値を指定します. デフォルトは10^{-6}です.
("使用上の注意"d)参照)

dprm[4]: 反復中に得られた近似固有値θおよび近似固有ベクトル$\boldsymbol{u}$に対し, 残差ノルムを計算する方法を指定します.
dprm[4] = 0のとき, 近似固有値絶対値に対する相対残差ノルム$|\mathbf{Au}-\theta\mathbf{u}| / |\theta|$を使用します.
dprm[4] = 1のとき, 行列**A**の1-ノルムに対する相対残差ノルム$|\mathbf{Au}-\theta\mathbf{u}| / |\mathbf{A}|_1$を使用します.
dprm[4] = 2 のとき, 行列**A**のFrobeniusノルムに対する相対残差 $|\mathbf{Au}-\theta\mathbf{u}| / |\mathbf{A}|_{\mathrm{Fro}}$ を使用します.
dprm[4] = 3のとき, 行列**A**の∞-ノルムに対する相対残差ノルム $|\mathbf{Au}-\theta\mathbf{u}| / |\mathbf{A}|_\infty$ を使用します.
dprm[4] = 4のとき, 絶対残差ノルム $|\mathbf{Au}-\theta\mathbf{u}|$ を使用します.
デフォルトは, dprm[4] = 0です.
("使用上の注意"c)参照)

dprm[5]: 遅延減次処理における閾値を指定します(≤ 1.0).
デフォルトはdprm[5] = 0.9です.
("使用上の注意"d)参照)

dprm[6]: 反復開始初期ベクトルをzevec[0][i-1], i = 1, ... , nに指定するかどうかを指定します.
dprm[6] = 0.0のとき, ルーチン内部で生成した乱数ベクトルを反復開始初期ベクトルとして使用します.
dprm[6] = 1.0のとき, zevec[0][i-1], i = 1, ... , nに反復開始初期ベクトルを指定してください.
デフォルトは乱数ベクトルを使用します.

dprm[7]: 乱数列生成のための種(*seed*)の値を指定します(≥ 1.0). デフォルトは1です.

dprm[8]: 修正方程式で用いる固有値近似値として, 反復毎に得られる近似固有値θではなく, 固定シフトτを反復開始初期に用います.
反復回数がdprm[8]までのときに, 固定シフトを用います.
dprm[2] = 0のとき, デフォルトはdprm[8] = 0です.
dprm[2] = 1のとき, デフォルトはdprm[8] = $m_{\max}$です. ("使用上の注意"e)参照)

dprm[9]: 修正方程式を解く解法種別を指定します.
dprm[9] = 0のとき, 修正方程式を用いずに$\boldsymbol{t} = \boldsymbol{r}$とします.
dprm[9] = 1のとき, GMRES法を使用します.
dprm[9] = 2のとき, BiCGstab(L)法を使用します.
dprm[9] = 11のとき, MINRES法を使用します.
デフォルトはMINRES法です. ("使用上の注意"g), h)参照)

dprm[10]: 修正方程式解法で使用するパラメタを指定します.

BiCGstab(L)法を用いる場合に, Lの値を指定します(≤ 10). デフォルト値は4です.

dprm[11]: 修正方程式を連立一次方程式反復解法を用いて解く場合の反復回数上限を指定します(≥ 1). デフォルトは30です.

dprm[12]: 修正方程式を連立一次方程式反復解法を用いて解く場合に, 収束判定閾値を決めるパラメタを指定します(> 0.0).
デフォルト値は0.7です. (“使用上の注意”f)参照).

dprm[13]: 修正方程式を連立一次方程式反復解法を用いて解く場合に, 収束判定閾値を決めるパラメタを指定します. (0.0 < dprm[13] ≤ 1.0).
減次処理毎にリセットされる外部反復に応じたカウンタをlとしたとき, 修正方程式求解の収束判定閾値はdprm[12] × dprm[13]^l に設定されます.
デフォルト値は0.7です. (“使用上の注意”f)参照).

dprm[14]: 修正方程式を連立一次方程式反復解法を用いて解く場合に, 前処理方式を指定します(≤ 1).
dprm[14] = 0のとき, 前処理を行いません.
dprm[14] = 1のとき, 対角要素スケーリングによる左側前処理を行います. (“使用上の注意”g)参照). デフォルトはdprm[14] = 0です.

dprm[15] ~ [31]: 機能拡張用のリザーブ域です.

zeval	dcomplex zeval[nsel]	出力	求まった固有値が zeval[i-1] , i = 1, ... , nev に格納されます.
zevec	dcomplex zevec[nsel][kv]	出力	求まった固有ベクトルが zevec[i-1][j-1], i = 1, ... , nev, j = 1, ... , n に格納されます.
		入力	iflag[6] ≠ 0 かつ dprm[6] = 1.0 のとき, 反復開始初期ベクトルを zevec[0][i-1], i = 1, ... , n に指定します.
kv	int	入力	配列 zevec の整合寸法(≥ n).
dhis	double dhis[2][kh]	出力	dhis[0][i-1], i = 1, ... , *min*(kh, iter)に Jacobi-Davidson 法の残差ノルム収束履歴を出力します. dhis[1][i-1], i = 1, ... , *min*(kh, iter)に反復毎に解いた修正方程式の最終相対残差ノルムを出力します.
kh	int	入力	配列 dhis の整合寸法(≥ 0). kh = itmax であれば十分です. kh = 0 を指定すると, dhis への出力は抑止されます.
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.
1000	修正方程式の反復解法処理で breakdown が発生した.	それまでの近似解を用いて処理を続ける.
2000	直交化処理などで零ベクトルが発生した.	必要に応じて乱数ベクトルで空間拡大し処理を続ける.
3000	遅延減次処理における前反復情報回復などのリカバリー処理が動作した.	処理を継続する.
10000	反復回数上限までに固有値が nsel 個まで求まらなかった.	得られた nev 個までの固有値および固有ベクトルの結果は正常.

コード	意　味	処理内容
20000	射影により縮小された密行列固有値問題の求解処理に失敗した.	処理を打ち切る. nev > 0 であれば, nev 個までの固有値および固有ベクトルの結果は正常.
21000	固有値が一つも求まらずに反復回数の上限に達した.	処理を打ち切る. 配列 zeval[0]および zevec[0][i-1], i = 1, … , n 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.
29000	内部エラーが発生した.	処理を打ち切る.
30000	次のいずれかであった. • n < 1 • itrgt < 0 • itrgt > 6 • nsel < 1 • nsel > n • itmax < 0 • kv < n • kh < 0	
30001～30032	iflag または dprm の値が正しくない.	
31000	nz, ncol, nfrnz の値が正しくない.	

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 & 0 \\ 4 & 5 & 0 & 6 \\ 0 & 7 & 8 & 9 \\ 0 & 0 & 10 & 11 \end{bmatrix}$$

$$\mathtt{nfrnz} = \begin{bmatrix} 1 \\ 4 \\ 7 \\ 10 \\ 12 \end{bmatrix}, \quad \mathtt{za} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \end{bmatrix}, \quad \mathtt{ncol} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 1 \\ 2 \\ 4 \\ 2 \\ 3 \\ 4 \\ 3 \\ 4 \end{bmatrix}$$

図 c_dm_vjdnhcr-1

3. 使用上の注意

a) Jacobi Davidson アルゴリズムについて

Jacobi-Davidson 法は確定的な手法ではなく, 一般に行列変形を経る密行列向けの解法ほど堅固な方法ではありません. Jacobi-Davidson 法で得られる結果は初期ベクトルの選択にも依存し, 優先的に選択する固有値の出力順も保証されません. また, 求める固有値・固有ベクトルの個数 nsel が, 行列次数 n に比べてごく少数の場合に適しています.

アルゴリズム内では種々の動作調整パラメタが使用されており, それらのパラメタを調整することで, 収束が改善することがあります. アルゴリズムにおける各パラメタ変数の説明については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VJDNHCR の“手法概要”を参照してください.

b) itrgt, ztrgt パラメタについて

本関数では, itrgt に設定する固有値選択方法に応じて, 使用する内部アルゴリズムを指定する dprm[2] のデフォルト値を自動で切り換えています. iflag[2] ≠ 0 として明に dprm[2]の設定をした場合にはその選択が優先されます. すなわちアルゴリズム適性に合う問題設定であれば, itrgt = 0 , 2 で standard アルゴリズムを使用したり, itrgt = 1, 3, 4, 5, 6 で harmonic アルゴリズムを使用することもできます.

なお, 絶対値の小さい固有値を優先的に選択する itrgt = 2 でのデフォルトは harmonic アルゴリズムであるため, ztrgt パラメタの値が参照されることにご注意ください. 適切な値が不明な場合は ztrgt = (0.0, 0.0)を指定してください.

c) 残差ノルム計算方法について

本関数では, 収束判定に用いる残差ノルムの値として近似固有値絶対値に対する相対残差ノルムを用いる計算方法をデフォルトとしています. しかし求める固有値が行列ノルムに比べてはるかに小さなゼロ近傍固有値の場合, 収束判定 $|\mathbf{Au} - \theta\mathbf{u}| / |\theta| <$ dprm[3]を満たすのが困難な場合があります. その場合には収束判定閾値 dprm[3]の値を調整するか dprm[4]で指定する残差ノルム計算方法を変更することで, 収束が得られる可能性があります.

d) 遅延減次処理方式について

本関数では解の計算精度をあげるために, 収束判定閾値よりも残差ノルムが小さくなった後も減次処理をしないで反復を続ける方法を採用しています. これを遅延減次処理と呼んでいます. この処理が有効な場合, 残差ノルムが収束とみなせる許容閾値 dprm[3]よりも小さくなった後, 残差ノルムが前反復と比較して dprm[5]で指定する比率以下に減少する間は反復を続けてから減次処理を行います. この反復継続中に残差ノルムが前の反復より悪化してしまった場合は, 前の反復の近似固有値および近似固有ベクトルを復元してから減次処理を行っています. dprm[5] = 0.0 と指定した場合, この遅延減次処理は行われません.

e) 修正方程式に用いる近似固有値について

反復開始初期段階では, 反復毎に得られる近似固有値 θ が, 求める固有値に遠い値である可能性があります. 探索副空間拡大の良否は修正方程式に用いる近似固有値に大きく影響されるため, 本関数では反復開始初期段階では θ ではなく, ztrgt に指定された固定シフト τ の値を修正方程式に用いる方法を採用しています. 何反復目までを反復開始初期段階とみなすかは, 動作調整用パラメタの dprm[8]で指定できます. スペクトル端の固有値を求めるのに適した standard アルゴリズムを用いる場合には, あらかじめ適当な固有値近似値 τ を設定するのが難しいであろうという配慮から, dprm[2] = 0 の場合のデフォルトは本方式を使用しない dprm[8] = 0 としています.

f) 修正方程式求解の収束判定閾値

本関数では修正方程式求解に, スパース行列を係数行列とする連立一次方程式の反復解法を用いています. 固有ベクトルを求める Jacobi-Davidson アルゴリズムのループを外側反復と呼び, 修正方程式の反復解法でのループを内側反復と呼びます. 外側反復が収束に近づいていない状況では, 内側反復で行われる修正方程式の求解の精度は高くなくても良いと考えられています. 内側反復処理の節約のため, 修正方程式求解の収束判定閾値は外側反復に応じて変化させることができます. 減次処理毎に 0 にリセットされる外部反復に応じ

たカウンタを l としたとき, 修正方程式求解の収束判定閾値は dprm[12] × dprm[13]^l に設定されます. なお, 内側反復回数の上限は dprm[11]で指定します.

g) 修正方程式反復解法の前処理について

修正方程式に対する連立一次方程式反復解法において, 前処理が有効な場合があります. 前処理種別は dprm[14]で指定します. このとき修正方程式の前処理行列として, 左右の射影操作を含まない部分の $\mathbf{M} \fallingdotseq (\mathbf{A} - \tau\mathbf{I})$を設定するためのシフト値 τ として ztrgt への指定値を使用します. τ が求める固有値と離れていたり, 前処理行列が修正方程式の係数行列をうまく近似できていないと収束悪化になる場合もありますので注意が必要です.

h) 使用メモリについて

本関数では, 内部で使用する作業域を動的確保しています. このため, メモリが不足すると異常終了する可能性があります. そのサイズはパラメタの設定方法によって異なりますが, standard アルゴリズムの場合は $n \times (2 \times m_{\max} + 2 \times \mathtt{nsel} + 5) \times 16$byte 以上, harmonic アルゴリズムでは $n \times (3 \times m_{\max} + 2 \times \mathtt{nsel} \times 16$byte 以上が基本アルゴリズム部分で必要であり, 修正方程式解法として GMRES 法を用いる場合にはそれに加えて $n \times$ dprm[11] × 16byte が大規模問題での主要な作業域サイズになります.

4. 使用例

1 行あたりの非零要素数が 20 個の 10000 次元のランダム行列の絶対値最大固有値を 10 個と, それに対応する固有ベクトルを求めます.

(並列実行を行うスレッド数は環境変数(OMP_NUM_THREADS)などで指定できます. 例えば, 4 プロセッサーのシステムで 4 つのスレッドで並列実行するときは, OMP_NUM_THREADS を 4 に設定して実行します.)

```
/* **EXAMPLE** */
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <malloc.h>
#include <omp.h>
#include "cssl.h"

#define NMAX 10000
#define NZC 20
#define NNZMAX NMAX*NZC
#define LDK 10

int      mkspmat(int, int, dcomplex*, int*, int*);
dcomplex comp_add(dcomplex, dcomplex);
dcomplex comp_sub(dcomplex, dcomplex);
dcomplex comp_mult(dcomplex, dcomplex);
double   cdabs(dcomplex);


int MAIN__() {

  static dcomplex za[NNZMAX], ztrgt, zeval[LDK], zevec[LDK][NMAX];
  dcomplex rvec[NMAX];
  double derr, dprm[32], dhis[2][NMAX];
  int nz, ncol[NNZMAX], nfrnz[NMAX+1], n, itrgt, iflag[32];
  int nsel, nev, itmax, iter, i, j, k, icon, ldx, ldh;

  n = NMAX;
  mkspmat(n, NZC, za, ncol, nfrnz);
  nz = nfrnz[n] - 1;
  itmax = 500;
  nsel = 10;
  for (i=0; i<32; i++) {
    iflag[i] = 0;
  }
  ldx = NMAX;
  ldh = NMAX;
  ztrgt.re = 0.0;
```

```
  ztrgt.im = 0.0;
  itrgt = 1;
  c_dm_vjdnhcr(za, nz, ncol, nfrnz, n, itrgt, ztrgt, nsel, &nev,
               itmax, &iter, iflag, dprm, zeval, (dcomplex*)zevec, ldx,
               (double*)dhis, ldh, &icon);

  printf(" C_DM_VJDNHCR ICON= %d\n", icon);
  printf(" ITER= %d\n", iter);
  for (k=0; k<nev; k++) {
    for (i=0; i<n; i++) {
      rvec[i].re = 0.0;
      rvec[i].im = 0.0;
    }
#pragma omp parallel for private(j)
    for (i=0; i<n; i++) {
      for (j=nfrnz[i]-1; j<nfrnz[i+1]-1; j++) {
        rvec[i] = comp_add(rvec[i], comp_mult(za[j], zevec[k][ncol[j]-1]));
      }
      rvec[i] = comp_sub(rvec[i], comp_mult(zeval[k], zevec[k][i]));
    }
    derr = 0.0;
    for (i=0; i<n; i++) {
      derr = derr + (rvec[i].re * rvec[i].re) + (rvec[i].im * rvec[i].im);
    }
    derr = sqrt(derr);
    printf(" EIGEN VALUE %d = (%.15lf,%.15lf)\n", k+1, zeval[k].re, zeval[k].im);
    printf(" ERROR= %3.15le\n", derr/cdabs(zeval[k]));
  }
  return(0);
}

int mkspmat(int n, int nzc, dcomplex *za, int *ncol, int *nfrnz) {
#define LDW 1350

  int  i, ic, ict, j, k, iseed, icon;
  double  *dwork, rndwork[LDW];

  dwork = (double *)malloc(nzc * sizeof(double));

  iseed = 1;
  c_dvran4(0.0, 1.0, &iseed, (double*)za, (2*n*nzc), rndwork, LDW, &icon);
  iseed = 1;
  for (i=0; i<n; i++) {
    nfrnz[i] = i * nzc + 1;
LABEL_10:   c_dvrau4(&iseed, dwork, nzc, rndwork, LDW, &icon);
    ic = i * nzc;
    for (j=0; j<nzc; j++) {
      ict = n * fabs(dwork[j]) + 1;
      for (k=0; (k<=j) && (j!=0); k++) {
        if (ict == ncol[ic-k]) goto LABEL_10;
      }
      ic = ic + 1;
      ncol[ic-1] = ict;
    }
  }
  nfrnz[n] = ic + 1;
  free(dwork);
  return 0;
}

dcomplex comp_add(dcomplex so1, dcomplex so2) {

  dcomplex obj;

  obj.re = so1.re + so2.re;
  obj.im = so1.im + so2.im;
  return obj;
}

dcomplex comp_sub(dcomplex so1, dcomplex so2) {

  dcomplex obj;

  obj.re = so1.re - so2.re;
  obj.im = so1.im - so2.im;
  return obj;
}
```

```
dcomplex comp_mult(dcomplex so1, dcomplex so2) {

  dcomplex obj;

  obj.re = so1.re * so2.re - so1.im * so2.im;
  obj.im = so1.re * so2.im + so1.im * so2.re;
  return obj;
}

double cdabs(dcomplex so) {
  double obj;

  obj = sqrt(so.re * so.re + so.im * so.im);
  return obj;
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VJDNHCR の項目及び[7]を参照してください.

c_dm_vlax

実行列の連立 1 次方程式 (ブロック化された LU 分解法)
`ierr = c_dm_vlax(a, k, n, b, epsz, isw, &is, ip, &icon);`

1. 機能

実係数連立 1 次方程式をブロック化された LU 分解法 (ガウスの消去法)で解きます.

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

ただし, **A** は $n \times n$ の正則な実行列, **b** は n 次元の実定数ベクトル, **x** は n 次元の 解ベクトルです. ($n \geq 1$)

2. 引数

呼出し形式:

```
ierr = c_dm_vlax((double*)a, k, n, b, epsz, isw, &is, ip, &icon);
```

引数の説明:

a	double a[n][k]	入力	係数行列 **A**. 演算後, 内容は保存されません.
k	int	入力	配列 a の整合寸法 (≥ 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ます. ("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, 次のとおり指定します. isw = 1 のとき 1 組目の方程式を解きます. isw = 2 のとき 2 組目以降の方程式を解きます. 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で使います. ("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ます.
ip	int ip[n]	出力	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.
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. • k < n • n < 1 • epsz < 0 • isw ≠ 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ます. 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,

$$\left|a_{kk}^{k}\right| \le \max\left|a_{ij}\right| \ \mathrm{epsz}$$

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ります. epsz の標準値は丸め誤差の単位をμとしたとき, epsz = 16μです.

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いいのですが, その結果は保証されません.

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ります. なお, この場合 is の値は,isw = 1 のときの値が保証されます.

4. 使用例

1000×1000 の係数実行列を持つ連立 1 次方程式を解きます.

```
#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 NMAX      (1000)
#define LDA       (NMAX+1)

MAIN__()
{
  int    n, is, isw, i, j, icon, ierr;
  int    ip[NMAX];
  double a[NMAX][LDA], b[NMAX];
  double epsz, s, det;

  n    = NMAX;
  epsz = 0.0;
  isw  = 1;

#pragma omp parallel for shared(a,n) private(i,j)
  for(i=0; i<n; i++)
    for(j=0; j<n; j++) a[i][j] = min(i,j)+1;

#pragma omp parallel for shared(b,n) private(i)
  for(i=0; i<n; i++) b[i] = (i+1)*(i+2)/2+(i+1)*(n-i-1);

  ierr = c_dm_vlax((double*)a, LDA, n, b, epsz, isw, &is, ip, &icon);

  if (icon != 0) {
    printf("ERROR: c_dm_vlax failed with icon = %d\n", icon);
    exit(1);
```

```
  }

  s = 1.0;
#pragma omp parallel for shared(a,n) private(i) reduction(*:s)
  for(i=0; i<n; i++) s *= a[i][i];

  printf("solution vector:\n");
  for(i=0; i<10; i++) printf("    b[%d] = %e\n", i, b[i]);

  det = is*s;
  printf("\ndeterminant of the matrix = %e\n", det);
  return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VLAX の項目を参照してください.

c_dm_vlbx

実バンド行列の連立 1 次方程式 (ガウスの消去法)
`ierr = c_dm_vlbx(a, k, n, nh1, nh2, b, epsz, isw, &is, ip, &icon);`

1. 機能

バンド行列の連立 1 次方程式をガウスの消去法により解きます.

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

ただし, **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, **b** は n 次元の実定数ベクトル, **x** は n 次元の解ベクトルです. $n > h_1 \geq 0, n > h_2 \geq 0$.

2. 引数

呼出し形式:

```
ierr = c_dm_vlbx((double*)a, k, n, nh1, nh2, b, epsz, isw, &is, ip, &icon);
```

引数の説明:

a	double a[n][k]	入力	バンド係数行列 **A** を格納します. 詳細については, 図 c_dm_vlbx-1.を参照してください.
		出力	LU 分解された行列 ***L*** および ***U*** が格納されます. 詳細については, 図 c_dm_vlbx-2.を参照してください.
k	int	入力	配列 a の整合寸法 ($\geq 2 \times$ nh1 + nh2 + 1).
n	int	入力	行列 **A** の次数 n.
nh1	int	入力	下バンド巾の大きさ h_1.
nh2	int	入力	上バンド巾の大きさ h_2.
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	ピボットの零判定値 (≥ 0.0). 0.0 のときは, 標準値が設定されます. ("使用上の注意" a)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, 次のとおり指定してください. 1 1 組目の方程式を解きます. 2 2 組目以降の方程式を解きます. 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, それ以外のパラメタはそのまま使用してください.
is	int	出力	行ベクトルの入換えの回数を示します. ("使用上の注意" c)参照) 1 偶数回の入換え. –1 奇数回の入換え.
ip	int ip[n]	出力	行の入れ換えの履歴情報を格納するトランスポジションベクトルが格納されます. ("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.

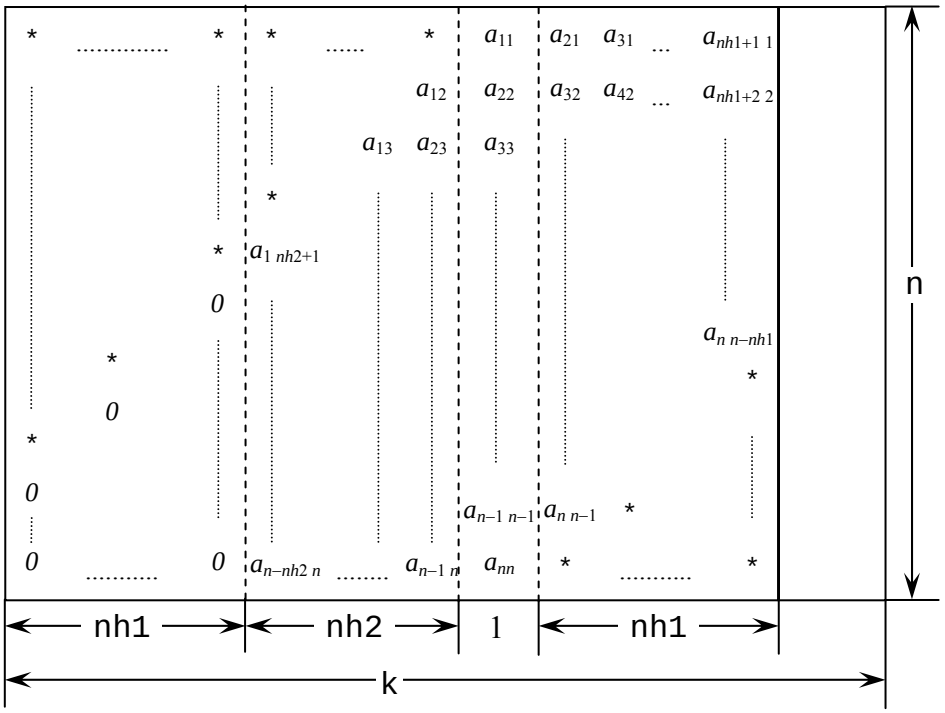


図 c_dm_vlbx-1. 配列 a に行列 **A** を格納する方法

バンド行列 **A** の対角要素 a_{ii}, $i = 1, \ldots, n$ が a[i–1][h_1+h_2]に格納されるように, 行列 **A** の列ベクトルを配列 a に連続に格納します.

上バンド行列部分, $a_{j-i,j}$, $i = 1, \ldots, h_2, j = 1, \ldots, n, j - i \geq 1$ を, a[i][j], i = 0, ... , n–1, j = h_1, ... , h_1+h_2–1 に格納します. 下バンド行列部分, $a_{j+i,j}$, $i = 1, \ldots, h_1, j = 1, \ldots, n, j + i \leq n$ を, a[i][j], i = 0, ... , n–1, j = h_1+h_2+1, ... , 2×h_1+h_2 に格納します.

配列 a[i][j], i = 0, ... , n – 1, j = 0, ... , h_1–1, で, バンドの外側の行列 **A** の要素はゼロを設定してください.

*は不定値を示します.

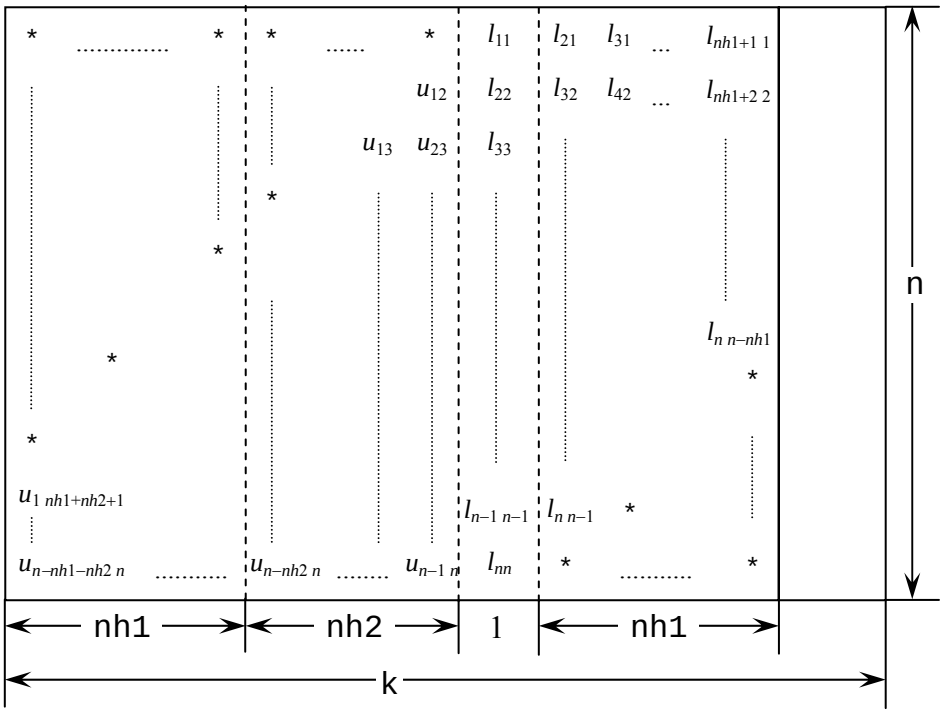


図 c_dm_vlbx-2. LU 分解された行列 **L** および **U** の配列 a に格納される方法

LU 分解された単位上バンド行列の対角要素を除いた部分, $u_{j-i+1,\,j}$, $i = 1, \ldots, h_1+h_2$, $j = 1, \ldots, n$, $j - i + 1 \geq 1$ が a[i][j], i = 0, ... , n–1, j = 0 , ... , h_1+h_2,に格納されます.

下バンド行列, $l_{j+i,\,j}$, $i = 0, \ldots, h_2$, $j = 1, \ldots, n$, $j + i \leq n$ が a[i][j], i = 0, ... , n–1, j = h_1+h_2, ... , 2×h_1+h_2 に格納されます.

*は不定値を示します.

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 < 1 • nh1 ≥ n • nh1 < 0 • nh2 ≥ n • nh2 < 0 • k < 2 × nh1 + nh2 + 1 • epsz < 0	

3. 使用上の注意

a) epsz について

epsz が設定されると, LU 分解の過程でピボットが epsz 以下のときは相対的に零と見なし, icon = 20000 で処理を打ち切ります. epsz の標準値は丸め誤差の単位を u としたとき $16\times u$ です. なおピボットが小さくても処理を続行させたいときには, epsz に極小の値を設定すればよいのですが, その結果は保証されません.

b) ip について

本関数では, 部分ピボッティングに伴い, 行ベクトルの交換を行っています. すなわち, 分解の J 段階目($J = 1, \ldots, n$)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, 第 I 行と第 J 行の内容が交換されます. そして, その履歴を示すために ip[$J-1$]に I が格納されます.

c) 行列式の値について

行列式の値は, is および a[i][h_1+h_2], i = $0, \ldots, n-1$ を掛け合わせることで求めることができます.

4. 使用例

$n = 10000, nh_1 = 2000, nh_2 = 3000$ の実バンド行列を入力して, バンド行列の連立 1 次方程式を解きます.

```
#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 NH1   2000
#define NH2   3000
#define N     10000
#define KA    (2*NH1+NH2+1)
#define NWORK 4500

int MAIN__()
{
  double a[N][KA], b[N], dwork[NWORK];
  double tt1, tt2, tmp, epsz;
  int    ip[N], i, j, is, ix, isw, icon, nptr, nbase, nn;

  ix = 123;
  nn = NH1+NH2+1;
  for (i=0; i<N; i++) {
    c_dvrau4(&ix,&a[i][NH1],nn,dwork,NWORK,&icon);
  }

  printf("nh1 = %d, nh2 = %d, n = %d\n", NH1, NH2, N);

  /* zero clear */
  for (j=0; j<N; j++) {
    for (i=0; i<NH1; i++) {
      a[j][i] = 0.0;
    }
  }

  /* left upper triangular part */
  for (j=0; j<NH2; j++) {
    for (i=0; i<NH2-j; i++) {
      a[j][i+NH1] = 0.0;
    }
  }

  /* right rower triangular part */
  nbase = 2*NH1+NH2+1;
  for (j=0; j<NH1; j++) {
```

```
      for (i=0; i<j; i++) {
        a[N-NH1+j][nbase-i-1] = 0.0;
      }
    }

    /* set right hand constant vector */
    for (i=0; i<N; i++) {
      b[i] = 0.0;
    }

    for (i=0; i<N; i++) {
      nptr = i;
      for (j=max(nptr-NH2,0); j<min(N,nptr+NH1+1); j++) {
        b[j] += a[i][j-i+NH1+NH2];
      }
    }

    epsz = 0.0;
    isw  = 1;
    c_dm_vlbx((double*)a, KA, N, NH1, NH2, b, epsz, isw, &is, ip, &icon);

    tmp = 0.0;
    for (i=0; i<N; i++) {
      tmp = max(tmp,fabs(b[i]-1));
    }

    printf("maximum error = %e\n", tmp);
    return(0);
  }
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VLBX の項目を参照してください.

c_dm_vlcspsxcr1

非エルミート複素対称スパース行列の連立 1 次方程式 (不完全 ***LDL^T*** 分解に基づく前処理付き共役直交共役残差法 (*COCR* 法), 対称行列用圧縮行格納法)

`ierr = c_dm_vlcspsxcr1(zsa, nz, ncol, nfrnz,`
`n, zb, isw, zx, ipar, rpar, zvw,`
`&icon);`

1. 機能

$n \times n$ の非エルミート複素対称なスパース行列を係数行列とする連立 1 次方程式を, 不完全 **LDL^T** 分解に基づく前処理付きの共役直交共役残差法(*COCR* 法)で解きます.

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

$n \times n$ の複素係数行列 **A** は, 対称行列用の圧縮行格納法で格納します.

定数ベクトル **b** および解ベクトル **x** は n 次元ベクトルです.

2. 引数

呼出し形式:

```
ierr = c_dm_vlcspsxcr1(zsa, nz, ncol, nfrnz, n zb, isw, zx, ipar, rpar, zvw,
            &icon);
```

引数の説明:

`zsa`	`dcomplex zsa[nz]`	入力	係数行列の非零要素を格納します. 対称行列用の圧縮行格納法については, 図 c_dm_vlcspsxcr1-1 を参照してください.
`nz`	`int`	入力	係数行列 **A** の非零要素の総数. ($1 \le$ `nz`)
`ncol`	`int ncol[nz]`	入力	圧縮行格納法で使用される列指標で, 配列 `zsa` に格納される要素が何番目の列ベクトルに属するかを示します.
`nfrnz`	`int nfrnz[n+1]`	入力	行列 **A** の上三角行列部分の各行の非零要素を, 行方向に圧縮して順次配列 `zsa` に格納するとき, 対応する行の最初の非零要素が格納される位置を表します. `nfrnz[n]` = `nz` + 1
`n`	`int`	入力	行列 **A** の次数 n. ($1 \le$ `n`)
`zb`	`dcomplex zb[n]`	入力	連立 1 次方程式の右辺の定数ベクトル **b** を格納します.
`isw`	`int`	入力	方程式を解く方法を指定します(`isw` = 1 又は `isw` = 3). `isw` = 1: 初回目に方程式を解く場合に選択します. `isw` = 3: 直前に解いた方程式と同一の行列 **A** を持ち, ベクトル **b** の要素が異なる方程式を解く場合に選択します. `isw` = 3 で本ルーチンを呼び出す場合, 新たに設定するパラメタ `zb` および初期値 `zx` など変更のあるパラメタだけを修正し, それ以外のパラメタの内容は, 直前に呼び出した際の内容を変更せずに呼び出します.
`zx`	`dcomplex zx[n]`	入力	解ベクトル **x** の初期値を格納します.
		出力	解ベクトル **x** が格納されます.

ipar	int ipar[20]		動作調整用パラメタ(整数)を指定します. 演算結果の指標値が設定される場合もあります. 何れのパラメタも 0 を指定すれば, 内部でデフォルト値を設定します. デフォルト値で収束しない場合は, パラメタ値を変えて試みることを奨めます.
		入力	ipar[0] ～ [4]: 機能拡張用のリザーブ域で, 0 を指定しておきます.
		入力	ipar[5]: *COCR* 法の反復回数の上限を指定します ($0 \le$ ipar[5]). デフォルト値は 2000 です.
		出力	ipar[6]: 実際の反復回数が設定されます.
		出力	ipar[7]: *COCR* 法の反復で, 係数行列 **A** と反復過程のベクトル $\boldsymbol{v}$ の積 $\boldsymbol{Av}$ の評価回数が設定されます.
		入力	ipar[8] ～ [9]: 機能拡張用のリザーブ域で, 0 を指定しておきます.
		入力	ipar[10]: 前処理のための不完全 **LDL**$^{\mathbf{T}}$ 分解で, 新たな非零要素は破棄しますが, その補償を対角要素に反映するか否かを指定します. ipar[10] = 0 を指定すると補償しません. ipar[10] = 1 を指定すると補償します.デフォルト値は 0 です. ("使用上の注意" a)参照)
		出力	ipar[11]: 不完全 **LDL**$^{\mathbf{T}}$ 分解で破棄した非零の要素数が設定されます.
		入力	ipar[12] ～ [19]: 機能拡張用のリザーブ域で, 0 を指定しておきます.
rpar	double rpar[20]		動作調整用パラメタ(実数)を指定します. 演算後, 結果の指標値が設定される場合もあります. 何れのパラメタも 0.0 を指定すれば, 内部でデフォルト値を設定します. デフォルト値で収束しない場合は, パラメタ値を変えて試みることを奨めます.
		入力	rpar[0]: 機能拡張用のリザーブ域で, 0 を指定しておきます.
		入力	rpar[1]: *COCR* 法の反復で, 解の収束判定値 *epst* を指定します($0.0 \le epst$). デフォルト値は 10^{-8} です.
		出力	rpar[2]: 収束判定条件を満たす解の, 残差ベクトルに対する相対残差ノルムが設定されます.
		出力	rpar[3]: 不完全 **LDL**$^{\mathbf{T}}$ 分解で破棄した非零要素の累積値 *zdrp* の実部が設定されます. ("使用上の注意" a)参照)
		出力	rpar[4]: 不完全 **LDL**$^{\mathbf{T}}$ 分解で破棄した非零要素の累積値 *zdrp* の虚部が設定されます. ("使用上の注意" a)参照)
		入力	rpar[5] ～ [19]: 機能拡張用のリザーブ域で, 0.0 を指定しておきます.

zvw	dcomplex zvw[nz]	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	指定した反復回数では, 収束条件を満たさなかった.	配列 zx にはその時までに得られた近似値が格納される. 対応する相対残差ノルムも出力される.
29000	行列 **A** が非正則であった.	処理を打ち切る.
30000	パラメタに誤りがあった. • n < 1 • nz < 1 • nz ≠ nfrnz[n] - 1 • isw < 1 • isw = 2 • isw > 3 • ipar[5] < 0 • ipar[10] < 0 • ipar[10] > 1 • rpar[1] < 0.0	

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 & 3 & 0 \\ 2 & 5 & 0 & 6 \\ 3 & 0 & 8 & 9 \\ 0 & 6 & 9 & 11 \end{bmatrix}$$

$$\downarrow$$

$$\mathtt{nfrnz} = \begin{bmatrix} 1 \\ 4 \\ 6 \\ 8 \\ 9 \end{bmatrix}, \quad \mathtt{zsa} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 5 \\ 6 \\ 8 \\ 9 \\ 11 \end{bmatrix}, \quad \mathtt{ncol} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 2 \\ 4 \\ 3 \\ 4 \\ 4 \end{bmatrix}$$

図 c_dm_vlcspsxcr1-1 行列 A の対称行列用の圧縮行格納法

3. 使用上の注意

a) 新非零要素の破棄と補償について

本関数では, 不完全 $\mathbf{LDL^T}$ 分解の過程で新たに発生した非零要素は通常破棄します. 破棄した要素の影響を緩和するために, ipar[10]によってその補償を制御することができます. ipar[10] = 1 と指定すると, 破棄した要素分を当該行の対角要素に反映し補償します. この補償によって, 前処理行列としての特性が向上する場合があります.

更に本関数では, ipar[10]指定の内容に関わらず, 破棄した要素の累積値 *zdrp* を指標として設定します. rpar[3]と rpar[4]には, 累積値の実部と虚部の値がそれぞれ設定されます.

4. 使用例

複素対称行列を読み込み, 本関数で連立 1 次方程式の解を求めます. (並列実行を行うスレッド数は環境変数(OMP_NUM_THREADS)などで指定できます. 例えば, 4 プロセッサーのシステムで 4 つのスレッドで並列実行するときは, OMP_NUM_THREADS を 4 に設定して実行します.)

```
/* ======================================================
    TEST PROGRAM FOR KRYLOV ITERATION METHODS
    FOR SPARSE LINEAR EQUATIONS
    WITH NON-HERMIT COMPLEX SYMMETRIC MATRIX.
   ====================================================== */
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h"

#define NZMAX 500000
#define NMAX 10000

dcomplex comp_add(dcomplex, dcomplex);
dcomplex comp_mult(dcomplex, dcomplex);
void cmsvcr1(dcomplex*, int, int*, int*, dcomplex*, dcomplex*, int);
void creadmat(char*, double*, int*, int*, int*, double*);
void cmatcopy(dcomplex*, int, int*, int*, dcomplex*, dcomplex*, dcomplex*,
              int*, int*, dcomplex*, dcomplex*);
void cvecgen(dcomplex*, int, int*, int*, dcomplex*, dcomplex*);
double cnorm(dcomplex*, int);

int MAIN__() {

  dcomplex    zsa[NZMAX], zx[NMAX], zb[NMAX], zsat[NZMAX], zxt[NMAX],
              zbt[NMAX], zvw[NZMAX];
  int     nfrnz[NMAX+1], ncol[NZMAX], nfrnzt[NMAX+1], ncolt[NZMAX], ipar[20];
  double  rpar[20];
  char    title[74];

  int     n, nz, isw, ii, ic, icmav, mdrp, nzdrp, icon;
  double  epst, relres, drpr, drpi,rel, relerr;
```

```
/* ---------------------------------------------------
        INPUT MATRIX FROM UF SPARSE MATRIX COLLECTION
   --------------------------------------------------- */
  creadmat(title, (double *)zsat, &n, nfrnzt, ncolt, (double *)zsa);
  cvecgen(zsat, n, nfrnzt, ncolt, zxt, zbt);
  cmatcopy(zsat, n, nfrnzt, ncolt, zxt, zbt, zsa, nfrnz, ncol, zx, zb);

  printf(
    "\n-------------------------------------------------------------------\n");
  printf("TEST MATRIX : \n%s\n", title);
/* --------------------------------------------------- */
  isw = 1;
  for (ii = 0; ii < 20; ii++) {
    ipar[ii] = 0;
    rpar[ii] = 0.0;
  }
  nz = nfrnz[n] - 1;
  c_dm_vlcspsxcr1(zsa, nz, ncol, nfrnz, n, zb,
                  isw, zx, ipar, rpar, zvw, &icon);

  ic = ipar[6];
  icmav = ipar[7];
  mdrp = ipar[10];
  nzdrp = ipar[11];
  epst = rpar[1];
  relres = rpar[2];
  drpr = rpar[3];
  drpi = rpar[4];
  rel = cnorm(zb, n);
  cmsvcr1(zsa, n, nfrnz, ncol, zx, zb, 0);
  relerr = cnorm(zb, n) / rel;

  printf(
    "\n-------------------------------------------------------------------\n");
  printf(" SOLUTION RESULTS BY \"C_DM_VLCSPSXCR1\"\n\n");
  printf(" N           =%12d\n", n);
  printf(" NZ          =%12d\n", nfrnz[n]-1);
  printf(" MDRP        =%12d\n\n", mdrp);
  printf(" ICON        =%12d\n", icon);
  printf(" IC          =%12d\n", ic);
  printf(" ICMAV       =%12d\n", icmav);
  printf(" NZDRP       =%12d\n", nzdrp);
  printf(" DRPR        =%12.2le\n", drpr);
  printf(" DRPI        =%12.2le\n", drpi);
  printf(" EPST        =%12.2le\n", epst);
  printf(" RELRES      =%12.2le\n", relres);
  printf(" RELERR      =%12.2le\n", relerr);
  printf(
    "-------------------------------------------------------------------\n");
```

```
  if ((relerr <= epst * 1.1) && (icon == 0)) {
    printf("  ********** OK **********\n");
  } else {
    printf("  ********** NG **********\n");
  }
  return(0);
}

dcomplex comp_add(dcomplex so1, dcomplex so2) {

  dcomplex obj;

  obj.re = so1.re + so2.re;
  obj.im = so1.im + so2.im;
  return obj;
}

dcomplex comp_mult(dcomplex so1, dcomplex so2) {

  dcomplex obj;

  obj.re = so1.re * so2.re - so1.im * so2.im;
  obj.im = so1.re * so2.im + so1.im * so2.re;
  return obj;
}

/* ================================================
        MATRIX VECTOR MULTIPLICATION.
        COMPLEX SYMMETRIC MATRIX STORED IN CSR FORM.
   ================================================ */
void cmsvcr1(dcomplex *zsa, int n, int *nfrnz, int *ncol, dcomplex *zx,
       dcomplex *zb, int isw) {
  int     i, j, k1, k2;
  dcomplex    zsa_w;

  if (isw == 1) {     /* *** MULTIPLICATION (AX=>B) */
    for (i = 0; i < n; i++) {
      zb[i].re = 0.0;
      zb[i].im = 0.0;
    }
    for (i = 0; i < n; i++) {
      k1 = nfrnz[i] - 1;
      k2 = nfrnz[i+ 1] - 1;
      if (zx[i].re != 0.0 || zx[i].im != 0.0) {
        for (j = k1; j < k2; j++) {
          zb[ncol[j] - 1] = comp_add(comp_mult(zsa[j], zx[i]),
                              zb[ncol[j] - 1]);

          if (ncol[j] != i + 1)
```

```
            zb[i] = comp_add(comp_mult(zsa[j], zx[ncol[j] -1]), zb[i]);
          }
        } else {
          for (j = k1; j < k2; j++) {
            zb[i] = comp_add(comp_mult(zsa[j], zx[ncol[j] - 1]), zb[i]);
          }
        }
      }
    } else {    /* *** RESIDUAL VECTOR (B-AX=>B) */
      for (i = 0; i < n; i++) {
        k1 = nfrnz[i] - 1;
        k2 = nfrnz[i + 1] - 1;
        if (zx[i].re != 0.0 || zx[i].im != 0.0) {
          for (j = k1; j < k2; j++) {
            zsa_w = zsa[j];
            zsa_w.re = -zsa_w.re;
            zsa_w.im = -zsa_w.im;
            zb[ncol[j] - 1] = comp_add(comp_mult(zsa_w, zx[i]), zb[ncol[j] - 1]);
            if (ncol[j] != i + 1) {
              zsa_w = zsa[j];
              zsa_w.re = -zsa_w.re;
              zsa_w.im = -zsa_w.im;
              zb[i] = comp_add(comp_mult(zsa_w, zx[ncol[j] - 1]), zb[i]);
            }
          }
        } else {
          for (j = k1; j < k2; j++) {
            zsa_w = zsa[j];
            zsa_w.re = -zsa_w.re;
            zsa_w.im = -zsa_w.im;
            zb[i] = comp_add(comp_mult(zsa_w, zx[ncol[j] - 1]), zb[i]);
          }
        }
      }
    }
    return;
}

/* ================================================
        READ TEST MATRIX FOR COMPLEX SYMMETRIC MATRIX.
   ================================================ */
void creadmat(char *title, double *a, int *ncol, int *is, int *js, double *w) {

/*  THIS ROUTINE READS MATRIX DATA OF RB SPARSE FORM.
    THE FOLLOWING SAMPLE CODE IS ORIGINATED FROM MATRIX
    MARKET;                                          */

  char  key[11], mxtype[4], rhstyp[4],
        ptrfmt[17], indfmt[17], valfmt[21], rhsfmt[23];
```

```
  char  dummy[12];
  int  totcrd, ptrcrd, indcrd, valcrd, rhscrd,
       nrow, nnzero, neltvl,
       nrhs, nrhsix;
  int  i;
/*  -----------------------
     READ IN HEADER BLOCK
    ----------------------- */
  scanf("%72c%8c", title, key);
  title[72] ='\0';
  scanf("%14d%14d%14d%14d%14d", &totcrd, &ptrcrd, &indcrd,
      &valcrd, &rhscrd);
  scanf("%3c%11c%14d%14d%14d%14d", mxtype, dummy, &nrow, ncol,
      &nnzero, &neltvl);
  scanf("%16c%16c%20c%20c", ptrfmt, indfmt, valfmt, rhsfmt);
  if (rhscrd > 0) {
    scanf("%3c%11c%14d%14d", rhstyp, dummy, &nrhs, &nrhsix);
  }
/*   ------------------------
      READ MATRIX STRUCTURE
     ------------------------ */
  for (i = 0; i <= *ncol; i++) {
    scanf("%5d", &is[i]);
  }
  for (i = 0; i < nnzero; i++) {
    scanf("%4d", &js[i]);
  }

  if (valcrd > 0) {
/*  ----------------------
     READ MATRIX VALUES
    ---------------------- */
    if (mxtype[0] == 'R') {
      for (i = 0; i < nnzero; i++) {
        scanf("%le", &a[i]);
      }
    } else {
      for (i = 0; i < 2 * nnzero; i++) {
        scanf("%le", &a[i]);
      }
    }
  }
  return;
}

/* ================================================
        COPY COMPLEX MATRIX AND VECTORS.
   ================================================ */
void cmatcopy(dcomplex *zsat, int n, int *nfrnzt, int *ncolt,
```

```
        dcomplex *zxt, dcomplex *zbt, dcomplex *zsa, int *nfrnz, int *ncol,
        dcomplex *zx, dcomplex *zb) {
  int  nz, i;

  nz = nfrnzt[n] - 1;
  for (i = 0; i <= n; i++) {
    nfrnz[i] = nfrnzt[i];
  }
  for (i = 0; i < nz; i++) {
    zsa[i] = zsat[i];
    ncol[i] = ncolt[i];
  }

  for (i = 0; i < n; i++) {
    zx[i] = zxt[i];
    zb[i] = zbt[i];
  }
  return;
}

/* ================================================
        GENERATE COMPLEX B AND X VECTORS.
   ================================================ */
void cvecgen(dcomplex *zsat, int n, int *nfrnzt, int *ncolt, dcomplex *zxt,
       dcomplex *zbt) {
  int  ii;

/*  COMPUTE RIGHT HAND SIDE VECTOR B. */
  for (ii = 1; ii <= n; ii++) {
    zxt[ii - 1].re = 1.0 + (double)ii / (double)n;
    zxt[ii - 1].im = 0.0;
  }
  cmsvcr1(zsat, n, nfrnzt, ncolt, zxt, zbt, 1);

/*  SET INITIAL VALUE */
  for (ii = 0; ii < n; ii++) {
    zxt[ii].re = 0.0;
    zxt[ii].im = 0.0;
  }
  return;
}

/* ================================================
        L2 NORM OF A COMPLEX VECTOR.
   ================================================ */
double cnorm(dcomplex *zx, int n) {
  int  i;
  double  cnorm_ret;
```

```
  cnorm_ret = 0.0;
  for (i = 0; i < n; i++) {
    cnorm_ret +=  (zx[i].re * zx[i].re + zx[i].im * zx[i].im);
  }
  if (cnorm_ret != 0.0)
    cnorm_ret = sqrt(cnorm_ret);
  return(cnorm_ret);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VLCSPSXCR1 の項目及び[66], [74]を参照してください.

c_dm_vlcx

複素行列の連立 1 次方程式 (ブロック化された LU 分解法)
`ierr = c_dm_vlcx(za, k, n, zb, epsz, isw, &is, ip, &icon);`

1. 機能

複素係数連立 1 次方程式

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

をブロック化した外積形の LU 分解法で解きます. ただし, **A** は $n \times n$ の正則な複素行列, **b** は n 次元の複素定数ベクトル, **x** は n 次元の解ベクトル. ($n \geq 1$)

2. 引数

呼出し形式:

```
ierr = c_dm_vlcx((dcomplex*)za, k, n, zb, epsz, isw, &is, ip, &icon);
```

引数の説明:

za	dcomplex	入力	係数行列 **A**.
	za[n][k]	出力	行列 **L** と行列 **U**. 演算後, 内容は保存されません.
k	int	入力	配列 za の整合寸法 (≥ n).
n	int	入力	係数行列 **A** の次数 n.
zb	dcomplex	入力	定数ベクトル **b**.
	zb[n]	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ます. ("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, 次のとおり指定します. isw = 1 のとき, 1 組目の方程式を解きます. isw = 2 のとき, 2 組目以降の方程式を解きます. ただし, このとき zb の値だけを新しい定数ベクトルの値に変え, それ以外の引数はそのままで使います. ("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ます. ("使用上の注意" b)参照)
ip	int ip[n]	出力	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.
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 • isw ≠ 1 , 2	

3. 使用上の注意

a) epsz について

epsz に値を設定した場合, ピボットが epsz 以下なら零と見なし, icon = 20000 として処理を打ち切ります. epsz の標準値は丸め誤差の単位μとしたとき, epsz = 16μです. 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のですが , その結果は保証されません.

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ります. なお, この場合 is の値は, isw = 1 のときの値が保証されます.

4. 使用例

複素行列の連立 1 次方程式を解きます.

```
#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 N  (2000)
#define K  (N+1)

MAIN__()
{
  dcomplex za[N][K], zb[N];
  double   epsz, c, t, s, error;
  int      ip[N];
  int      isw, is, icon, i, j;

  c = sqrt(1.0/(double)(N+1));
  t = atan(1.0)*8.0/(N+1);

  for (j=0; j<N; j++) {
    for (i=0; i<N; i++) {
      za[j][i].re = c*cos(t*(i+1)*(j+1));
      za[j][i].im = c*sin(t*(i+1)*(j+1));
    }
  }

  for (i=0; i<N; i++) {
    s = 0.0;
    for (j=0; j<N; j++) {
      s += cos(t*(i+1)*(j+1));
      zb[i].re = s*c;
      zb[i].im = 0.0;
    }
  }
```

```
  epsz = 0.0;
  isw = 1;
  c_dm_vlcx((dcomplex*)za, K, N, zb, epsz, isw, &is, ip, &icon);

  printf("icon    = %d\n", icon);

  error = 0.0;

  for (i=0; i<N; i++) {
    error = max(fabs(1.0-zb[i].re), error);
  }

  printf("error   = %f\n", error);
  printf("ORDER   = %d\n", N);
  printf("zb[0]   = %e\n", zb[0].re);
  printf("zb[n-1] = %e\n", zb[N-1].re);

  return(0);
}
```

c_dm_vldlx

LDL^T分解された正値対称行列の連立 1 次方程式
`ierr = c_dm_vldlx(b, fa, kfa, 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)を解きます.

$$\mathbf{L}\mathbf{D}\mathbf{L}^T\mathbf{x} = \mathbf{b} \tag{1}$$

ただし, **L**, **D** はそれぞれ $n \times n$ の単位下三角行列と対角行列, **b** は n 次元の実定数ベクトル, **x** は n 次元の解ベクトルです. ($n \geq 1$)

本関数は, 関数 c_dm_vsldl 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す.

2. 引数

呼出し形式:

```
ierr = c_dm_vldlx(b, (double*)fa, kfa, n, &icon);
```

引数の説明:

`b`	`double b[n]`	入力	定数ベクトル **b**.
		出力	解ベクトル **x**.
`fa`	`double fa[n][kfa]`	入力	LDL^T分解された行列 **L**, $\mathbf{D}^{-1}$ および $\mathbf{L}^T$ を, `fa` の上三角部分 `fa[i-1][j-1]`, `i` ≤ `j` に格納します. 格納法の詳細については, 図 c_dm_vldlx-1.を参照してください.
`kfa`	`int`	入力	配列 `fa` の整合寸法 (≥ `n`).
`n`	`int`	入力	行列 **L**, **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	係数行列が正値でなかった.	処理は続行する.
30000	`n` < 1, `kfa` < `n`	処理を打ち切る.

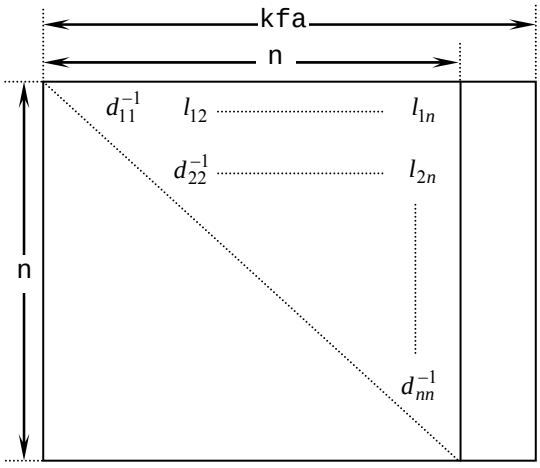


図 c_dm_vldlx-1. 配列 fa へ行列 **L**, $\mathbf{D}^{-1}$ および $\mathbf{L}^{\mathrm{T}}$ を格納する方法

LDL^{T} 分解された行列 **L**, $\mathbf{D}^{-1}$ および $\mathbf{L}^{\mathrm{T}}$ を, 配列 fa[i−1][j−1], i=1, ... , n, j=i, ... , n に格納します.

3. 使用上の注意

本関数を, 関数 c_dm_vsldl 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ます. しかし, 通常は, 関数 c_dm_vlsx を呼び出せば, 一度に解が求められます.

4. 使用例

1000×1000 の行列を LDL^{T} 分解し, 続いて連立 1 次方程式を解きます.

```
#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 NMAX      (1000)
#define LDA       (NMAX+1)

MAIN__()
{
  int    n, i, j, icon, ierr;
  double a[NMAX][LDA], b[NMAX];
  double epsz, s, det;

  n    = NMAX;
  epsz = 0.0;

#pragma omp parallel for shared(a,n) private(i,j)
  for(i=0; i<n; i++)
    for(j=0; j<n; j++) a[i][j] = min(i,j)+1;

#pragma omp parallel for shared(b,n) private(i)
  for(i=0; i<n; i++) b[i] = (i+1)*(i+2)/2+(i+1)*(n-i-1);

  ierr = c_dm_vsldl((double*)a, LDA, n, epsz, &icon);

  if (icon != 0) {
    printf("ERROR: c_dm_vsldl failed with icon = %d\n", icon);
    exit(1);
  }

  ierr = c_dm_vldlx(b, (double*)a, LDA, n, &icon);

  if (icon != 0) {
```

```
    printf("ERROR: c_dm_vldlx failed with icon = %d\n", icon);
    exit(1);
  }

  s = 1.0;
#pragma omp parallel for shared(a,n) private(i) reduction(*:s)
  for(i=0; i<n; i++) s *= a[i][i];

  printf("solution vector:\n");
  for(i=0; i<10; i++) printf("    b[%d] = %e\n", i, b[i]);

  det = 1.0/s;
  printf("\ndeterminant of the matrix = %e\n", det);
  return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VLDLX の項目及び[54]を参照してください.

c_dm_vlspaxcr2

実非対称スパース行列の連立 1 次方程式 (近似逆行列に基づく前処理付きの演繹的次元縮小法 (IDR 法), 圧縮行格納法)

`ierr = c_dm_vlspaxcr2(a, nz, ncol, nfrnz, n,`
`b, isw, x, am, nzm, ncolm,`
`nfrnzm, nwm, ipar, rpar, vw1,`
`ivw1, vw2, ivw2, lmmax, lnmax,`
`numt, &icon);`

1. 機能

$n \times n$ の実非対称なスパース行列を係数行列とする連立 1 次方程式を, 近似逆行列に基づく前処理付きの演繹的次元縮小法 (*IDRstab(s,l)*法) で解きます.

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

$n \times n$ の実係数行列 **A** は, 圧縮行格納法で格納します.

定数ベクトル **b** および解ベクトル **x** は n 次元ベクトルです.

また, s はシャドウ残差の次数, l は加速多項式の次数です.

2. 引数

呼出し形式:

```
ierr = c_dm_vlspaxcr2(a, nz, ncol, nfrnz, n b, isw, x, am, &nzm, ncolm,
            nfrnzm, nwm, ipar, rpar, vw1, ivw1, (double*)vw2, (int*)ivw2,
            lmmax, lnmax, numt, &icon);
```

引数の説明:

a	double a[nz]	入力	係数行列の非零要素を格納します. 圧縮行格納法は, 行列 **A** の転置行列を圧縮列格納法で格納したもの. 圧縮列格納法については, 図 c_dm_vmvscc-1 を参照してください.
nz	int	入力	係数行列 **A** の非零要素の総数. ($1 \le$ nz)
ncol	int ncol[nz]	入力	圧縮行格納法で使用される列指標で, a に格納される要素が何番目の列ベクトルに属するかを示します.
nfrnz	int nfrnz[n+1]	入力	行列 **A** の各行の非零要素を行方向に圧縮して順次配列 a に格納するとき, 対応する行の最初の非零要素が格納される位置を表します. nfrnz[n] = nz + 1.
n	int	入力	行列 **A** の次数 n. ($1 \le$ n)
b	double b[n]	入力	連立 1 次方程式の右辺の定数ベクトル **b** を格納します.
isw	int	入力	方程式を解く方法を指定します($1 \le$ isw ≤ 3). isw = 1: 初回目に方程式を解く場合に選択します. isw = 3: 直前に解いた方程式と同一のスパース構造を持ち, 行列 **A** とベクトル **b** の要素が異なる方程式を解く場合に選択します.

			isw = 2 又は 3 で本ルーチンを呼び出す場合, 新たに設定するパラメタ a, b および初期値 x ど変更のあるパラメタだけを修正し, それ以外のパラメタの内容は, 直前に呼び出した際の内容を変更せずに呼び出します.
x	double x[n]	入力	解ベクトル **x** の初期値を格納します.
		出力	解ベクトル **x** が格納されます.
am	double am[nwm]	入力	初期近似逆行列 $\mathbf{M_0}$ を指定する場合, その非零要素を圧縮行格納法で am[i-1], i = 1, ... , nzm に格納します. 圧縮行格納法は, 行列 **A** の格納法と同じです.
		出力	計算された近似逆行列が格納されます.
nzm	int	入力	初期近似逆行列 $\mathbf{M_0}$ の非零要素の総数. (0 ≤ nzm) 初期近似逆行列を指定しない場合は, nzm = 0 と指定します. この場合は初期近似逆行列として単位行列が内部で採用されます.
		出力	計算された近似逆行列 **M** の非零要素の総数が格納されます.
ncolm	int ncolm[nwm]	入力	初期近似逆行列 $\mathbf{M_0}$ を指定する場合, 圧縮行格納法の列指標で, am に格納される要素が何番目の列ベクトルに属するかを示します.
		出力	計算された近似逆行列 **M** の, 圧縮行格納法の列指標.
nfrnzm	int nfrnzm[n+1]	入力	初期近似逆行列 $\mathbf{M_0}$ を指定する場合, 各行の非零要素を行方向に圧縮して順次配列 am に格納するとき, 対応する行の最初の非零要素が格納される位置を表します. nfrnzm[n] = nzm + 1.
		出力	計算された近似逆行列 **M** の, 各行の最初の非零要素が格納される位置.
nwm	int	入力	近似逆行列を求める領域の最大サイズを指定します. (n ≤ nwm) 行列 **A** の第 k 列の非零要素数を nz_k として, 近似逆行列のサイズ nzm を下記の式で算定します. $nzm = \sum_{k=1}^{n} \max(1, nz_k \times ipar[1]/100)$ この値より, nwm を下記の式に従って指定します. $nwm = \max(nzm, nz)$ (“使用上の注意” a)参照)
ipar	int ipar[20]		動作調整用パラメタ(整数)を指定します. 演算結果の指標値が設定される場合もあります. 何れのパラメタも 0 を指定すれば, 内部でデフォルト値を設定します. デフォルト値で収束しない場合は, パラメタ値を変えて試みることを奨めます.
		入力	ipar[0]: 機能拡張用のリザーブ域で, 0 を指定しておきます.
		入力	ipar[1]: 近似逆行列の非零要素数の総数の上限として, 行列 ***A*** の非零要素数の総数 NZ に対する比率(%)を指定します. (0 ≤ ipar[1]) 例えば ipar[1] = 50 と指定した場合, 行列 ***A*** の非零要素数のおおよそ 50%の非零要素数を上限に持つ近似逆行列が生成されます. デフォルト値は 100 です. (“使用上の注意” c)参照)

		入力	ipar[2]:	近似逆行列の各列ベクトルの計算で, 順次追加する新たなインデックスの増量を指定します. ($0 \le$ ipar[2] $\le n$) 例えば ipar[2] = 2 と指定した場合, 生成する近似逆行列の非零要素数を, 効果の高い順番に 2 個ずつ順次増加させます. デフォルト値は 1 です. ("使用上の注意" d)参照)
		入力	ipar[3]:	演繹的次元縮小法 *IDRstab(s,l)* 法のシャドウ残差の次数 *s* を指定します. ($0 \le s \le n$) デフォルト値は 4 です.
		入力	ipar[4]:	演繹的次元縮小法 *IDRstab(s,l)* 法の加速多項式の次数 *l* を指定します. ($0 \le l \le n$) デフォルト値は 1 です.
		入力	ipar[5]:	入力. *IDRstab(s,l)* 法の反復回数の上限を指定します. ($0 \le$ ipar[5]) デフォルト値は 2000 です.
		出力	ipar[6]:	実際の反復回数が設定されます.
		出力	ipar[7]:	*IDRstab(s,l)* 法の反復で, 係数行列 $\boldsymbol{A}$ と反復過程のベクトル $\boldsymbol{v}$ の積 $\boldsymbol{Av}$ の評価回数が設定されます.
		出力	ipar[8]:	近似逆行列の領域 am, ncolm のサイズ nwm の予測値が設定されます. ("使用上の注意" a)参照)
		入力	ipar[9] ～ [11]:	機能拡張用のリザーブ域で, 0 を指定しておきます.
		出力	ipar[12]:	作業用配列 vw2, ivw2 で 実際に使用した lmmax の値.
		出力	ipar[13]:	作業用配列 vw2 で 実際に使用した lnmax の値.
		入力	ipar[14] ～ [19]:	機能拡張用のリザーブ域で, 0 を指定しておきます.
rpar	double rpar[20]			動作調整用パラメタ(実数)を指定します. 演算後, 結果の指標値が設定される場合もあります. 何れのパラメタも 0.0 を指定すれば, 内部でデフォルト値を設定します. デフォルト値で収束しない場合は, パラメタ値を変えて試みることを奨めます.
		入力	rpar[0]:	近似逆行列の計算で, 列ベクトルを求める収束判定値 *eps* を指定します. ($0.0 \le eps$) デフォルト値は 0.3 です.
		入力	rpar[1]:	*IDRstab(s,l)* 法の反復で, 解の収束判定値 *epst* を指定します. ($0.0 \le epst$) デフォルト値は 10^{-8} です.
		出力	rpar[2]:	収束判定条件を満たす解の, 残差ベクトルに対する相対残差ノルムが設定されます.
		入力	rpar[3] ～ [19]:	機能拡張用のリザーブ域で, 0.0 を指定しておきます.
vw1	double vw1[nwm]	作業領域		

ivw1	int ivw1[nwm]	作業領域	
vw2	double vw2[numt][lnmax+3][lmmax]	作業領域	
ivw2	int ivw2[numt][3][lmmax]	作業領域	
lmmax	int	入力	作業用配列の第 3 添字の大きさ. (1 ≤ lmmax) 行列 **A** の非零要素に関連する値ですが, 特定の列に着目した時, その列に含まれる非零要素の数と, その非零要素に連動する他の列に含まれる非零要素の総数を定義します. lmmax には, 列毎の総数の中で最大の値を指定します. 通常 1000 程度で充分ですが, 解が得られない場合は値を拡大し試みることを奨めます. ("使用上の注意" e)参照)
lnmax	int	入力	作業用配列の第 2 添字に関連する大きさ. (1 ≤ lnmax) 近似逆行列 **M** の各列ベクトルに含まれる非零要素の数の中で, 最大の値に比例した値を指定します. 標準的な使用の場合は, 行列 **A** の各列ベクトルの中で, 非零要素の数の最大値を目安に設定します. 解が得られない場合は, 値を拡大し試みることを奨めます. ("使用上の注意" e)参照)
numt	int	入力	作業用配列の第 1 添字の大きさ. (1 ≤ numt) 本関数で並列実行を行う最大のスレッド数を指定します.
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.
11000	行列 **A** に非正則の可能性があった.	処理を続行する.
19000	対角要素が無い行があった.	
20000	指定した反復回数では, 収束条件を満たさなかった.	処理を打ち切る. 配列 x にはその時までに得られた近似値が格納される. 対応する相対残差ノルムも出力される.
25000	nwm の値が小さく, 近似逆行列の領域 am, ncolm が不足した.	処理を打ち切る. ipar[8]に必要となる最低の大きさの予測値 nwm が設定される.
26000	lmmax の値が小さく, 作業領域 vw2, ivw2 が不足した.	処理を打ち切る.
27000	lnmax の値が小さく, 作業領域 vw2 が不足した.	
29000	行列 **A** が非正則であった.	

コード	意味	処理内容
30000	パラメタに誤りがあった. • n < 1 • nz < 1 • nz ≠ nfrfz[n] - 1 • isw < 1 • isw > 3 • nwm < n • nzm < 0 • ipar[1] < 0 • ipar[2] < 0 • ipar[3] < 0 • n < ipar[3] • ipar[4] < 0 • n < ipar[4] • ipar[5] < 0 • lmmax < 1 • lnmaz < 1 • numt < 1 • rpar[0] < 0.0 • rpar[1] < 0.0	処理を打ち切る.
30011	行列 **A** に関するパラメタに誤りがあった. 何れかのインデックスに以下の関係があった. nfrnz[*k*] > nfrnz[*k*+1], *k*=0,...,*n*-1	
30012	行列 **A** に関するパラメタに誤りがあった. 何れかのインデックスに以下の関係があった. ncol[*l*] > ncol[*l*+1], *l* = nfrnz[*k*] ,..., nfrnz[*k*+1], *k* = 0 ,..., *n*-1	
30021	初期近似逆行列 $\mathbf{M_0}$ に関するパラメタに誤りがあった. 何れかのインデックスに以下の関係があった. nfrnz[*k*]>nfrnz[*k*+1], *k*=0,...,*n*-1	
30022	初期近似逆行列 $\mathbf{M_0}$ に関するパラメタに誤りがあった. 何れかのインデックスに以下の関係があった. ncolm[*l*] > ncolm[*l*+1], *l* = nfrnzm[*k*] ,..., nfrnzm[*k*+1], *k* = 0 ,..., *n*-1	

3. 使用上の注意

a) 近似逆行列に関する領域のサイズについて

近似逆行列 **M** のサイズ *nzm* は, 行列 **A** の第 *k* 列の非零要素数を nz_k とすると,

$$nzm = \sum_{k=1}^{n} \max(1, nz_k \times \mathtt{ipar[1]} / 100)$$

で算定されます. 確保する領域のサイズ nwm は,

$$nwm = \max(nzm, nz)$$

となります. 近似逆行列の非零要素数の上限に関するパラメタ ipar[1]を, 標準的な値として使用する場合(ipar[1] = 0)は, nwm = nz と指定します. 上式による nwm の算定が困難な場合, 十分に大きな値

(nwm $= 2 \times$ nz など) を設定して呼び出します. 本関数の呼び出しの結果, 予測されるサイズが ipar[8]に設定されます. この値は, 類似した行列 **A** に関する方程式を解く際に, 有効な目安を与えます. スパース構造が同一で, 近似逆行列の非零要素の比率も同等の場合は, 後続する呼び出しで, ipar[8]の値を nwm の目安とします. 一方, 行列 **A** の非零要素数が多くなる場合や, 近似逆行列 **M** の非零要素の比率を上げる場合は, ipar[8]にそれぞれの増加の比率を掛けた量を nwm の目安とします.

b) 初期近似逆行列の設定について

本関数を呼び出すにあたって, 前処理のための近似逆行列の初期行列 $\mathbf{M}_0$ を設定することができます. この場合, 初期行列の非零要素の総数を nzm に指定し, 初期行列を圧縮行格納法でパラメタ am, ncolm, nfrnzm にそれぞれ設定します. この初期近似行列の設定によって, 本関数を用いて以下の様な問題を解く場合に, 効率的に処理することができます.

1) スパース構造が同一で, 値が異なる係数行列 **A** や定数ベクトル *b* を持つ 複数組みの方程式を解く場合.
2) 類似したスパース構造を持つ複数組みの方程式を解く場合.

これらの処理は isw と合わせて制御します. この処理では, a と関連するパラメタや b,x の値など変更のあるパラメタだけを修正し, パラメタ am や作業用配列などその他のパラメタの内容は, 本ルーチンを直前に呼び出した結果を保持したままで呼び出します. この場合, 初期行列の非零要素の総数の上限 ipar[1]を増加させることが可能です.

c) 近似逆行列 *M* の非零要素数について

本関数では, 近似逆行列 **M** に基づく前処理付き連立 1 次方程式

AMy = b, x = My

を解きます. ここで行列 **M** は, **AM** ≒ **I** となる様に近似しますが, 行列 **M** の非零要素数は, 反復の過程で頻繁に実行する行列ベクトル積 **x = My** の効率とのトレードオフがあります. 本関数では, パラメタ ipar[1] によって, 生成する非零要素数を制御します. 一般的には, 行列 **A** の非零要素数 *nz* と同等程度であること, 即ち ipar[1] = 100 が推奨されます.

本関数では, 近似逆行列 **M** を列ベクトル $\boldsymbol{m}_k$, $k = 1,\ldots, n$ 毎に計算します. 行列 **A** の第 *k* 列の非零要素数を nz_k とすると, 近似逆行列の第 *k* 列の非零要素数の上限を $nz_k \times$ ipar[1] / 100 として求めます. その過程で, 収束条件

$$\left\| \boldsymbol{A}\boldsymbol{m}_k - \boldsymbol{e}_k \right\|_2 \le eps$$

を満たした場合は, $\boldsymbol{m}_k$ の非零要素の数が上限 $nz_k \times$ ipar[1] / 100 に達していなくても, その時点で近似逆行列の該当する列ベクトル $\boldsymbol{m}_k$ の計算を終了します.

d) 近似逆行列 *M* の計算におけるインデックスの増分について

列ベクトル $\boldsymbol{m}_k$ の計算は, 以下の最小二乗問題を解くことで求めます.

$$\min_{m_k} \left\| \boldsymbol{A}\boldsymbol{m}_k - \boldsymbol{e}_k \right\|_2, k = 1,\ldots,n$$

ここで $\boldsymbol{e}_k$ は単位ベクトルです. この最小二乗問題の解に基づく残差ベクトルで, $\boldsymbol{m}_k$ に新たに発生する非零要素の中から, 残差ベクトルを最小化する最も貢献度が大きい要素を優先的に選択します. インデックスの増分は, 優先的に選択する個数を指定するものです. 列ベクトル $\boldsymbol{m}_k$ のスパース性を維持するために, 徐々に非零要素を増加させること, 即ち ipar[2] = 1 が推奨されます.

e) 作業領域 vw2, ivw2 について

作業用配列 vw2, ivw2 は, それぞれ 3 次元配列です. これらの配列は, 近似逆行列 **M** の列ベクトル $\boldsymbol{m}_k$ を計算する最小二乗問題を解くための領域として使用されます. 一般に $\boldsymbol{m}_k$ は, スパースベクトルですが, 近似逆行列を求める過程でそのスパース性が変化します. d 項の式に従って最小二乗問題を定義しますが, 残差ベクトル $\boldsymbol{A}\boldsymbol{m}_k$ - $\boldsymbol{e}_k$ はスパースベクトル $\boldsymbol{m}_k$ の非零要素に連動した行列 **A** の列ベクトルだけが関与し, その非零要素だけからなる矩形の最小二乗問題が定義されます. この矩形の方程式のサイズの最大値を lmmax, lnmax として設定し, 配列 vw2, ivw2 を確保します. 本関数内部で実際に必要とするこの矩形領域のサイズは, 行列 **A** の特性や ipar[1]など近似逆行列を求めるための各種パラメタの設定方法に応じて変化しま

す. 従って, 以下に述べる目安で設定して本関数を呼び出すこと, 解が得られない場合は, 更に値を拡大し試みることを奨めます.
lnmax は, 近似逆行列 **M** の各列ベクトルに含まれる非零要素の中で, 最大の値に比例した値を指定します. 最大の値は, 下記の式で算定します.

$$\max_k \left[\max(\ 1, nz_k \times ipar\ [1]/100\ \ )\right]$$

lnmax には, 最大値予測の 1.2 倍程度の値を指定します.
正常に処理を終了した場合, 実際に使用した領域の大きさが, lmmax と lnmax に対応してそれぞれ ipar[12]と ipar[13]に格納されます.

4. 使用例

連立 1 次方程式 **Ax** = **f** を解きます. 行列 **A** は境界条件として立方体の境界で 0 となる楕円型の偏微分方程式に有限差分法を適用して生成されます.

$$-\Delta \mathbf{u} + \mathbf{a} \nabla \mathbf{u} + \mathbf{u} = \mathbf{f}$$

ここで $\boldsymbol{a} = (a_1, a_2, a_3)$, a_1, a_2 および a_3 はある定数です. 行列 **A** はルーチン init_mat_diag によって生成されます. これを圧縮型格納法に変換します. (並列実行を行うスレッド数は環境変数(OMP_NUM_THREADS)などで指定できます. 例えば, 4 プロセッサーのシステムで 4 つのスレッドで並列実行するときは, OMP_NUM_THREADS を 4 に設定して実行します.)

```
/* **EXAMPLE** */
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <malloc.h>
#include <omp.h>
#include "cssl.h"

#define  NORD   60
#define  NX     NORD
#define  NY     NORD
#define  NZ     NORD
#define  N      (NX * NY * NZ)
#define  K      (N + 1)
#define  NDIAG  7
#define  L      4
#define  LMMAX  1000
#define  LNMAX  200
#define  NUMT   4

double errnrm(double*, double*, int);
void init_mat_diag(double, double, double, double, double*, int*, int, int,
                   int, double, double, double, int, int, int);
void convgcr(double*, int, int*, int*, double*, int*, int*, int*);

int MAIN__() {
  int  nofst[NDIAG];
  int  nrow[K * NDIAG], nfcnz[K], iw[K * NDIAG][2];
  int  ivw[N];
  int  *ivw2;
  int  ipar[20];
  int  nfrnz[K], nfrnzm[K];
  int  j, l, nbase, length, numnz, ncoll, ntopcfg, nnz, icon, isw, nwm,
       nzm, itmax, icont;
  int  i;
  double  diag[NDIAG][K], diag2[NDIAG][K];
  double  a[K * NDIAG], w[K * NDIAG];
  double  x[N], b[N], solex[N], y[N];
  double  *vw2;
  double  rpar[20];
  double  va1, va2, va3, vc, xl, yl, zl, err1, err2, err3, err4, eps;

  double  *aa, *am, *vw1;
  int *ncol, *ncolm, *ivw1;
```

```
vw2  = (double *)malloc(LMMAX * (LNMAX + 3) * NUMT * sizeof(double));
ivw2 = (int *)malloc(LMMAX * 3 * NUMT * sizeof(int));
if (vw2 == NULL || ivw2 == NULL)
  exit(-1);

printf("  *** SPARSE LINEAR EQUATIONS BY IDR METHOD");
printf(" WITH PRECONDITIONING\n");
printf("  *** COMPRESSED ROW STORAGE.\n");
printf("\n");

for (i = 0; i < N; i++)
  solex[i] = 1.0;

printf("  *** EXPECTED SOLUTIONS\n");
printf("  X(1) = %18.15lf  X(N) = %18.15lf\n", solex[0], solex[N-1]);
printf("\n");

va1 = 3.0;
va2 = 1.0/3.0;
va3 = 5.0;
vc = 1.0;
xl = 1.0;
yl = 1.0;
zl = 1.0;

init_mat_diag(va1, va2, va3, vc, (double *)diag, nofst,
              NX, NY, NZ, xl, yl, zl, NDIAG, N, K);

for (i = 0; i < NDIAG; i++) {
  if (nofst[i] < 0) {
    nbase = -nofst[i];
    length = N - nbase;
    for (j = 0,l = nbase; j < length; j++,l++)
      diag2[i][j] = diag[i][l];
  } else {
    nbase = nofst[i];
    length = N - nbase;
    for (j = 0,l = nbase; j < length; j++,l++)
      diag2[i][l] = diag[i][j];
  }
}

numnz = 1;

for (j = 0; j < N; j++) {
  ntopcfg = 1;
  for (i = NDIAG; i > 0; i--) {
    if (diag2[i-1][j] != 0.0) {
      ncoll = (j+1) - nofst[i-1];
      a[numnz-1] = diag2[i-1][j];
      nrow[numnz-1] = ncoll;
      if (ntopcfg == 1) {
        nfcnz[j] = numnz;
        ntopcfg = 0;
      }
      numnz++;
    }
  }
}

nfcnz[N] = numnz;
nnz = numnz - 1;
c_dm_vmvscc(a, nnz, nrow, nfcnz, N, solex, b, w, (int *)iw, &icon);
err1 = errnrm(solex, x, N);

for (i = 0; i < N; i++)
  x[i] = 0.0;
c_dm_vmvscc(a, nnz, nrow, nfcnz, N, x, y, w, (int *)iw, &icon);
err2 = errnrm(y, b, N);

aa    = (double *)malloc(sizeof(double) * nnz);
am    = (double *)malloc(sizeof(double) * nnz);
vw1   = (double *)malloc(sizeof(double) * nnz);
ncol  = (int *)malloc(sizeof(int) * nnz);
ncolm = (int *)malloc(sizeof(int) * nnz);
ivw1  = (int *)malloc(sizeof(int) * nnz);
if (aa == NULL || am == NULL || vw1 == NULL ||
```

```
    ncol == NULL || ncolm == NULL || ivw1 == NULL)
    exit(-1);
  isw = 1;
  for (i = 0; i < 20; i++) {
    ipar[i] = 0;
    rpar[i] = 0.0;
  }
  nwm = nnz;
  nzm = 0;

  convgcr(a, N, nfcnz, nrow, aa, nfrnz, ncol, ivw);
  c_dm_vlspaxcr2(aa, nnz, ncol, nfrnz, N, b, isw, x,
                 am, &nzm, ncolm, nfrnzm, nwm, ipar, rpar,
                 vw1, ivw1, vw2, ivw2, LMMAX, LNMAX, NUMT, &icon);

  eps = rpar[1];
  itmax = 2000;
  err3 = errnrm(solex, x, N);
  c_dm_vmvscc(a, nnz, nrow, nfcnz, N, x, y, w, (int *)iw, &icont);
  err4 = errnrm(y, b, N);
  printf("  *** COMPUTED SOLUTIONS\n");
  printf("  X(1) = %19.16lf  X(N) = %19.16lf\n", x[0], x[N-1]);
  printf("\n");
  printf("  C_DM_VLSPAXCR2 ICON =  %d\n", icon);
  printf("\n");
  printf("  N        =  %d\n", N);
  printf("        NX =  %d\n", NX);
  printf("        NY =  %d\n",NY);
  printf("        NZ =  %d\n", NZ);
  printf("  ITER MAX =  %d\n", itmax);
  printf("  ITER     =  %d\n", ipar[6]);
  printf("  ICMAV    =  %d\n", ipar[7]);
  printf("\n");
  printf("  EPS      =  %21.15le\n", rpar[1]);
  printf("\n");
  printf("  INITIAL ERROR           = %18.13lf\n", err1);
  printf("  INITIAL RESIDUAL ERROR  = %18.10lf\n", err2);
  printf("  CRITERIA RESIDUAL ERROR =  %20.15le\n", err2*eps);
  printf("\n");
  printf("  ERROR                   =  %20.15le\n", err3);
  printf("  RESIDUAL ERROR          =  %20.15le\n", err4);
  printf("\n");
  printf("\n");
  if (err4 <= err2*eps*1.1 && icon == 0) {
    printf(" ********** OK **********\n");
  } else {
    printf(" ********** NG **********\n");
  }
  free(vw2);
  free(ivw2);
  free(aa);
  free(am);
  free(vw1);
  free(ncol);
  free(ncolm);
  free(ivw1);
  return(0);
}

/* ========================================
       ABSOLUTE ERROR : | X1 - X2 |
   ======================================== */
double errnrm(double *x1, double *x2, int len) {
  int  i;
  double  s, ss, errnrm_ret;

  s = 0;
  for (i = 0; i < len; i++) {
    ss = x1[i] - x2[i];
    s = s + ss * ss;
  }
  errnrm_ret = sqrt(s);
  return(errnrm_ret);
}

/* ========================================
   INITIALIZE COEFFICIENT MATRIX
```

```
   ======================================== */
void init_mat_diag(double va1, double va2, double va3, double vc,
                   double *d_l, int *offset, int nx, int ny, int nz,
                   double xl, double yl, double zl, int ndiag, int len,
                   int ndivp) {

  if (ndiag < 1) {
    printf("FUNCTION INIT_MAT_DIAG:\n");
    printf(" NDIAG SHOULD BE GREATER THAN OR EQUAL TO 1\n");
    return;
  }
#pragma omp parallel default(shared)
{
  int  j, l, ndiag_loc, nxy, js, i0, j0, k0;
  int  i;
  double  hx, hy, hz, hx2, hy2, hz2;
/* NDIAG CANNOT BE GREATER THAN 7 */
  ndiag_loc = ndiag;
  if (ndiag > 7)
    ndiag_loc = 7;
/* INITIAL SETTING */
  hx = xl / (nx + 1);
  hy = yl / (ny + 1);
  hz = zl / (nz + 1);
#pragma omp for
  for (i = 0; i < ndivp; i++) {
    for (j = 0; j < ndiag; j++) {
      d_l[(j * ndivp) + i] = 0.0;
    }
  }
  nxy = nx * ny;
/* OFFSET SETTING */
#pragma omp single
  {
  l = 0;
  if (ndiag_loc >= 7) {
    offset[l]  = -nxy;
    l++;
  }
  if (ndiag_loc >= 5) {
    offset[l] = -nx;
    l++;
  }
  if (ndiag_loc >= 3) {
    offset[l] = -1;
    l++;
  }
  offset[l] = 0;
  l++;
  if (ndiag_loc >= 2) {
    offset[l] = 1;
    l++;
  }
  if (ndiag_loc >= 4) {
    offset[l] = nx;
    l++;
  }
  if (ndiag_loc >= 6) {
    offset[l] = nxy;
  }
  }
/* MAIN LOOP */
#pragma omp for
  for (j = 1; j <= len; j++) {
    js = j;
/* DECOMPOSE JS-1 = (K0-1)*NX*NY+(J0-1)*NX+I0-1 */
    k0 = (js - 1) / nxy + 1;
    if (k0 > nz) {
      printf("ERROR; K0.GH.NZ \n");
      continue;
    }
    j0 = (js - 1 - nxy * (k0 - 1)) / nx + 1;
    i0 = js - nxy * (k0 - 1) - nx * (j0 - 1);
    l = 0;
    if (ndiag_loc >= 7) {
      if (k0 > 1)
        d_l[(l * ndivp) + (j-1)] = -(1.0 / hz + 0.5 * va3) / hz;
```

```
      l++;
    }
    if (ndiag_loc >= 5) {
      if (j0 > 1)
        d_l[(l * ndivp) + (j-1)] = -(1.0 / hy + 0.5 * va2) / hy;
      l++;
    }
    if (ndiag_loc >= 3) {
      if (i0 > 1)
        d_l[(l * ndivp) + (j-1)] = -(1.0 / hx + 0.5 * va1) / hx;
      l++;
    }
    hx2 = hx * hx;
    hy2 = hy * hy;
    hz2 = hz * hz;
    d_l[(l * ndivp) + (j-1)] = 2.0 / hx2 + vc;
    if (ndiag_loc >= 5) {
      d_l[(l * ndivp) + (j-1)] += 2.0 / hy2;
      if (ndiag_loc >= 7) {
        d_l[(l * ndivp) + (j-1)] += 2.0 / hz2;
      }
    }
    l++;
    if (ndiag_loc >= 2) {
      if (i0 < nx)
        d_l[(l * ndivp) + (j-1)] = -(1.0 / hx -0.5 * va1) / hx;
      l++;
    }
    if (ndiag_loc >= 4) {
      if (j0 < ny)
        d_l[(l * ndivp) + (j-1)] = -(1.0 / hy - 0.5 * va2) / hy;
      l++;
    }
    if (ndiag_loc >= 6) {
      if (k0 < nz)
        d_l[(l * ndivp) + (j-1)] = -(1.0 / hz - 0.5 * va3) / hz;
    }
  }
}
  return;
}

/* ============================================================
      MODE CONV UNSYM MATRIX FROM COMPRESSED COLUMN TO ROW.
   ============================================================ */
void convgcr(double *ac, int n, int *ic, int *jc, double *ar,
  int *ir, int *jr, int *iw) {
  int     j, icol, nz;
  int  i;


  nz = ic[n] - 1;
  for (i = 0; i <= n; i++) {
    ir[i] = 0;
  }
  for (j = 0; j < nz; j++) {
    ir[jc[j]] = ir[jc[j]]+1;
  }
  ir[0] = 1;
  for (i = 1; i <= n; i++) {
    ir[i] = ir[i] + ir[i-1];
  }
  for (i=0; i < n; i++) {
    iw[i] = ir[i];
  }
  icol = 1;
  for (j = 0; j < nz; j++) {
```

```
      if (j == ic[icol]-1)
        icol++;
      jr[iw[jc[j]-1]-1] = icol;
      ar[iw[jc[j]-1]-1] = ac[j];
      iw[jc[j]-1] = iw[jc[j]-1] + 1;
    }
    return;
  }
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VLSPAXCR2 の項目及び [29], [31], [73]を参照してください.

c_dm_vlsx

正値対称行列の連立 1 次方程式 (ブロック化された変形コレスキー分解法)
`ierr = c_dm_vlsx(a, k, n, b, epsz, isw,` `&icon);`

1. 機能

実係数連立 1 次方程式をブロック化された外積形の変形コレスキー分解法で解きます.

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

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 次元の解ベクトルです. ($n \geq 1$)

2. 引数

呼出し形式:

```
ierr = c_dm_vlsx((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 ≤ j に格納します. 演算後, 内容は保存されません. 格納方法の詳細については, 図 c_dm_vlsx-1.を参照してください.
		出力	LDL^T 分解された行列 **L**, $\mathbf{D}^{-1}$ および $\mathbf{L}^T$ が, a の上三角部分 a[i−1][j−1], i ≤ j に格納されます.
k	int	入力	配列 a の整合寸法 (≥ 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ます. ("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 のとおり指定します. isw = 1 のとき, 1 組目の方程式を解きます. isw = 2 のとき, 2 組目以降の方程式を解きます. ただし, このとき b 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で使用します. ("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.

コード	意味	処理内容
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 ≠ 1 ,2 • k < n	

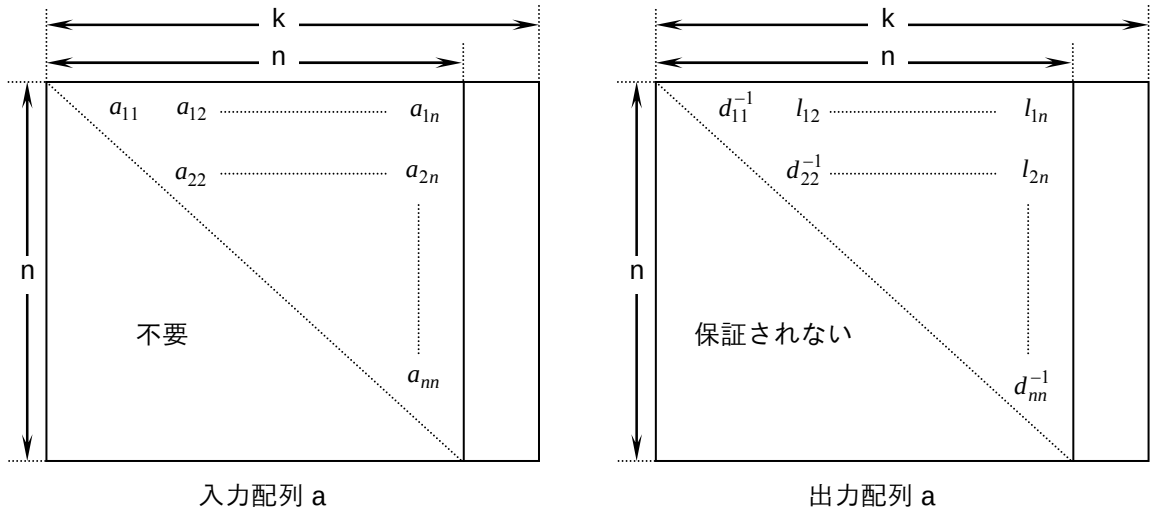


図 c_dm_vlsx-1. コレスキー分解法のデータの格納方法

LDL^{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, ... , n, j=i, ... , n に格納します. LDL^{T}分解された結果は, 行列 $\mathbf{D}^{-1}$ を対角部分に, **L** の対角要素を除いた部分を上三角部分にそれぞれ格納されます. 下三角部分の値は保証されません.

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^{-s} を設定したとすると, この値は次の意味を持っています. 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, icon = 20000 として処理を打ち切ります. epsz の標準値は丸め誤差の単位をμとしたとき, epsz = 16μです.

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 (例えば 10^{-70})を与えればいいのですが, その結果は保証されません.

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 LDL^{T}分解過程を省略するので計算時間が少なくなります.

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ります. このとき, icon = 10000 としますが処理は続行します. ただし, ピボッティ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す.

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 個の対角線上の値($\mathbf{D}^{-1}$ の対角要素)を掛け合わせ, その逆数をとって下さい.

4. 使用例

1000×1000 の係数行列の連立 1 次方程式を解きます.

```
#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 NMAX      (1000)
#define LDA       (NMAX+1)

MAIN__()
{
  int    n, isw, i, j, icon, ierr;
  double a[NMAX][LDA], b[NMAX];
  double epsz, s, det;

  n    = NMAX;
  epsz = 0.0;
  isw  = 1;

#pragma omp parallel for shared(a,n) private(i,j)
  for(i=0; i<n; i++)
    for(j=0; j<n; j++)
      a[i][j] = min(i,j)+1;

#pragma omp parallel for shared(b,n) private(i)
  for(i=0; i<n; i++) b[i] = (i+1)*(i+2)/2+(i+1)*(n-i-1);

  ierr = c_dm_vlsx((double*)a, LDA, n, b, epsz, isw, &icon);

  if (icon != 0) {
    printf("ERROR: c_dm_vlsx failed with icon = %d\n", icon);
    exit(1);
  }

  s = 1.0;
#pragma omp parallel for shared(a,n) private(i) reduction(*:s)
  for(i=0; i<n; i++) s *= a[i][i];

  printf("solution vector:\n");
  for(i=0; i<10; i++) printf("    b[%d] = %e\n", i, b[i]);

  det = 1.0/s;
  printf("\ndeterminant of the matrix = %e\n", det);
  return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VLSX の項目及び[30], [54], [70]を参照してください.

c_dm_vlux

LU 分解された実行列の連立 1 次方程式
`ierr = c_dm_vlux(b, fa, kfa, n, 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

$$\mathbf{LUx} = \mathbf{Pb} \tag{1}$$

を解きます. 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, **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います), **b** は n 次元の実定数ベクトル, **x** は n 次元の解ベクトルです. ($n \geq 1$)

2. 引数

呼出し形式:

```
ierr = c_dm_vlux(b, (double*)fa, kfa, n, ip, &icon);
```

引数の説明:

`b`	`double b[n]`	入力	定数ベクトル **b**.
		出力	解ベクトル **x**.
`fa`	`double fa[n][kfa]`	入力	行列 **L** + (**U** − **I**). ("使用上の注意" a)参照)
`kfa`	`int`	入力	配列 `fa` の整合寸法 (≥ `n`).
`n`	`int`	入力	行列 **L**, **U** の次数 n.
`ip`	`int ip[n]`	出力	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" 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.
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` • `ip` に誤りがあった	

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_dm_valu を呼び出して, 係数行列を LU 分解してから, 本関数を呼び出せば方程式を解くことができます. このとき入力引数 `fa` と `ip` は c_dm_valu の `a` と `ip` にそれぞれ対応しています. しかし, 通常は関数 c_dm_vlax を呼び出せば一度に解が求まります.

4. 使用例

1000 × 1000 の係数行列を LU 分解してから, 連立 1 次方程式を解きます.

```
#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 NMAX      (1000)
#define LDA       (NMAX+1)

MAIN__()
{
  int    n, is, isw, i, j, icon, ierr;
  int    ip[NMAX];
  double a[NMAX][LDA], b[NMAX];
  double epsz, s, det;

  n    = NMAX;
  epsz = 0.0;
  isw  = 1;

#pragma omp parallel for shared(a,n) private(i,j)
  for(i=0; i<n; i++)
    for(j=0; j<n; j++) a[i][j] = min(i,j)+1;

#pragma omp parallel for shared(b,n) private(i)
  for(i=0; i<n; i++) b[i] = (i+1)*(i+2)/2+(i+1)*(n-i-1);

  ierr = c_dm_valu((double*)a, LDA, n, epsz, ip, &is, &icon);

  if (icon != 0) {
    printf("ERROR: c_dm_valu failed with icon = %d\n", icon);
    exit(1);
  }

  ierr = c_dm_vlux(b, (double*)a, LDA, n, ip, &icon);

  if (icon != 0) {
    printf("ERROR: c_dm_vlux failed with icon = %d\n", icon);
    exit(1);
  }

  s = 1.0;
#pragma omp parallel for shared(a,n) private(i) reduction(*:s)
  for(i=0; i<n; i++) s *= a[i][i];

  printf("solution vector:\n");
  for(i=0; i<10; i++) printf("    b[%d] = %e\n", i, b[i]);

  det = is*s;
  printf("\ndeterminant of the matrix = %e\n", det);
  return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VLUX の項目を参照してください.

c_dm_vmggm

行列の積 (実行列)
`ierr = c_dm_vmggm(a, ka, b, kb, c, kc, m, n, l, &icon);`

1. 機能

$m \times n$ の実行列 **A** と, $n \times l$ の実行列 **B** の積 **C** を求めます.

$$\mathbf{C} = \mathbf{AB}$$

ここで **C** は, $m \times l$ の実行列です. ($m, n, l \geq 1$)

2. 引数

呼出し形式:

```
ierr = c_dm_vmggm((double*)a, ka, (double*)b, kb, (double*)c, kc, m, n, l,
            &icon);
```

引数の説明:

a	double a[m][ka]	入力	行列 **A**.
ka	int	入力	配列 a の整合寸法 ($\geq n$).
b	double b[n][kb]	入力	行列 **B**.
kb	int	入力	配列 b の整合寸法 ($\geq l$).
c	double c[m][kc]	出力	行列 **C** ("使用上の注意" a)参照).
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ません. 必ず, 行列 **A**, **B**, **C** は別々の領域を確保してください.

4. 使用例

行列積を計算し, 結果を確認します.

```
#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_dm_vmggm((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);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VMGGM の項目及び[30]を参照してください.

c_dm_vminv

実行列の逆行列 (ブロック化された Gauss-Jordan 法)
`ierr = c_dm_vminv(a, k, n, epsz, &icon);`

1. 機能

Gauss-Jordan 法により正則な $n \times n$ 実行列 **A** の逆行列 $\mathbf{A}^{-1}$ を求めます.

2. 引数

呼出し形式:

```
ierr = c_dm_vminv((double*)a, k, n, epsz, &icon);
```

引数の説明:

a	double	入力	行列 **A**.
	a[n][k]	出力	行列 $\mathbf{A}^{-1}$.
k	int	入力	配列 a の整合寸法 (≥ 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 が入力されたときは, 標準値が採用されます. ("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.
20000	行列 **A** のある行の要素がすべて零であったか, またはピボットが相対的に零となった. 行列 **A** は非正則である可能性が強い.	処理を打ち切る.
30000	次のいずれかであった. • n < 1 • k < n • epsz < 0.0	

3. 使用上の注意

a) epsz について

部分ピボッティングで選択されたピボット要素が 0.0 か絶対値が epsz 以下なら, 相対的に零とみなして, icon = 20000 として処理を打ち切ります. epsz の標準値は丸め誤差の単位を *u* としたとき, 16*u* です. epsz に極小の値を設定すると, ピボットの絶対値が小さくなっても処理は続行されますが, 計算結果の精度は保証されません.

4. 使用例

逆行列を求めます.

```
#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 N 2000
#define K (N+1)

int MAIN__()
{
  double a[N][K], as[N][K];
  double c, t, error, epsz;
  int    i, j, icon;

  c = sqrt(2.0/(N+1));
  t = atan(1.0)*4.0/(N+1);

  for (j=0; j<N; j++) {
    for (i=0; i<N; i++) {
      as[j][i] = a[j][i] = c*sin(t*(i+1)*(j+1));
    }
  }

  epsz = 0.0;
  c_dm_vminv((double*)a, K, N, epsz, &icon);

  error = 0.0;
  for (i=0; i<N; i++) {
    for (j=0; j<N; ++j) {
      error = max(error,fabs(a[j][i]-as[j][i]));
    }
  }

  printf("order = %d, error = %e\n", N, error);
  return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VMINV の項目を参照してください.

c_dm_vmlbife

スパース行列の連立 1 次方程式 (不完全ブロック分解に基づくマルチレベル前処理付き反復法, ELLPACK 形式格納法)
`ierr = c_dm_vmlbife(a, k, iwidt, n, icol, b, isw, iguss, info, infoep, epsot, epsin, epsep, x, w, nw, iw, niw, &icon);`

1. 機能

$n \times n$ のスパース行列を係数行列とする連立 1 次方程式を反復法で解きます.

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

$n \times n$ の係数行列 **A** は, 対角形式の格納法で格納します. ベクトル **b** および **x** は n 次元ベクトルです.

反復法は行列 **A** が対称のときは, ORTHOMIN 法が, 非対称のときは, GMRES 法が使用できます. この反復解法(以下外部反復と呼びます)にはマルチレベル構造を持つ不完全ブロック分解による前処理が施されていて, 安定な解法です. 各レベルである変数の集合に対して消去を行い, その過程で消去される変数の集合からなる部分行列に対する近似的逆行列を利用します.

消去の手続きは, いわゆる最も粗いレベルの小さな連立 1 次方程式が生成されるまで続きます. 外部反復の各ステップに対して, 最も粗いレベルの連立 1 次方程式は反復法(内部反復)によって解かれます.

2. 引数

呼出し形式:

```
ierr = c_dm_vmlbife((double*)a, k, iwidt, n, (int*)icol, b, isw, iguss, info,
            infoep, epsot, epsin, epsep, x, w, nw, iw, niw, &icon);
```

引数の説明:

a	double a[iwidt][k]	入力	係数行列 **A** の非零要素を ELLPACK 形式で格納します.
k	int	入力	配列 a の整合寸法 (≥ 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.
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かを示します.
b	double b[k]	入力	連立 1 次方程式の右辺の定数ベクトルを格納します.
isw	int	入力	制御情報. 1 初回の呼び出し. 2 2 回目以降の呼び出し. このとき, a, icol, iw, w の 値は, 初回の呼び出しの結果を変更せずに呼び出す必要があります. (“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.

			iguss = 0　解ベクトルの近似値を指定しません. このときゼロベクトルから反復を開始します.
			iguss ≠ 0　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します.
info	int info[14]	入出力	反復に関する制御情報.

[行列 **A** が対称な場合の指定例]

```
info[0] = 10;   info[1] = NTHRD*100; info[2] = 0;
info[4] = 1;    info[5] = 2000;      info[9] = 1;
info[10]= 1000;
```

[行列 **A** が非対称な場合の指定例]

```
info[0] = 10;   info[1] = NTHRD*100; info[2] = 0;
info[4] = 2;    info[5] = 2000;      info[6] = 5;
info[7] = 20;   info[9] = 2;         info[10]= 1000;
info[11]= 10;   info[12]= 0;
```

NTHRD は, 並列実行するスレッドの数.

info[0]	入力	MAXLVL. 代数的マルチレベル法のレベルの最大値. MAXLVL < 1 のとき, 前処理は施されません. MAXLVL > 0 のとき, 指定された値以上の粗いレベルは使われません. (“使用上の注意” e), h)参照)
info[1]	入力	MINUK. もっとも深いレベルの縮小された連立 1 次方程式の未知数の最小個数. MINUK は n よりずっと小さく, 並列処理するスレッドの数 NTHRD よりずっと大きな値を指定することを勧めます. 例えば, NTHRD × 100.
info[2]	入力	NORM. 行列の正規化の方法を指定します. NORM < 1 のとき, 行列 **A** は **A** の対角要素の平方根の逆数を対角要素として持つ対角行列で左右から正規化されます. 対称行列を正規化した行列は対称行列です. 対称な正規化を勧めます. しかし, ある場合は, 非対称な正規化の方が速く収束することもあり得ます. NORM ≥ 1 のとき, 行列 **A** は **A** の主対角要素と同じ符号とした行の絶対値の和の逆数からなる対角行列で左から正規化します. **A** が対称行列でも正規化されたあとは対称行列ではありません. (“使用上の注意” c), d)参照)
info[3]	出力	使用されたレベル数.
info[4]	入力	METHOT. 外部反復で使われる反復法. METHOT = 1 のとき, 前処理付き ORTHOMIN 法が使われます. 行列 **A** が対

称で対称な正規化を指定した場合に指定してください.
METHOT ≠ 1 のとき, 切り捨て再起動する GMRES 法が使われます. 行列 **A** が非対称であるか, 非対称な正規化が使われた場合に指定してください.

`info[5]` 入力 ITMXOT.
外部反復の最大回数. 例えば, 2000.
外部反復回数がこの値に達したとき, 処理を打ち切ります. このとき返却された解は, 保証されません.

`info[6]` 入力 NRESOT.
外部反復で使われる反復法に GMRES 法を指定したとき, 外部反復の直交手続きで使う残差ベクトルの本数を指定します. 例えば 5. つまり GMRES 法で NRESOT 個のベクトルを使い解を更新し, 残りのベクトルによる更新はされません.
(“使用上の注意” d)参照)

`info[7]` 入力 NRSTOT.
外部反復で使われる反復法に GMRES 法を指定したとき, 再起動(restart)する反復回数を指定します. NRSTOT ≥ NRESOT.
例えば 20 回.
NRSTOT < 1 のとき, 再起動は行いません.
(“使用上の注意” d)参照)

`info[8]` 出力 ITEROT.
外部反復の回数.

`info[9]` 入力 METHIN.
内部反復で使われる反復法.
METHIN = 1 のとき, 前処理付き ORTHOMIN 法が使われます. 行列 **A** が対称で対称な正規化を指定した場合に指定してください.
METHIN ≠ 1 のとき, 切り捨て再起動する GMRES 法が使われます. 行列 **A** が非対称であるか, 非対称な正規化が使われた場合に指定してください.

`info[10]` 入力 ITMXIN.
内部反復の最大回数. 例えば, 1000.
内部反復回数がこの値に達すると, 外部反復に処理は引き継がれます.

`info[11]` 入力 NRESIN.
内部反復で使われる反復法に GMRES 法を指定したとき, 内部反復の直交手続きで計算する残差ベクトルの数を指定します. 例えば 10. つまり GMRES 法で NRESOT 個のベクトルを使い解を更新し, 残りのベクトルによる更新はされません.

(“使用上の注意” d)参照)

`info[12]` 入力 NRSTIN.
内部反復で使われる反復法に GMRES 法を指定したとき, 再起動(restart)する反復回数を指定します. NRSTIN ≥ NRESIN.
NRSTIN < 1 のとき, 再起動は行いません.
(“使用上の注意” d)参照)

`info[13]` 出力 内部反復の平均回数.

`infoep` `int infoep[3]` 入力 消去する変数の部分行列及び縮小された行列の構成に関する制御情報.

`infoep` の代表的な指定例.

各レベルで, Schur の complement の計算などで使う部分行列の逆行列を近似的に求める方法を指定します.
(“使用上の注意” g)参照)

[逆行列を対角行列で近似する]

```
infoep[0] = 1;
infoep[1] = 5;
infoep[2] = 2*nrow;
```

[逆行列を反復法で近似的に求める]

```
infoep[0] = nrow;
infoep[1] = 5;
infoep[2] = 2*nrow;
```

`nrow` =係数行列 **A** の行の非ゼロ要素の代表的な数.

`infoep[0]` 入力 MAXNCV.
ブロック分解で消去する変数よりなる部分行列の逆行列の近似行列を構成するとき, 各行ベクトルに存在する非ゼロ要素数の最大個数.
典型的な設定は, MAXNCV = 1 または, MAXNCV = MAXNC です.
MAXNCV = 1 のときは, 逆行列は対角要素で近似されます.

`infoep[1]` 入力 MAXITV.
近似的に逆行列を計算する最大ステップ数.
MAXITV は, 精度 TAUV での近似的な逆行列を計算するための最大反復ステップを示します. ステップ数が MAXITV に達したとき, 反復は打ち切られます.
いかなる場合も, 近似には $\frac{\log(\mathrm{TAUV})}{\log(\mathrm{LAMBDA})}$ 以下のステップが必要と考えられます.
MAXITV ≤ 1 のとき, 逆行列は主対角要素のみで近似されます.

`infoep[2]` 入力 MAXNC.
ブロック分解の後, Schur の complement として縮小された行列に残る要素を制御します.
MAXNC < 2 のとき, TAU よりも小さな要

素は除去されます.
MAXNC > 1 のとき, 各行ベクトルに存在する要素の数は MAXNC により制御されます. 最も大きな MAXNC 個の要素が残ります. 他の要素は TAU より大きくても除去されます.

epsot	double	入力	解の収束判定値. 外部反復の k ステップで求められた解ベクトルが正規化された残差ベクトル $\hat{\mathbf{r}}_k = \hat{\mathbf{A}}\mathbf{x}_k - \hat{\mathbf{b}}_k$ に対して以下の条件を満たしたときベクトルは収束したと見なします. $\|\hat{\mathbf{r}}_k\| \le \mathtt{epsot}\|\hat{\mathbf{b}}\|$, $\|\mathbf{y}\|^2 = \mathbf{y}^{\mathrm{T}}\mathbf{y}$ なるユークリッドノルムであり, $\hat{\mathbf{A}}$ および $\hat{\mathbf{b}}$ は正規化された係数行列および右辺ベクトルです. (“使用上の注意” h)参照)
epsin	double	入力	内部反復での解の収束判定値. たいていの場合 10^{-3} が最適です.

epsep	double epsep[4]	入力	縮小行列の近似及び逆行列の近似の制御に関係する制御情報.
			epsep の代表的な指定例
			`epsep[0] = 1.0e-2;`
			`epsep[1] = 1.0e-2;`
			`epsep[2] = 0.2;`
			`epsep[3] = 1.0e-3;`

	epsep[0]	入力	TAU. ブロック分解の後 Shur の complement として縮小された行列が疎行列であることを保つために, TAU より小さな要素は除去します. TAU を大きくすると, 低いレベルの反復法は速く収束しますが, 前処理の品質は悪くなります. $0 \le \mathrm{TAU} < 1$
	epsep[1]	入力	TAUV. 近似的逆行列の精度. TAUV を小さくすると消去にかかる時間は増しますが, 前処理の品質は向上します. 普通は, epsin = TAUV が最適です.
	epsep[2]	入力	LAMBDA. ブロック行列に対する対角しきい値. ブロック行列の中の除去する変数は, 行の各要素の絶対値の和が, 対角要素の絶対値 × LAMBDA 以下となるように選ばれます. LAMBDA を大きく指定すると, 除去する変数の集合は小さくなりますが, ブロック行列の近似的な逆行列を計算するコストは増加します. LAMBDA としては, 0.2 を推奨します. $\mathrm{TAUV} \le \mathrm{LAMBDA} < 1$ か $\mathrm{LAMBDA} = 0$ とすべきです.
	epsep[3]	入力	RHO. 消去の過程で, 主対角要素が小さな値であ

			る変数は, 消去する変数から外します. 主対角要素は, 行の要素の絶対値の和に RHO を掛けたものより小さいとき小さいと考えます. RHO = 10^{-3} を推奨します. $0 < \mathrm{RHO} < 1$
x	double x[n]	出力	解ベクトルが格納されます.
w	double w[nw]	作業領域	
nw	int	入力	作業用配列 w の大きさ. nw の大きさの上限目安は次のとおり. $\mathtt{nw} \le \max(2 \times \mathrm{MAXLVL} + 2, 10) \times \mathtt{NBAND} \times \mathtt{MAXT} + (4 \times \mathtt{NC} + 6) \times (\mathtt{n} + \mathtt{MAXT}) + \max(2 \times \mathtt{NC} \times (\mathtt{n} + \mathtt{MAXT}), \mathtt{LR0(n)}) + \max(\mathtt{LR0(nf)} + \mathtt{n} + \mathtt{MAXT}, 6 \times (\mathtt{n} + \mathtt{MAXT}))$ NBAND は, 下バンド巾と上バンド巾の最大値. NC は, 行での非ゼロ要素の数の上限. 典型的には, NC = MAXNC. nf は, 最後のレベルの変数の数(典型的には, $2^{-\mathrm{MAXLVL}} \times (\mathtt{n} + \mathtt{MAXT})$). MAXT は, 本関数を並列実行する最大スレッド数. $\mathtt{LR0}(\mathrm{N}) = \begin{cases} 4 \times \mathrm{N} & : \text{ORTHOMIN法のとき} \\ (2 \times \mathtt{NRES} + 1) \times \mathrm{N} & : \text{GMRES法のとき} \end{cases}$ NRES は GMRES 法で使う残差の数. 普通は LR0(nf)は, 無視できます.
iw	int iw[niw]	作業領域	
niw	int	入力	作業用配列 iw の大きさ. 大きさの上限目安は次のとおり. $\mathtt{niw} \le ((4 \times \mathrm{MAXLVL} + 10) \times \mathtt{MAXT} + 12 \times \mathtt{NBAND}) + 3400) \times \mathtt{MAXT} + (6 \times \mathtt{NC} + 11) \times (\mathrm{n} + \mathtt{MAXT})$ NBAND は, 下バンド巾と上バンド巾の最大値. NC は, 行での非ゼロ要素の数の上限. 典型的には, NC = MAXNC. MAXT は, 本関数を並列実行する最大スレッド数.
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.
10100	逆行列が十分な精度で計算できなかった.	処理を続ける.
10800	GMRES 法で回復可能な break down を起こした.	
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.
20003	GMRES 法で回復不可能な break down を起こした.	処理を打ち切る.
20005	ORTHOMIN 法で $\mathbf{p}^{\mathrm{T}} \mathbf{A}\, \mathbf{p} = 0$ となり回復不可能な break down を起こした.	
20006	ORTHOMIN 法で $\mathbf{p}^{\mathrm{T}} \mathbf{r} = 0$ となり回復不可能な break down を起こした.	

コード	意味	処理内容
30000	次のいずれかであった. • n < 1 • n > k • iwidt < 1 • isw ≠ 1, 2	
30103	icol の値が正しくない.	
30105	主対角要素がない.	
30210	行列の圧縮が非正値であるためできない.	
30213	非ゼロ要素のない行がある.	
30310	niw が小さ過ぎる.	
30320	nw が小さ過ぎる.	

3. 使用上の注意

a) isw について

同じ係数行列を持つ右辺の異なる多重組みの連立 1 次方程式を解く場合は, 最初の呼び出しで isw = 1 で呼び出して解き, 2 回目以降の呼び出しでは isw = 2 として呼び出すことで解けます. 最初の呼び出しで組み立てられた粗いレベルに対応する行列が 2 回目以降の呼び出しで再利用されます.

b) nw 及び niw について

nw および niw の大きさで, たいていの場合 NC は iwidt × 1.5 で十分です. 一般的に, 作業域が足りないときは, NC を増やすことを奨めます. もし係数行列のバンド巾が非常に大きいなら, NBAND を最初に増やすことを奨めます.

c) ORTHOMIN 法について

できる限り ORTHOMIN 法を利用することを奨めます. これは, 行列が対称である必要があります.

たとえ与えられた行列が対称でなくても, 本ルーチンは反復を開始する前に計算できる変数を簡単に除去するので, 実際の反復計算で使う行列は対称となることがあります. そのため, 反復計算に使う行列が対称となる可能性があるときは, 最初に対称的正規化を行う ORTHOMIN 法を試みることを奨めます.

d) GMRES 法について

行列が対称でない場合は, 非対称な正規化を行って GMRES 法を使うことを奨めます. たいていの場合, 外部反復で NRESOT = 5 として切り捨て 20 ステップ後に再起動すれば十分です. 内部反復では, NRESIN として切り捨てる数をもっと大きくし, もっと大きなステップの後に再起動するか再起動を禁じることが必要です. NRESIN を大きくすると作業域を大きくする必要があります. このため作業域の大きさの計算式の中の NRES を大きくしなければなりません. しかし NRES は NRESOT より小さくできます.

一般に, NRESOT および NRESIN を大きくすると GMRES 法の精度はよくなりますが, 計算およびメモリ使用量も増えます.

e) 変数消去について

変数消去は次の条件の 1 つが満たされたとき終了します.

- レベルの数が MAXLVL を超えるか等しくなった.
- 最後のレベルの行列が対角行列である.
- 消去する変数の数が最後のレベルの変数の 10%以下である.

f) `ILUM` 前処理

TAU = 0, LAMBDA = 0, RHO = 0.99, MAXNC = `iwidt` としたとき, 古典的な Wavefront ordering による ILUM 前処理になります.

TAU > 0, LAMBDA = 0, RHO < 1, MAXNC >> `iwidt` としたとき, しきい値付きの ILUM 前処理となります.

g) パラメタについて

問題に応じて効率のよい前処理を見つけるために, パラメタを変えて試みることを奨めます.

h) `Schur complement`

各レベルの行列を $\mathbf{A}_n$(n= 1, ... , MAXLVL–1)としたとき,

$$\mathbf{A}_n = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}$$

$\mathbf{S} = \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12}$ を Schur complement と呼びます. 消去する変数の集合を選ぶことでこのようなブロックを形成します. $\mathbf{A}_{11}$ の逆行列を近似します. この近似的逆行列を使って Schur complement を計算し, レベル n+1 の行列 $\mathbf{A}_{n+1}$ とします.

Schur complement を使い $\mathbf{A}_n$ は次のように分解できます.

$$\mathbf{A}_n = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{0} \\ \mathbf{A}_{21} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{A}_{11}^{-1}\mathbf{A}_{12} \\ \mathbf{0} & \mathbf{S} \end{bmatrix}$$

$\mathbf{A}_{11}^{-1}$ の近似を利用して, 各レベルをこのように近似的に分解し前処理として利用します.

4. 使用例

偏微分方程式を領域 $[0, 1]^2$ で離散化して解きます.

$$-\left(\frac{\partial^2}{\partial^2 x_1}u + \frac{\partial^2}{\partial^2 x_2}u + \frac{\partial^2}{\partial^2 x_3}u\right) + t\left((x_2 - x_3)\frac{\partial u}{\partial x_1} + (x_3 - x_1)\frac{\partial u}{\partial x_2} + (x_1 - x_2)\frac{\partial u}{\partial x_3}\right) = f$$

t = 1.0, Dirichlet 境界条件 u = 0 を設定します.

```
#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 MAXT   2
#define N1     39
#define N2     (N1)
#define N3     (N1)
#define L1     (N1)
#define L2     (N2)
#define L3     (N3)
#define KA     (N1*N2*N3)
#define NA     7
#define NLBMAX (N1*N2)
#define MAXNC  11
#define NW   ((KA+MAXT)*(6*MAXNC+11)+(85*NLBMAX+100)*MAXT)
#define NIW  ((KA+MAXT)*(6*MAXNC+11)+(13*NLBMAX+200+61*51+13)*MAXT)

int MAIN__()
{
  double a1[L3][L2][L1], a2[L3][L2][L1], a3[L3][L2][L1];
```

```
double b1[L3][L2][L1], b2[L3][L2][L1], b3[L3][L2][L1];
double x1[L1], x2[L2], x3[L3], c[L3][L2][L1], f[L3][L2][L1];
double w[NW], epsin, epsot, epsep[4], mat[NA][KA], rhs[KA], v[KA];
double sol[KA*3], rhsx[KA], rhsc[KA];
double tmp, t, hr1, hr2, hr3, hr4, hr6, hr7, hr13, one=1.0;
int    ndlt[NA], iw[NIW], info[14], infoep[3], icol[NA][KA];
int    isw, iguss, nband, ndiag, icon;
int    z, z1, z2, z3, n, i, nc;

/* THESE ARE PARAMETERS OF THE TEST PDES. CHANGES OF THE  */
/* VALUES CAN PRODUCE DIVERGENCE IN THE ITERATIVE SOLVER. */
t = 1.0;

/* CREATE NODE COORDINATES */
for (z1=0; z1<N1; z1++) {
  x1[z1] = (double)z1/(double)(N1-1);
}
for (z2=0; z2<N2; z2++) {
  x2[z2] = (double)z2/(double)(N2-1);
}
for (z3=0; z3<N3; z3++) {
  x3[z3] = (double)z3/(double)(N3-1);
}

/* -UX1X1-UX2X2-UX3X3+T*((X2-X3)*UX1+(X3-X1)*UX2+(X1-X2)*UX3)=F */
/*                                                             */
/*   REMARK: IF T IS TO LARGE THE PDE IS SINGULAR.             */
for (z3=0; z3<N3; z3++) {
  for (z2=0; z2<N2; z2++) {
    for (z1=0; z1<N1; z1++) {
      a1[z3][z2][z1] = 1.0;
      a2[z3][z2][z1] = 1.0;
      a3[z3][z2][z1] = 1.0;
      b1[z3][z2][z1] = t*(x2[z2]-x3[z3]);
      b2[z3][z2][z1] = t*(x3[z3]-x1[z1]);
      b3[z3][z2][z1] = t*(x1[z1]-x2[z2]);
      c[z3][z2][z1]  = 0.0;
      hr1            = one-x2[z2];
      hr2            = x2[z2]*hr1;
      hr3            = one-x3[z3];
      hr4            = x3[z3]*hr3;
      hr6            = one-x1[z1];
      hr7            = x1[z1]*hr6;
      hr13           = hr1*x3[z3]*hr3;
      f[z3][z2][z1]  = 2*hr2*hr4+2*hr7*hr4+2*hr7*hr2+
                       t*((x2[z2]-x3[z3])*(hr6*x2[z2]*hr13-x1[z1]*x2[z2]*hr13)
                         +(x3[z3]-x1[z1])*(hr7*hr13-hr7*x2[z2]*x3[z3]*hr3)
                         +(x1[z1]-x2[z2])*(hr7*hr2*hr3-hr7*hr2*x3[z3]));
    }
  }
}

/* DIRICHLET CONDITIONS: */
for (z3=0; z3<N3; z3++) {
  for (z2=0; z2<N2; z2++) {
    c[z3][z2][0]     = 1.0;
    b1[z3][z2][0]    = 0.0;
    b2[z3][z2][0]    = 0.0;
    b3[z3][z2][0]    = 0.0;
    f[z3][z2][0]     = 0.0;
    c[z3][z2][N1-1]  = 1.0;
    b1[z3][z2][N1-1] = 0.0;
    b2[z3][z2][N1-1] = 0.0;
    b3[z3][z2][N1-1] = 0.0;
    f[z3][z2][N1-1]  = 0.0;

    if (z2 == 0) {
      for (z1=0; z1<N1; z1++) {
        c[z3][0][z1]  = 1.0;
        b1[z3][0][z1] = 0.0;
        b2[z3][0][z1] = 0.0;
        b3[z3][0][z1] = 0.0;
        f[z3][0][z1]  = 0.0;
      }
    } else if (z2 == N2-1) {
      for (z1=0; z1<N1; z1++) {
        c[z3][N2-1][z1]  = 1.0;
```

```
            b1[z3][N2-1][z1] = 0.0;
            b2[z3][N2-1][z1] = 0.0;
            b3[z3][N2-1][z1] = 0.0;
            f[z3][N2-1][z1]  = 0.0;
          }
        }

        if (z3 == 0) {
          for (z1=0; z1<N1; z1++) {
            c[0][z2][z1]  = 1.0;
            b1[0][z2][z1] = 0.0;
            b2[0][z2][z1] = 0.0;
            b3[0][z2][z1] = 0.0;
            f[0][z2][z1]  = 0.0;
          }
        } else if (z3 == N3-1) {
          for (z1=0; z1<N1; z1++) {
            c[N3-1][z2][z1]  = 1.0;
            b1[N3-1][z2][z1] = 0.0;
            b2[N3-1][z2][z1] = 0.0;
            b3[N3-1][z2][z1] = 0.0;
            f[N3-1][z2][z1]  = 0.0;
          }
        }
      }
    }

    n = N1*N2*N3;
    c_dm_vpde3d((double*)a1, L1, L2, N1, N2, N3, (double*)a2, (double*)a3, x1, x2, x3,
                (double*)b1, (double*)b2, (double*)b3, (double*)c, (double*)f,
(double*)mat,
                 KA, NA, n, &ndiag, ndlt, rhs, &icon);
    printf("icon of c_dm_vpde3d = %d\n", icon);

    for (z =0; z<n; z++) {
      rhsx[z] = rhs[z];
    }

    nband = 0;
    for (i=0; i<ndiag; i++) {
      nband=max(nband,fabs(ndlt[i]));
    }

    /* CHANGE TO ELLPACK FORMAT: */
    nc = ndiag;
    for (i=0; i<nc; i++) {
      for (z=0; z<KA; z++) {
        icol[i][z] = z+ndlt[i]+1;
      }
    }

    /* CALL THE ITERATIVE SOLVER: */
    isw       = 1;
    iguss     = 0;
    epsot     = 1.0e-6;
    epsin     = 1.0e-3;
    info[0]   = 10;
    info[1]   = MAXT*100;
    info[2]   = 1;
    info[4]   = 2;
    info[5]   = 5000;
    info[6]   = 5;
    info[7]   = 20;
    info[9]   = 2;
    info[10]  = 5000;
    info[11]  = 20;
    info[12]  = 0;
    infoep[0] = 1;
    infoep[1] = 5;
    infoep[2] = 14;
    epsep[0]  = 1.0e-2;
    epsep[1]  = 1.0e-2;
    epsep[2]  = 0.2;
    epsep[3]  = 1.0e-3;

    c_dm_vmlbife((double*)mat, KA, nc, n, (int*)icol, rhs, isw, iguss, info,
                 infoep, epsot, epsin, epsep, v, w, NW, iw, NIW, &icon);
```

```
  printf("icon of c_dm_vmlbife = %d\n", icon);

  for (i=0; i<nband; i++) {
    sol[i]         = 0.0;
    sol[nband+n+i] = 0.0;
  }

  for (z=0; z<n; z++) {
    sol[nband+z] = v[z];
  }

  c_dm_vmvsd((double*)mat, KA, ndiag, n, ndlt, nband, sol, rhsc, &icon);

  tmp = 0.0;
  for (z=0; z<n; z++) {
    tmp = max(tmp,fabs((rhsx[z]-rhsc[z])/(rhsx[z]+1.0)));
  }

  printf("error = %e\n", tmp);
  return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VMLBIFE の項目を参照してください.

c_dm_vmvscc

実スパース行列と実ベクトルの積(圧縮列格納法)
`ierr = c_dm_vmvscc(a, nz, nrow, nfcnz, n, x, y, w, iw, &icon);`

1. 機能

$n \times n$ のスパース行列とベクトルの積を計算します.

$$\mathbf{y} = \mathbf{A}\mathbf{x}$$

スパース行列 **A** は, 圧縮列格納法(Compressed Column Storage)で格納します. ベクトル **x** 及び **y** は n 次元ベクトル.

2. 引数

呼出し形式:

```
ierr = c_dm_vmvscc(a, nz, nrow, nfcnz, n, x, y, w, (int*)iw, &icon);
```

引数の説明:

a	double a[nz]	入力	係数行列の非零要素を格納します. a[i], i=0,…,nz-1 にスパース行列の非零要素を格納します. 圧縮列格納法については, 図 c_dm_vmvscc-1 参照.
nz	int	入力	係数行列 **A** の非零要素の総数
nrow	int nrow[nz]	入力	圧縮列格納法で使用される行指標で, **A** に格納される要素が何番目の行ベクトルに属するかを示します.
nfcnz	int nfcnz[n+1]	入力	行列 **A** の各列の非零要素を列方向に圧縮して順次配列 **A** に格納するとき, 対応する列の最初の非零要素が格納される位置を表します. nfcnz[n]=nz+1
n	int	入力	行列 **A** の次数 n.
x	double x[n]	入力	ベクトル x を格納します.
y	double y[n]	出力	行列とベクトルの積の結果が格納されます.
w	double w[nz]	作業領域	
iw	int iw[nz][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. • n < 1 • nz < 0 • nfcnz[n]≠nz+1	処理を打ち切る.

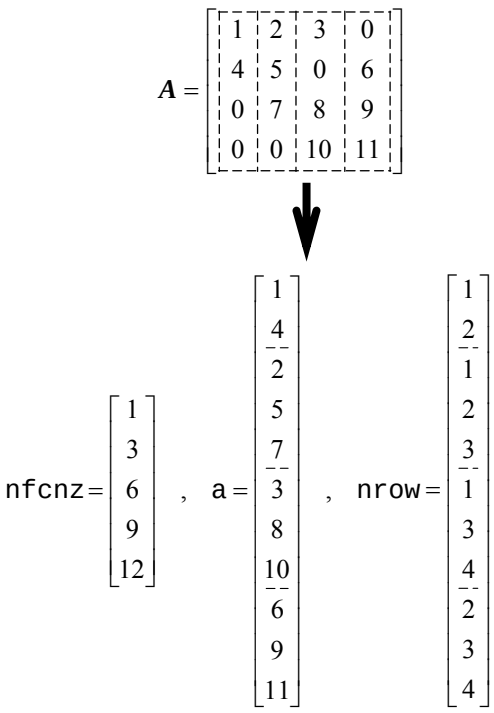


図 c_dm_vmvscc -1

行列 **A** を圧縮列格納法で格納する方法を示します.

1 次元配列 a に行列 **A** の列の非零要素を圧縮して順次格納します.

nfcnz[*i*-1]には, 行列 **A** の *i* 番目の列の最初の非零要素を 1 次元配列 a に格納する位置を設定します. 行列 **A** の次数を *n* として, nfcnz[*n*] = *nz* + 1 と設定します. *nz* は, 行列 **A** の非零要素の総数です. 配列 a の *i* 番目の要素 a[*i*-1]に格納された行列 **A** の非零要素が何番目の行にあるかを nrow[*i*-1]に設定します.

3. 使用例

スパース行列と実ベクトルの積を計算し, 結果を確認します.

```
#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 NORD    (60)
#define NX      (NORD)
#define NY      (NORD)
#define NZ      (NORD)
#define N       (NX*NY*NZ)
#define K       (N+1)
#define NDIAG   (7)

MAIN__()
{
  int    ierr, icon;
  int    i, ii, j;
  int    ne, ns, nnz;
  int    numnz, ntopcfg, ncol;
  int    length, nbase;
  int    nofst[NDIAG];
  int    nrow[K*NDIAG];
  int    nfcnz[N+1];
  int    iw[K*NDIAG][2];
```

```
    double s;
    double diag[NDIAG][K];
    double a[K*NDIAG];
    double w[K*NDIAG];
    double x[N];
    double b[N];
    double y[N];

    for (i=1; i<=N; i++){
        x[i-1]=1.0;
    }

    nofst[1]=-NX*NY;
    nofst[2]=-NX;
    nofst[3]=-1;
    nofst[4]=0;
    nofst[5]=1;
    nofst[6]=NX;
    nofst[7]=NX*NY;

    for (i=1; i<=NDIAG; i++){
      if (nofst[i-1] < 0){
        nbase=-nofst[i-1];
        length=N-nbase;
        for (j=1; j<=length; j++){
          diag[i-1][j-1]=(double)(i-1);
        }
      }
      else{
        nbase=nofst[i-1];
        length=N-nbase;
        for (j=nbase+1; j<=N; j++){
          diag[i-1][j-1]=(double)(i-1);
        }
      }
    }

    numnz = 1;
    for (j=1; j<=N; j++){
      ntopcfg = 1;
      for (i=NDIAG; i>=1; i--){
if (diag[i-1][j-1] != 0){
          ncol = j-nofst[i-1];
          a[numnz-1] = diag[i-1][j-1];
          nrow[numnz-1] = ncol;
          if (ntopcfg == 1){
            nfcnz[j-1] = numnz;
            ntopcfg = 0;
          }
          numnz = numnz+1;
        }
      }
    }
    nfcnz[N] = numnz;
    nnz = numnz-1;

    ierr = c_dm_vmvscc(a, nnz, nrow, nfcnz, N, x, y, w, (int*)iw, &icon);
    for (i=1; i<=N; i++){
      b[i-1]=0.0;
    }

    for (i=1; i<=N; i++){
      ns = nfcnz[i-1];
      ne = nfcnz[i]-1;
      for (j=ns; j<=ne; j++){
        ii = nrow[j-1];
        b[ii-1] = b[ii-1]+a[j-1]*x[i-1];
      }
    }

    s = 0.0;
    for (i=1; i<=N; i++){
      s=max(s,fabs(y[i-1]-b[i-1]));
    }

    printf("ERROR=%e\n", s);
```

```
}
```

4. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VMVSCC の項目を参照してください.

c_dm_vmvsccc

複素スパース行列と複素ベクトルの積 (圧縮列格納法)
`ierr = c_dm_vmvsccc(za, nz, nrow, nfcnz, n, zx, zy, zw, iw, &icon);`

1. 機能

$n \times n$ の複素スパース行列と複素ベクトルの積を計算します.

$$\mathbf{y} = \mathbf{A}\mathbf{x}$$

スパース行列 **A** は, 圧縮列格納法(Compressed Column Storage)で格納します.
ベクトル **x** および **y** は n 次元ベクトルです.

2. 引数

呼出し形式:

```
ierr = c_dm_vmvsccc(za, nz, nrow, nfcnz, n, zx, zy, zw, (int*)iw, &icon);
```

引数の説明:

za	dcomplex za[nz]	入力	係数行列の非零要素を格納します. za[i], i=0,…,nz-1 にスパース行列の非零要素を格納します. 圧縮列格納法については, 図 c_dm_vmvscc-1 参照. 図 c_dm_vmscc-1 の実数型の配列 a を複素数型の配列に変えたものを利用します.
nz	int	入力	係数行列 **A** の非零要素の総数
nrow	int nrow[nz]	入力	圧縮列格納法で使用される行指標で, za に格納される要素が何番目の行ベクトルに属するかを示します.
nfcnz	int nfcnz[n+1]	入力	行列 **A** の各列の非零要素を列方向に圧縮して順次配列 za に格納するとき, 対応する列の最初の非零要素が格納される位置を表します. nfcnz[n]=nz+1
n	int	入力	行列 **A** の次数 n.
zx	dcomplex zx[n]	入力	ベクトル x を格納します.
zy	dcomplex zy[n]	出力	行列とベクトルの積の結果が格納されます.
zw	dcomplex zw[nz]	作業領域	
iw	int iw[nz][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. • n < 1 • nz < 0 • nfcnz[n]≠nz+1	処理を打ち切る.

3. 使用例

複素スパース行列と複素ベクトルの積を計算します.
(並列実行を行うスレッド数は環境変数(OMP_NUM_THREADS)などで指定できます. 例えば, 4 プロセッサーのシステムで 4 つのスレッドで並列実行するときは, OMP_NUM_THREADS を 4 に設定して実行します.)

```
/* **EXAMPLE** */
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h"

#define  NORD 60
#define  NX NORD
#define  NY NORD
#define  NZ NORD
#define  N NX * NY * NZ
#define  K (N + 1)
#define  NDIAG 7

dcomplex comp_add(dcomplex, dcomplex);
dcomplex comp_sub(dcomplex, dcomplex);
dcomplex comp_mult(dcomplex, dcomplex);
double   cdabs(dcomplex);

int MAIN__() {

  int  nofst[NDIAG];
  dcomplex  zdiag[NDIAG][K], za[K * NDIAG], zw[K * NDIAG];
  int nrow[K * NDIAG], nfcnz[N + 1],
      iw[K * NDIAG][2];
  dcomplex  zx[N], zb[N], zy[N];
  int  i, ii, j, icon, nbase, length, ncol, numnz, ntopcfg, nnz, ns, ne;
  double s;

  for (i = 0; i < N; i++) {
    zx[i].re = 1.0;
    zx[i].im = 0.0;
  }

  nofst[0] = -NX * NY;
  nofst[1] = -NX;
  nofst[2] = -1;
  nofst[3] = 0;
  nofst[4] = 1;
  nofst[5] = NX;
  nofst[6] = NX * NY;

  for (i = 0; i < NDIAG; i++) {
    if (nofst[i] < 0) {
      nbase = -nofst[i];
      length = N - nbase;
      for (j = 0; j < length; j++) {
        zdiag[i][j].re = (double)i;
        zdiag[i][j].im = 0.0;
      }
    } else {
      nbase = nofst[i];
      length = N - nbase;
      for (j = nbase; j < N; j++) {
        zdiag[i][j].re = (double)i;
        zdiag[i][j].im = 0.0;
      }
    }
  }

  numnz = 1;
```

```
  for (j = 0; j < N; j++) {
    ntopcfg = 1;
    for (i = NDIAG - 1; i >= 0; i--) {
      if (zdiag[i][j].re != 0.0 || zdiag[i][j].im != 0.0) {
        ncol = (j + 1) - nofst[i];
        za[numnz - 1] = zdiag[i][j];
        nrow[numnz - 1] = ncol;

        if (ntopcfg == 1) {
          nfcnz[j] = numnz;
          ntopcfg = 0;
        }
        numnz++;
      }
    }
  }

  nfcnz[N] = numnz;
  nnz = numnz - 1;
  c_dm_vmvsccc(za, nnz, nrow, nfcnz, N, zx,
               zy, zw, (int *)iw, &icon);

  for (i = 0; i < N; i++) {
    zb[i].re = 0.0;
    zb[i].im = 0.0;
  }

  for (i = 0; i < N; i++) {
    ns = nfcnz[i];
    ne = nfcnz[i + 1] - 1;
    for (j = ns - 1; j < ne; j++) {
      ii = nrow[j];
      zb[ii - 1] = comp_add(zb[ii - 1], comp_mult(za [j], zx[i]));
    }
  }

  s = 0.0;

  for (i = 0; i < N; i++) {
    s = fmax(s, cdabs(comp_sub(zy[i], zb[i])));
  }

  printf("ERROR=%18.15lf\n", s);

  return(0);
}

dcomplex comp_add(dcomplex so1, dcomplex so2) {

  dcomplex obj;

  obj.re = so1.re + so2.re;
  obj.im = so1.im + so2.im;
  return obj;
}

dcomplex comp_sub(dcomplex so1, dcomplex so2) {

  dcomplex obj;

  obj.re = so1.re - so2.re;
  obj.im = so1.im - so2.im;
  return obj;
}

dcomplex comp_mult(dcomplex so1, dcomplex so2) {

  dcomplex obj;

  obj.re = so1.re * so2.re - so1.im * so2.im;
  obj.im = so1.re * so2.im + so1.im * so2.re;
  return obj;
}

double cdabs(dcomplex so) {
  double obj;
```

```
  obj = sqrt(so.re * so.re + so.im * so.im);
  return obj;
}
```

4. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VMVSCCC の項目を参照してください.

c_dm_vmvsd

実スパース行列と実ベクトルの積 (対角形式格納法)
`ierr = c_dm_vmvsd(a, k, ndiag, n, nofst, nlb, x, y, &icon);`

1. 機能

$n \times n$ のスパース行列とベクトルの積を計算します.

$$\mathbf{y} = \mathbf{A}\mathbf{x}$$

スパース行列 **A** は, 対角形式の格納法で格納します. ベクトル **x** 及び **y** は n 次元ベクトル.

2. 引数

呼出し形式:

```
ierr = c_dm_vmvsd((double*)a, k, ndiag, n, nofst, nlb, x, y, &icon);
```

引数の説明:

a	double a[ndiag][k]	入力	係数行列の非零要素を格納します. ("使用上の注意" a)参照).
k	int	入力	配列 a の整合寸法 (≥ 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 nofst[ndiag]	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します. 上対角ベクトル列は正, 下対角ベクトル列は負の値で表します. ("使用上の注意" 参照 a)参照).
nlb	int	入力	行列 A の下バンド幅.
x	double x[*Xlen*]	入力	ベクトル **x** を x[i], nlb ≤ i < nlb + n に格納します. *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ます.
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	次のいずれかであった. • k < 1 • n < 1 • n > k • ndiag < 1 • nlb ≠ max(–nofst[i]), 0 ≤ i < ndiag • |nofst[i]| > n–1, 0 ≤ 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します. 対角形式の格納方法については, “1.4.2 一般スパース行列の格納方法”を参照してください.

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ります. 対角ベクトル列を配列に格納する順序に制限は特にありません. 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す. 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す.

4. 使用例

実スパース行列積と実ベクトルの積を計算し, 結果を確認します.

```
#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 NDIAG   (UBANDW + LBANDW + 1)

MAIN__()
{
  double one=1.0, eps=1.e-6;
  int   ierr, icon;
  int   nlb, nub, n, i, j, k;
  int   nofst[UBANDW + LBANDW + 1];
  double a[NDIAG][NMAX], x[NMAX + UBANDW + LBANDW], y[NMAX];

  /* initialize matrix and vector */
  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_dm_vmvsd((double*)a, k, NDIAG, n, nofst, nlb, x, y, &icon);
  if (icon != 0) {
    printf("ERROR: c_dm_vmvsd 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);
    }
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VMVSD の項目を参照してください.

c_dm_vmvse

実スパース行列と実ベクトルの積 (ELLPACK 形式格納法)
`ierr = c_dm_vmvse(a, k, nw, n, icol, x, y, &icon);`

1. 機能

$n \times n$ のスパース行列とベクトルの積を計算します.

$$\mathbf{y} = \mathbf{A}\mathbf{x}$$

$n \times n$ 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します. **y** 及び **x** は n 次元ベクトル.

2. 引数

呼出し形式:

```
ierr = c_dm_vmvse((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します. ("使用上の注意" a)参照).
k	int	入力	配列 a 及び icol の整合寸法 (≥ 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 n.
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かを示します. ("使用上の注意" 参照 a)参照).
x	double x[n]	入力	ベクトル **x** を格納します.
y	double y[n]	出力	行列とベクトルの積の結果が格納されます.
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	次のいずれかであった. • k < 1 • n < 1 • 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 **A** は, 2 つの配列を使用する ELLPACK 形式の格納法で格納します. ELLPACK 形式の格納方法については, "1.4.2 一般スパース行列の格納方法"を参照してください.

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ます.

4. 使用例

実スパース行列積と実ベクトルの積を計算し, 結果を確認します.

```
#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)
#define NW      (UBANDW + LBANDW + 1)

MAIN__()
{
  double lcf=-2.0, ucf=-1.0, one=1.0, eps=1.e-6;
  int   ierr, icon;
  int   nlb, nub, n, i, j, k, ix;
  int   icol[NW][NMAX];
  double a[NW][NMAX], x[NMAX], y[NMAX];

  /* initialize matrix and vector */
  nub = UBANDW;
  nlb = LBANDW;
  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_dm_vmvse((double*)a, k, NW, n, (int*)icol, x, y, &icon);
  if (icon != 0) {
    printf("ERROR: c_dm_vmvse 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);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VMVSE の項目を参照してください.

c_dm_vpde2d

2 次元 2 階偏微分方程式の有限差分法による離散化によるスパース行列の連立 1 次方程式の生成
`ierr = c_dm_vpde2d(a1,l1, n1, n2, a2, x1, x2, b1, b2, c, f, a, k, na, n, &ndiag, nofst, r, &icon);`

1. 機能

本関数は, 長方形領域 B で以下の境界条件を満たす 2 次元 2 階偏微分方程式を, 有限差分法によって離散化して生成される連立 1 次方程式の係数行列と定数ベクトルを生成します.

$$-\left(\frac{\partial}{\partial x_1}a_1\frac{\partial u}{\partial x_1}+\frac{\partial}{\partial x_2}a_2\frac{\partial u}{\partial x_2}\right)+b_1\frac{\partial u}{\partial x_1}+b_2\frac{\partial u}{\partial x_2}+cu=f \tag{1}$$

長方形領域 B で偏微分方程式(1)を満たし, 長方形領域 B の境界で境界条件(2)を満たします.

$$\beta_1\frac{\partial u}{\partial x_1}+\beta_2\frac{\partial u}{\partial x_2}+\gamma u=\phi \tag{2}$$

偏微分方程式に現れる関数 a_1, a_2, b_1, b_2, c, f は長方形領域 B で定義されます. 境界条件に現れる $\beta_1, \beta_2, \gamma, \phi$ は長方形領域 B の境界で定義された関数です.

離散化に使う n1 × n2 の格子点は $x_{i,j}$ = (x1[i–1], x2[j–1])で定義されます.

i = 1, ... , n1, j = 1, ... , n2

B = [x1[0], x1[n1–1]] × [x2[0], x2[n2–1]].

偏微分方程式に現れる関数および境界条件は領域の格子点での値で定義されます. 離散化によって生成された係数行列 **A** は, 対角形式格納法で格納されます.

2. 引数

呼出し形式:

```
ierr = c_dm_vpde2d((double*)a1, l1, n1, n2, (double*)a2, x1, x2, (double*)b1,
           (double*)b2, (double*)c, (double*)f, (double*)a, k, na, n, &ndiag,
           nofst, r, &icon);
```

引数の説明:

a1	double a1[n2][l1]	入力	係数 $a_1(x_{i,j})$, a1[$j-1$][$i-1$], i = 1, ... , n1, j = 1, ... , n2 を設定します.
l1	int	入力	配列 a1, a2, b1, b2, c, f の 2 次元目の大きさ (l1 ≥ n1).
n1	int	入力	x_1 方向の格子点の数(n1 > 2).
n2	int	入力	x_2 方向の格子点の数(n2 > 2).
a2	double a2[n2][l1]	入力	係数 $a_2(x_{i,j})$, a2[$j-1$][$i-1$], i = 1, ... , n1, j = 1, ... , n2 を設定します.
x1	double x1[n1]	入力	x1 に格子点での x_1 座標の値を格納します. 座標の値は昇順に格納します. x1[i] < x1[i+1], i = 0, ... , n1–2
x2	double x2[n2]	入力	x2 に格子点での x_2 座標の値を格納します. 座標の値は昇順に格

			納します. x2[i] < x2[i+1], i = 0, ... , n2–2
b1	double b1[n2][l1]	入力	係数 $b_1(x_{i,j})$, および境界条件β_1を以下のように設定します. $\text{b1[j-1][i-1]} = \begin{cases} \beta_1(x_{1,j}) & i = 1 \\ \beta_1(x_{\text{n1},j}) & i = \text{n1} \\ \beta_1(x_{i,1}) & j = 1 \\ \beta_1(x_{i,\text{n2}}) & j = \text{n2} \\ b_1(x_{i,j}) & \text{上記以外} \end{cases}$
b2	double b2[n2][l1]	入力	係数 $b_2(x_{i,j})$, および境界条件β_2を以下のように設定します. $\text{b2[j-1][i-1]} = \begin{cases} \beta_2(x_{1,j}) & i = 1 \\ \beta_2(x_{\text{n1},j}) & i = \text{n1} \\ \beta_2(x_{i,1}) & j = 1 \\ \beta_2(x_{i,\text{n2}}) & j = \text{n2} \\ b_2(x_{i,j}) & \text{上記以外} \end{cases}$
c	double c[n2][l1]	入力	係数 $c(x_{i,j})$, および境界条件γを以下のように設定します. $\text{c[j-1][i-1]} = \begin{cases} \gamma(x_{1,j}) & i = 1 \\ \gamma(x_{\text{n1},j}) & i = \text{n1} \\ \gamma(x_{i,1}) & j = 1 \\ \gamma(x_{i,\text{n2}}) & j = \text{n2} \\ c(x_{i,j}) & \text{上記以外} \end{cases}$
f	double f[n2][l1]	入力	係数 $f(x_{i,j})$, および境界条件ϕを以下のように設定します. $\text{f[j-1][i-1]} = \begin{cases} \phi(x_{1,j}) & i = 1 \\ \phi(x_{\text{n1},j}) & i = \text{n1} \\ \phi(x_{i,1}) & j = 1 \\ \phi(x_{i,\text{n2}}) & j = \text{n2} \\ f(x_{i,j}) & \text{上記以外} \end{cases}$
a	double a[na][k]	出力	偏微分方程式を離散化して生成された係数行列の非零要素が対角形式で格納されます.
k	int	入力	配列 a の 2 次元目の大きさ (≥ n).
na	int	入力	配列 a の 1 次元目の大きさ (≥ ndiag).
n	int	入力	行列 **A** の次数 n (n = n1 × n2).
ndiag	int	出力	係数行列 **A** の非零要素を含む対角ベクトル列の総数. (= 5)
nofst	int nofst[ndiag]	出力	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します. 上対角ベクトル列は正, 下対角ベクトル列は負の値で表します.
r	double r[k]	出力	偏微分方程式を離散化して生成された連立 1 次方程式の右辺定数ベクトルが格納されます.
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.

コード	意　味	処理内容
30000	次のいずれかであった. • l1 < n1 • n1 < 3 • n2 < 3 • na < 5 • k < n1 × n2	処理を打ち切る.
30001	格子点の座標が昇順に格納されていない.	

3. 使用上の注意

a) 格子点の解の値について

線型系や固有値問題の解法によって計算された格子点での解の値は, 格子点の配置や数に大きく依存します.

b) 隣接する格子点について

隣接する格子点の間の距離の変化は緩やかでなければなりません. 例えば x_1 方向の条件として以下を満たすべきです.

$$0.5 \le \frac{\mathtt{x1}[i-1]-\mathtt{x1}[i-2]}{\mathtt{x1}[i]-\mathtt{x1}[i-1]} \le 2 \quad ,i = 2,...,\mathtt{n1}-1$$

(x_2 方向も同様です.) この条件が満たされていないと係数行列は悪条件となる可能性があります. 係数行列の条件数は格子ばかりでなく係数関数によっても決まることに注意してください.

4. 使用例

領域は channel$[-1, 1]^2$ とします. 偏微分方程式は

$$-\left(\frac{\partial^2 u}{\partial^2 x_1}+\frac{\partial^2 u}{\partial^2 x_2}\right)+v_1\frac{\partial u}{\partial x_1}+v_2\frac{\partial u}{\partial x_2}=0$$

回転している速度場により生じる channel を伝わる量 u の拡散をモデル化したものです.

$$v=(v_1,v_2)=v_0\cdot\left(\frac{x_2}{\sqrt{x_1^2+x_2^2}},\frac{-x_1}{\sqrt{x_1^2+x_2^2}}\right)$$

ここで, v_0 は実定数(例えば, v_0=1)です. 境界条件は次のように設定されます.

$$\begin{aligned} u &= 0 && x_2 = -1 \\ u &= 1 && x_2 = 1 \\ \frac{\partial u}{\partial n} &= 0 && \text{上記以外} \end{aligned}$$

```
#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 N1 49
#define N2 (N1)
#define L1 (N1)
#define L2 (N2)
#define KA (N1*N2)
#define NA 5
```

```
int MAIN__()
{
  double x1[L1], x2[L2], a1[L2][L1], a2[L2][L1], b1[L2][L1], b2[L2][L1];
  double c[L2][L1], f[L2][L1], a[NA][KA], r[KA], v0;
  int    nofst[NA], z1, z2, i, j, n, ndiag, icon;

  v0 = 1.0;

  /* create grid nodes nodes: */
  for (z1=0; z1<N1; z1++) {
    x1[z1] = 2*(double)(z1)/(double)(N1-1)-1.0;
  }

  for (z2=0; z2<N2; z2++) {
    x2[z2] = 2*(double)(z2)/(double)(N2-1)-1.0;
  }

  /* coefficient functions: */
  for (z2=0; z2<N2; z2++) {
    for (z1=0; z1<N1; z1++) {
      a1[z2][z1] = 1.0;
      a2[z2][z1] = 1.0;
    }

    for (z1=1; z1<N1-1; z1++) {
      b1[z2][z1] =  v0*x2[z2]/sqrt(x1[z1]*x1[z1]+x2[z2]*x2[z2]+1.0e-10);
      b2[z2][z1] = -v0*x1[z1]/sqrt(x1[z1]*x1[z1]+x2[z2]*x2[z2]+1.0e-10);
      c[z2][z1]  = 0.0;
      f[z2][z1]  = 0.0;
    }

    /* boundary conditions at faces X1=-1 and X1=1: */
    b1[z2][0]    = -1.0;
    b2[z2][0]    =  0.0;
    c[z2][0]     =  0.0;
    f[z2][0]     =  0.0;
    b1[z2][N1-1] =  1.0;
    b2[z2][N1-1] =  0.0;
    c[z2][N1-1]  =  0.0;
    f[z2][N1-1]  =  0.0;

    /* boundary conditions at faces X2=-1 and X2=1: */
    if (z2 == 0) {
      for (z1=0; z1<N1; z1++) {
        b1[z1][0] = 0.0;
        b2[z1][0] = 0.0;
        c[z1][0]  = 1.0;
        f[z1][0]  = 0.0;
      }
    } else if (z2 == N2-1) {
      for (z1=0; z1<N1; z1++) {
        b1[z1][N2-1] = 0.0;
        b2[z1][N2-1] = 0.0;
        c[z1][N2-1]  = 1.0;
        f[z1][N2-1]  = 1.0;
      }
    }
  }

  /* build the linear system: */
  n = N1*N2;
  c_dm_vpde2d((double*)a1, L1, N1, N2, (double*)a2, x1, x2, (double*)b1, (double*)b2,
              (double*)c, (double*)f, (double*)a, KA, NA, n, &ndiag, nofst, r, &icon);
  printf("icon of c_dm_vpde2d = %d\n", icon);

  /* write the matrix to a file: */
  for (j=0; j<ndiag; j++) {
    for (i=0; i<n; i+=100) {
      if(i%3 == 0) { printf("\n");};
      printf("%23.16e ", a[j][i]);
    }
  }

  for (i=0; i<ndiag; i++) {
    if(i%3 == 0) { printf("\n");};
    printf("%10d ", nofst[i]);
  }
```

```
  for (i=0; i<n; i+=100) {
    if(i%3 == 0) { printf("\n");};
    printf("%23.16e ", r[i]);
  }
  return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VPDE2D の項目を参照してください.

c_dm_vpde3d

3 次元 2 階偏微分方程式の有限差分法による離散化によるスパース行列の連立 1 次方程式の生成
`ierr = c_dm_vpde3d(a1,l1, l2, n1, n2, n3, a2, a3, x1, x2, x3, b1, b2, b3, c, f, a, k, na, n, &ndiag, nofst, r, &icon);`

1. 機能

本関数は, 長方形領域 B で以下の境界条件を満たす 3 次元 2 階偏微分方程式を, 有限差分法によって離散化して生成される連立 1 次方程式の係数行列と定数ベクトルを生成します.

$$-\left(\frac{\partial}{\partial x_1}a_1\frac{\partial u}{\partial x_1}+\frac{\partial}{\partial x_2}a_2\frac{\partial u}{\partial x_2}+\frac{\partial}{\partial x_3}a_3\frac{\partial u}{\partial x_3}\right)+b_1\frac{\partial u}{\partial x_1}+b_2\frac{\partial u}{\partial x_2}+b_3\frac{\partial u}{\partial x_3}+cu=f \tag{1}$$

長方形領域 B で偏微分方程式(1)を満たし, 長方形領域 B の境界で境界条件(2)を満たします.

$$\beta_1\frac{\partial u}{\partial x_1}+\beta_2\frac{\partial u}{\partial x_2}+\beta_3\frac{\partial u}{\partial x_3}+\gamma u=\phi \tag{2}$$

偏微分方程式に現れる関数 $a_1, a_2, a_3, b_1, b_2, b_3, c, f$ は長方形領域 B で定義されます. 境界条件に現れる$\beta_1, \beta_2, \beta_3, \gamma, \phi$は長方形領域 B の境界で定義された関数です.

離散化に使う n1 × n2 × n3 の格子点は $x_{i,j,k}$ = (x1[i–1], x2[j–1] , x3[k–1])で定義されます.

$i = 1, \ldots,$ n1, $j = 1, \ldots,$ n2, $k = 1, \ldots,$ n3

B = [x1[0], x1[n1–1]] × [x2[0], x2[n2–1]] × [x3[0], x3[n3–1]].

偏微分方程式に現れる関数および境界条件は領域の格子点での値で定義されます. 離散化によって生成された係数行列 **A** は, 対角形式格納法で格納されます.

2. 引数

呼出し形式:

```
ierr = c_dm_vpde3d((double*)a1, l1, l2, n1, n2, n3, (double*)a2, (double*)a3,
            x1, x2, x3, (double*)b1, (double*)b2, (double*)b3, (double*)c,
            (double*)f, (double*)a, k, na, n, &ndiag, nofst, r, &icon);
```

引数の説明:

a1	double a1[n3][l2][l1]	入力	係数 $a_1(x_{i,j,k})$, a1[$k-1$][$j-1$][$i-1$], $i = 1, \ldots,$ n1, $j = 1, \ldots,$ n2, $k = 1, \ldots,$ n3 を設定します.
l1	int	入力	配列 a1, a2, a3, b1, b2, b3, c, f の 3 次元目の大きさ (l1 ≥ n1).
l2	int	入力	配列 a1, a2, a3, b1, b2, b3, c, f の 2 次元目の大きさ (l2 ≥ n2).
n1	int	入力	x_1 方向の格子点の数(n1 > 2).
n2	int	入力	x_2 方向の格子点の数(n2 > 2).
n3	int	入力	x_3 方向の格子点の数(n3 > 2).
a2	double a2[n3][l2][l1]	入力	係数 $a_2(x_{i,j,k})$, a2[$k-1$][$j-1$][$i-1$], $i = 1, \ldots,$ n1, $j = 1, \ldots,$ n2, $k = 1, \ldots,$ n3 を設定します.

a3	double a3[n3][l2][l1]	入力	係数 $a_3(x_{i,j,k})$, a3[$k-1$][$j-1$][$i-1$], $i=1,\ldots,$ n1, $j=1,\ldots,$ n2, $k=1,\ldots,$ n3 を設定します.
x1	double x1[n1]	入力	x1 に格子点での x_1 座標の値を格納します. 座標の値は昇順に格納します. x1[i] < x1[i+1], i = 0, ... , n1–2
x2	double x2[n2]	入力	x2 に格子点での x_2 座標の値を格納します. 座標の値は昇順に格納します. x2[i] < x2[i+1], i = 0, ... , n2–2
x3	double x3[n3]	入力	x3 に格子点での x_2 座標の値を格納します. 座標の値は昇順に格納します. x3[i] < x3[i+1], i = 0, ... , n3–2
b1	double b1[n3][l2][l1]	入力	係数 $b_1(x_{i,j,k})$, および境界条件β_1 を以下のように設定します. b1[k−1][j−1][i−1]= $\begin{cases}\beta_1(x_{1,j,k}) & i=1\\ \beta_1(x_{\mathrm{n1},j,k}) & i=\mathrm{n1}\\ \beta_1(x_{i,1,k}) & j=1\\ \beta_1(x_{i,\mathrm{n2},k}) & j=\mathrm{n2}\\ \beta_1(x_{i,j,1}) & k=1\\ \beta_1(x_{i,j,\mathrm{n3}}) & k=\mathrm{n3}\\ b_1(x_{i,j,k}) & \text{上記以外}\end{cases}$
b2	double b2[n3][l2][l1]	入力	係数 $b_2(x_{i,j,k})$, および境界条件β_2 を以下のように設定します. b2[k−1][j−1][i−1]= $\begin{cases}\beta_2(x_{1,j,k}) & i=1\\ \beta_2(x_{\mathrm{n1},j,k}) & i=\mathrm{n1}\\ \beta_2(x_{i,1,k}) & j=1\\ \beta_2(x_{i,\mathrm{n2},k}) & j=\mathrm{n2}\\ \beta_2(x_{i,j,1}) & k=1\\ \beta_2(x_{i,j,\mathrm{n3}}) & k=\mathrm{n3}\\ b_2(x_{i,j,k}) & \text{上記以外}\end{cases}$
b3	double b3[n3][l2][l1]	入力	係数 $b_3(x_{i,j,k})$, および境界条件β_3 を以下のように設定します. b3[k−1][j−1][i−1]= $\begin{cases}\beta_3(x_{1,j,k}) & i=1\\ \beta_3(x_{\mathrm{n1},j,k}) & i=\mathrm{n1}\\ \beta_3(x_{i,1,k}) & j=1\\ \beta_3(x_{i,\mathrm{n2},k}) & j=\mathrm{n2}\\ \beta_3(x_{i,j,1}) & k=1\\ \beta_3(x_{i,j,\mathrm{n3}}) & k=\mathrm{n3}\\ b_3(x_{i,j,k}) & \text{上記以外}\end{cases}$
c	double c[n3][l2][l1]	入力	係数 $c(x_{i,j,k})$, および境界条件γを以下のように設定します. c[k−1][j−1][i−1]= $\begin{cases}\gamma(x_{1,j,k}) & i=1\\ \gamma(x_{\mathrm{n1},j,k}) & i=\mathrm{n1}\\ \gamma(x_{i,1,k}) & j=1\\ \gamma(x_{i,\mathrm{n2},k}) & j=\mathrm{n2}\\ \gamma(x_{i,j,1}) & k=1\\ \gamma(x_{i,j,\mathrm{n3}}) & k=\mathrm{n3}\\ c(x_{i,j,k}) & \text{上記以外}\end{cases}$
f	double f[n3][l2][l1]	入力	係数 $f(x_{i,j,k})$, および境界条件ϕを以下のように設定します.

$$\texttt{f[k-1][j-1][i-1]}=\begin{cases}\phi(x_{1,j,k}) & i=1\\ \phi(x_{\texttt{n1},j,k}) & i=\texttt{n1}\\ \phi(x_{i,1,k}) & j=1\\ \phi(x_{i,\texttt{n2},k}) & j=\texttt{n2}\\ \phi(x_{i,j,1}) & k=1\\ \phi(x_{i,j,\texttt{n3}}) & k=\texttt{n3}\\ f(x_{i,j,k}) & \text{上記以外}\end{cases}$$

a	double a[na][k]	出力	偏微分方程式を離散化して生成された係数行列の非零要素が対角形式で格納されます.
k	int	入力	配列 a の 2 次元目の大きさ (≥ n).
na	int	入力	配列 a の 1 次元目の大きさ (≥ ndiag).
n	int	入力	行列 **A** の次数 n (n = n1 × n2 × n3).
ndiag	int	出力	係数行列 **A** の非零要素を含む対角ベクトル列の総数. (= 7)
nofst	int nofst[ndiag]	出力	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します. 上対角ベクトル列は正, 下対角ベクトル列は負の値で表します.
r	double r[n]	出力	偏微分方程式を離散化して生成された連立 1 次方程式の右辺定数ベクトルが格納されます.
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	次のいずれかであった. • l1 < n1 • l2 < n2 • n1 < 3 • n2 < 3 • n3 < 3 • na < 7 • k < n1 × n2 × n3	処理を打ち切る.
30001	格子点の座標が昇順に格納されていない.	

3. 使用上の注意

a) 格子点の解の値について

線型系や固有値問題の解法によって計算された格子点での解の値は, 格子点の配置や数に大きく依存します.

b) 隣接する格子点について

隣接する格子点の間の距離の変化は緩やかでなければなりません. 例えば x_1 方向の条件として以下を満たすべきです.

$$0.5 \le \frac{\texttt{x1}[i-1]-\texttt{x1}[i-2]}{\texttt{x1}[i]-\texttt{x1}[i-1]} \le 2 \quad ,i=2,\ldots,\texttt{n1}-1$$

(x_2 および x_2 方向も同様です.) この条件が満たされていないと係数行列は悪条件となる可能性があります. 係数行列の条件数は格子ばかりでなく係数関数によっても決まることに注意してください.

4. 使用例

領域は channel$[-1, 1]^2 \times [0, 5]$ とします. 偏微分方程式は

$$-\left(\frac{\partial^2 u}{\partial^2 x_1}+\frac{\partial^2 u}{\partial^2 x_2}+\frac{\partial^2 u}{\partial^2 x_3}\right)+v_1\frac{\partial u}{\partial x_1}+v_2\frac{\partial u}{\partial x_2}=0$$

回転している速度場により生じる channel を伝わる量 u の拡散をモデル化したものです.

$$v=(v_1,v_2,v_3)=v_0\cdot\left(\frac{x_2}{\sqrt{x_1^2+x_2^2}},\frac{-x_1}{\sqrt{x_1^2+x_2^2}},0\right)$$

ここで, v_0 は実定数(例えば, v_0=1)です. 境界条件は次のように設定されます.

$$\begin{array}{ll} u=0 & x_3=0 \\ u=1 & x_3=5 \\ \dfrac{\partial u}{\partial n}=0 & \text{上記以外} \end{array}$$

ここで n は channel の境界での法線場を示しています.

```
#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 N1 49
#define N2 49
#define N3 25
#define L1 (N1)
#define L2 (N2)
#define L3 (N3)
#define KA (N1*N2*N3)
#define NA 7

int MAIN__()
{
  double x1[L1], x2[L2], x3[L3], a1[L3][L2][L1], a2[L3][L2][L1], a3[L3][L2][L1];
  double b1[L3][L2][L1], b2[L3][L2][L1], b3[L3][L2][L1], c[L1][L2][L3];
  double f[L3][L2][L1], a[NA][KA], r[KA], v0;
  int    nofst[NA], z1, z2, z3, i, j, n, ndiag, icon;

  v0 = 1.0;

  for (z1=0; z1<N1; z1++) {
    x1[z1] = 2*(double)z1/(double)(N1-1)-1.0;
  }
  for (z2=0; z2<N2; z2++) {
    x2[z2] = 2*(double)z2/(double)(N2-1)-1.0;
  }
  for (z3=0; z3<N3; z3++) {
    x3[z3] = (double)z3/(double)(N3-1);
  }

  /* coefficient functions: */
  for (z3=0; z3<N3; z3++) {
    for (z2=0; z2<N2; z2++) {
      for (z1=0; z1<N1; z1++) {
        a1[z3][z2][z1] = 1.0;
        a2[z3][z2][z1] = 1.0;
        a3[z3][z2][z1] = 1.0;
      }
    }

    for (z2=1; z2<N2-1; z2++) {
      for (z1=1; z1<N1-1; z1++) {
        b1[z3][z2][z1] = v0*x2[z2]/sqrt(x1[z1]*x1[z1]+x2[z2]*x2[z2]+1.0e-10);
```

```
      b2[z3][z2][z1] = v0*x1[z1]/sqrt(x1[z1]*x1[z1]+x2[z2]*x2[z2]+1.0e-10);
      b3[z3][z2][z1] = 0.0;
      c[z3][z2][z1]  = 0.0;
      f[z3][z2][z1]  = 0.0;
    }
  }

  /* boundary conditions at faces X1=-1 and X1=1: */
  for (z2=0; z2<N2; z2++) {
    b1[z3][z2][0] = -1.0;
    b2[z3][z2][0] =  0.0;
    b3[z3][z2][0] =  0.0;
    c[z3][z2][0]  =  0.0;
    f[z3][z2][0]  =  0.0;

    b1[z3][z2][N1-1] = 1.0;
    b2[z3][z2][N1-1] = 0.0;
    b3[z3][z2][N1-1] = 0.0;
    c[z3][z2][N1-1]  = 0.0;
    f[z3][z2][N1-1]  = 0.0;
  }

  /* boundary conditions at faces X2=-1 and X2=1: */
  for (z1=0; z1<N1; z1++) {
    b1[z3][0][z1] =  0.0;
    b2[z3][0][z1] = -1.0;
    b3[z3][0][z1] =  0.0;
    c[z3][0][z1]  =  0.0;
    f[z3][0][z1]  =  0.0;

    b1[z3][N2-1][z1] = 0.0;
    b2[z3][N2-1][z1] = 1.0;
    b3[z3][N2-1][z1] = 0.0;
    c[z3][N2-1][z1]  = 0.0;
    f[z3][N2-1][z1]  = 0.0;
  }

  /* boundary conditions at faces X3=0 and X3=5: */
  if (z3==0) {
    for (z1=0; z1<N1; z1++) {
      for (z2=0; z2<N2; z2++) {
        b1[0][z2][z1] = 0.0;
        b2[0][z2][z1] = 0.0;
        b3[0][z2][z1] = 0.0;
        c[0][z2][z1]  = 1.0;
        f[0][z2][z1]  = 0.0;
      }
    }
  } else if (z3==N3-1) {
    for (z1=0; z1<N1; z1++) {
      for (z2=0; z2<N2; z2++) {
        b1[N3-1][z2][z1] = 0.0;
        b2[N3-1][z2][z1] = 0.0;
        b3[N3-1][z2][z1] = 0.0;
        c[N3-1][z2][z1]  = 1.0;
        f[N3-1][z2][z1]  = 1.0;
      }
    }
  }
}

/* build the linear system: */
n = N1*N2*N3;
c_dm_vpde3d((double*)a1, L1, L2, N1, N2, N3, (double*)a2, (double*)a3, x1, x2, x3,
            (double*)b1, (double*)b2, (double*)b3, (double*)c, (double*)f, (double*)a,
            KA, NA, n, &ndiag, nofst, r, &icon);
printf("c_dm_vpde3d : icon = %d\n", icon);

/* write the matrix to a file: */
for (j=0; j<ndiag; j++) {
  for (i=0; i<n; i+=1000) {
    if(i%3 == 0) { printf("\n");};
    printf("%23.16e ", a[j][i]);
  }
}

for (i=0; i<ndiag; i++) {
```

```
      if(i%3 == 0) { printf("\n");};
      printf("%10d ", nofst[i]);
    }

    for (i=0; i<n; i+=1000) {
      if(i%3 == 0) { printf("\n");};
      printf("%23.16e ", r[i]);
    }
    return(0);
  }
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VPDE3D の項目を参照してください.

c_dm_vradau5

スティフ連立 1 階常微分方程式/微分代数方程式 (陰的ルンゲ・クッタ法)
ierr = c_dm_vradau5(n, fcn, &x, y, xend, &h, rtol, Atol, itol, jac, ijac, mljac, mujac, mas, imas, mlmas, mumas, solout, iout, work, lwork, iwork, liwork, &rpar, &ipar, &icon);

1. 機能

スティフな連立 1 階常微分方程式, または微分代数方程式の初期値問題の数値解を陰的ルンゲ・クッタ法で求めます.

$$\mathbf{M}\mathbf{y}' = \boldsymbol{f}(x,\mathbf{y}) \qquad \mathbf{y}(x_0) = \mathbf{y}_0$$

ここで **M** は n 次元の定数正方行列 (質量マトリックスと呼びます), **y** は n 次元の解ベクトル(成分を $y_1, y_2, \ldots, y_n$ とします), $\boldsymbol{f}(x,\mathbf{y})$ は n 次元の関数ベクトル(成分を $f_1, f_2, \ldots, f_n$ とします), そして $\boldsymbol{y}_0$ は $x = x_0$ おける初期値ベクトルです(成分を $y_{01}, y_{02}, \ldots, y_{0n}$ とします.) .

M が正則で単位行列 **I** でない場合は陰的な常微分方程式となります.また **M** が特異行列のときに微分代数方程式となります.

本関数は, 一回の呼出しで, 指定された点 $x_{end}\left(\neq x_0\right)$での解が求まった時点で復帰します. x_0 から x_{end} に向かって積分していく過程で 1 ステップが完了するたびに解を受け取るには引数 solout に相当するユーザ関数を用意します.

本関数は E.Hairer と G.Wanner(2011 年 3 月現在, ジュネーブ大学)によるフリーのプログラム RADAU5 に基づいた Fortran SSL II の DM_VRADAU5 を呼び出しています. RADAU5 のライセンスについては"FUJITSU SSL II スレッド並列機能使用手引書" の付録 2 を参照してください.

2. 引数

呼出し形式:

```
ierr = c_dm_vradau5(n, fcn, &x, y, xend, &h, rtol, Atol, itol, jac, ijac,
            mljac, mujac, mas, imas, mlmas, mumas, solout, iout, work, lwork,
            iwork, liwork, &rpar, &ipar, &icon);
```

引数の説明:

n	int	入力	方程式の元数(n≥1).
fcn	int	入力	$\boldsymbol{f}(x,\mathbf{y})$ の値を計算するユーザ関数の名前. プロトタイプ宣言: void fcn(int n, double x, double y[], double f, double *rpar, double *ipar); 引数の説明: n int 入力 方程式の元数

x	double	入力	独立変数 x
y	double y[n]	入力	解ベクトル y
f	double f[n]	出力	$\boldsymbol{f}(x,\boldsymbol{y})$. f[0]= f_1 , f[1]= f_2 ,….. f[n-1]= f_n

rpar,ipar については下記参照.

x	double	入力	初期値 x_0 .
		出力	解が求まった最終点 (正常終了の場合は x_{end})
y	double y[n]	入力	解の初期値. y[0] = y_{01} , y[1] = y_{02} ,…, y[n-1]= y_{0n} の順に与えます.
		出力	引数 x の出力値における解.
xend	double	入力	解を求めたい最終点 x_{end} ($x_{end} - x_0$ は正でも負でも構いません).途中の解は引数 iout の指定によって関数 solout へ渡されます.
h	double	入力	初期ステップ幅の予測値. 初期の過渡期を持ったスティフな方程式においては h = $1.0/(\boldsymbol{f}'(x,\boldsymbol{y})$ のノルム) ---通常 1×10^{-3} または 10^{-5} ---が良いとされていますが, 与えられた予測値は本関数内部で調整されるので予測値の値はさほど重要ではありません. h = 0.0 と与えた場合は, デフォルトの h = 10^{-6} が採用されます.
		出力	積分の最終ステップで採用されたステップ幅.
rtol Atol	double	入力	解に対する要求精度です. rtol には相対誤差を, Atol には絶対誤差を与えます. 両者は共にスカラーか(定数ではなく変数で指定), あるいは大きさ n のベクトルの値とします.その選択は下記の itol で行います. なお, Atol は(またはその成分 Atol[i]は)正であり, rtol は(またはその成分 rtol[i]は) 10 u よりも大であること. ここで u は丸め誤差の単位であり利用者が work[0]で指定した値か, デフォルトの c_dmach( )の値.
itol	int	入力	rtol, Atol に対するスイッチ. itol = 0 とすると, rtol, Atol は共にスカラーを与えます. このとき解の成分 y[i] (i = 0,…, n-1) の局所誤差はほぼ rtol*|y[i]| + Atol 以下になるよう積分します. itol ≠ 0 とすると rtol, Atol は共に大きさ N のベクトルとして値を与えます. このとき解の成分 y[i] (i = 0 ,…, n - 1) の局所誤差はほぼ rtol[i]*|y[i]\)| + Atol[i] 以下になるよう積分します.
jac	int	入力	$\boldsymbol{f}(x,\boldsymbol{y})$ の $\boldsymbol{y}$ に関する偏微分 (ヤコビ行列) を計算する関数の名前. ただし, 下記に述べる引数 ijac を 0 とした場合は用意する必要はありません.その場合は, ダミーの関数を用意して下さい. ijac ≠ 0 のときは, 次のような引数を持つ関数として用意します. プロトタイプ宣言: `void jac(int n, double x, double y[], double dfy[], int ldfy, double *rpar, int *ipar);`

			ldfy は dfy を 2 次元配列と考えた時の整合寸法で, jac を呼出す側の関数から与えられます. 本関数は, ヤコビ行列が密行列の場合とバンド行列の場合の二通りのケースを引数 mljac を通して区別して扱います. mljac=n の場合, ヤコビ行列を密行列として扱います. 関数 jac の引数 dfy には $\texttt{dfy[(i-1)*ldfy+j-1]} = \frac{\partial f_i}{\partial y_j}$ となるように値を入れます. $0 \le$ mljac $<$ n の場合, ヤコビ行列をバンド行列として扱います.このとき mljac , mujac はバンド行列の下バンド幅, 上バンド幅を意味し, 配列 dfy には $\texttt{dfy[(i-j+mujac)*ldfy+j-1]} = \frac{\partial f_i}{\partial y_j}$ となるように値をいれます. 例として, n = 6 で mljac = 3 , mujac = 1 の場合の dfy への格納の様子は図 c_dm_vradau5-1 を参照してください.
ijac	int	入力	ヤコビ行列の計算法を指定するスイッチ. ijac = 0 とすると, ヤコビ行列は本関数内部で差分で近似計算します. ijac $\neq 0$ とすると,ヤコビ行列は利用者が与える関数 jac で計算します.
mljac	int	入力	ヤコビ行列が密行列かバンド行列かを示すスイッチ. mljac = n とすると, ヤコビ行列は密行列であることを, また $0 \le$ mljac $<$ n のときは, ヤコビ行列がバンド行列であることを示すとともに mljac には下バンド幅を与えます.
mujac	int	入力	ヤコビ行列がバンド行列の場合に上バンド幅を与えます. mljac = n の場合は入力する必要はありません.

mas	int	入力	質量マトリックスである $n \times n$ の定数正方行列 **M** を与える関数の名前. ただし **M** が単位行列の場合を特別扱いするために引数 imas で区別します.
			imas = 0 とすると **M** が単位行列であることを示します. この場合は関数 mas を与える必要はなく, 単にダミーの関数を与えます.
			imas $\neq$ 0 とすると **M** を与える関数が必要です. 次のような引数を持つ関数として用意します.
			プロトタイプ宣言:
			`void mas(int n, double am[], int lmas,`
			`double *rpar, int *ipar);`
			lmas は am を 2 次元配列と考えた時の整合寸法で, mas を呼出す側の関数から与えられます.
			本関数は, **M** が密行列の場合とバンド行列の場合の二通りのケースを引数 mlmas を通して区別して扱います.
			mlmas = n の場合, 質量マトリックスを密行列として扱います. 関数 mas の引数 am には
			$\texttt{am[(i-1)*lmas+j-1]} = \boldsymbol{M}_{ij}$
			となるように値を入れます.
			$0 \le$ mlmas $<$ n の場合, 質量マトリックスをバンド行列として扱います.このとき mlmas, mumas はバンド行列の下バン

			ド幅, 上バンド幅を意味し, 配列 am には am[(i-j+mumas)*lmas+j-1] = $\boldsymbol{M}_{ij}$ となるように値をいれます. ヤコビ行列がバンド行列の場合の格納方法と同じです. mlmas ≤ mljac , mumas ≤ mujac であること
imas	int	入力	質量マトリックス **M** についての情報を与えます. imas = 0 とすると **M** は単位行列として見なします. imas ≠ 0 とすると **M** は単位行列ではないことを意味し, 関数 mas を与えます.
mlmas	int	入力	質量マトリックス **M** が密行列かバンド行列かを示すスイッチ. mlmas = n とすると, **M** は密行列であることを, また 0 ≤ mlmas < n のときは, **M** がバンド行列であることを示すとともに mlmas には下バンド幅を与えます. ただし, mlmas ≤ mljac とします.
mumas	int	入力	質量マトリックス **M** がバンド行列の場合の上バンド幅を与えます. mlmas = n の場合は入力する必要はありません. mumas ≤ mujac.
solout	int	入力	xend まで積分する過程で求まった途中の解を受け取る関数の名前. 途中の解を受け取るか否かは 引数 iout で指定します. iout ≠ 0 とすると次の仕様で solout へ解が渡されます. プロトタイプ宣言: void solout(int nr, double xold, double x, double y[], double cont[], int lrc, int n, double *rpar, int *ipar, int *irtrn, double *work2, int *iwork2); solout へは積分の nr 番目のステップで求まった x における解 y が渡されます.初期値 $x_0, \boldsymbol{y}_0$ は nr = 1 として最初の解として渡されます.また xend における解が求まったときも渡されます. xold は一つ前のステップでの x の値です. irtan は積分を終了させたい場合に使われ負の値を入れると, 本関数は積分を終了し, 本関数の呼出しプログラムに戻ります. 密出力について： 利用者は solout 内で, 区間 [xold, x] 内の任意の点で解を計算することができます.たとえば, あらかじめ意図した等間隔な分点での解を計算したい場合に有益です. 区間 [xold, x]内の任意の点での解を計算するには関数 c_dm_vcontr5 を呼び出します. 例えば[xold, x]内の点 S における解の i 番目 (1 ≤ i ≤ n) の成分を計算したい場合は, double c_dm_vcontr5(i, s, cont, lrc, work2, iwork2)と呼出します. cont, lrc, work2, iwork2 は solout に渡された値をそのまま利用して下さい.
iout	int	入力	solout で途中の解を受け取るか否かを示すスイッチ. iout = 0 とすると solout には途中の解は渡されません. iout ≠ 0 とすると solout に途中の解を渡します.

work	double work[lwork]	作業領域	work[0], work[1],…. , work[19]は本関数内部で使われる各種パラメタとして使われ, デフォルトのパラメタ値を使う場合は work[0], ……., work[19] には 0.0 を入れます. デフォルト以外の値を指定する場合は後述の「パラメタの詳細な指定について」を参照. work[20]から work[lwork-1]までは内部で生成されるベクトルや行列のための作業領域です. 問題の大きさに依存します. lwork は少なくとも以下の大きさです. n * (ljac + lmas + 3 * le * 12) + 20 ここで ljac は ljac = n (mljac=n の場合) ljac = mljac + mujac + 1 (mljac < n の場合) lmas は lmas = 0 (imas = 0 の場合) lmas = n (imas ≠ 0 かつ mlmas = n の場合) lmas = mlmas + mumas + 1 (mlmas < n の場合) le は le = n (mljac = n の場合) le = 2 * mljac + mujac + 1 (mljac < n の場合) ヤコビ行列が密行列でかつ **M** が単位行列の場合は lwork = 4 * n * n + 12 * n + 20 となります. 常微分方程式または微分代数方程式が 2 階以上の高階の問題に対して効率の良い使い方を後述しますが, その際には iwork[8]に正の整数 M1 を入れます. その場合の作業領域の大きさは lwork = n * (ljac + 12) + (n - M1) * (lmas + 3 * le) + 20 となります. ただしこの式における ljac, lmas, le はこれらの上記の定義式における n を n - M1 で置き換えた値です.
lwork	int	入力	作業領域 work の大きさ.
iwork	int iwork[liwork]	作業領域	liwork は少なくとも 3 × n + 20 であること. iwork[0], iwork[1],…., iwork[19] は本関数内部で使われる各種パラメタとして使われ, デフォルトのパラメタ値を使う場合は iwork[0],……., iwork[19] には 0 を入れます. デフォルト以外の値を設定する場合は後述の「**パラメタの詳細な指定について**」を参照してください. iwork[20], … , iwork[liwork-1]は作業領域として使われます.
		出力	iwork[13]から iwork[19]には xend までの積分が完了した際の統計情報を出力します. iwork [13] NFCN 関数 fcn の呼出し回数(ただし, ijac = 0 の場合のヤコビ行列近似のための呼出しは含みません). iwork[14] NJAC ヤコビ行列を計算した回数(ijac ≠ 0 の場合は関数 jac を呼出した回

数となります).

iwork[15]	NSTEP	試行されたステップの回数.
iwork[16]	NACCPT	成功したステップの回数.
iwork[17]	NREJCT	受け入れられなかったステップの回数(これは要求精度を満たさなかったときのステップ). ただし積分の開始時はカウントしていません.
iwork[18]	NDEC	線形方程式の解法で LU 分解した回数.
iwork[19]	NSOL	LU 分解した行列を使って求解した回数(ただしステップ幅の選択のために要した求解はカウントしていません).

liwork	int	入力	作業領域 iwork の大きさ.
rpar	double*	パラメタ	これら引数の用途は, 本関数を呼出すプログラムと,関数 fcn, jac, mas, solout とで数値のやり取りをする場合に使います.
ipar	int*		例えば, 本関数を呼出すプログラムで, rpar, ipar に何らかの値を設定しておけば, 関数 fcn, jac, mas, solout の中でそれを参照することができます.
icon	int	出力	コンディションコード. 下の表を参照.

$$\begin{pmatrix} a_{11} & a_{12} & & & & \\ a_{21} & a_{22} & a_{23} & & & \\ a_{31} & a_{32} & a_{33} & a_{34} & & \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & \\ & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} \\ & & a_{63} & a_{64} & a_{65} & a_{66} \end{pmatrix} \Rightarrow$$

*	a_{12}	a_{23}	a_{34}	a_{45}	a_{56}
a_{11}	a_{22}	a_{33}	a_{44}	a_{55}	a_{66}
a_{21}	a_{32}	a_{43}	a_{54}	a_{65}	*
a_{31}	a_{42}	a_{53}	a_{64}	*	*
a_{41}	a_{52}	a_{63}	*	*	*

ここで $a_{ij} = \partial f_i / \partial y_j$. *印の要素は参照されません.

図 c_dm_vradau5-1

パラメタの詳細な指定について：

配列 iwork, work の最初の 20 要素には 0 を指定するのがデフォルトですが, デフォルト値以外の値を設定できます.

iwork[0]	入力	iwork[0] ≠ 0 と指定すると, ヤコビ行列が密行列のとき, かつ **M** が単位行列の場合に, ヤコビ行列をヘッセンベルグ行列へ変換します.これは n が大きいときに効率が良くなります.
iwork[1]	入力	xend まで積分するのに要するステップ数の上限を指定します(> 0). iwork[1] =0 の場合はデフォルトで 100000 が設定されます.
iwork[2]	入力	内部で解かれる連立非線形代数方程式の解法においてニュートン法の反復回数の上限を指定します(> 0).

iwork[3]	入力	iwork[2] = 0 の場合はデフォルトで 7 が設定されます. iwork[3] = 0 の場合, ニュートン法における初期値として補外された collocation 解を使います. iwork[3] ≠ 0 の場合, 通常の初期値が使われます. 後者は, ニュートン法の収束が遅い場合(これは前述の NSTEP が NACCPT + NREJCT よりも大きい場合)に有効です.

下記の 3 つのパラメタは，指数が 1 よりも大きい微分代数方程式を解く場合に重要です. その場合, 解ベクトルの要素 y[0], y[1], …, y[n-1] は指数 1, 指数 2, 指数 3 の順に並ぶようにして下さい. 本関数の内部で行われている数値解の誤差評価は, 指数 2 の要素に対しては h が乗ぜられ, 指数 3 の要素に対しては h^2 が乗ぜられます. デフォルトである iwork[4] = 0, iwork[5] = 0, iwork[6] = 0 以外の値を設定するときは，これら 3 つの整数の和は n であること.

iwork[4]	入力	解ベクトルにおける指数 1 の要素数(≥ 0). **M** が単位行列かあるいは正則な行列なら, 常微分方程式となるので, すべての要素が指数 1 となり, その場合はデフォルト設定 (iwork[4] = 0) とすれば十分です.
iwork[5]	入力	解ベクトルにおける指数 2 の要素数(≥ 0). デフォルトは iwork[5] = 0 です.
iwork[6]	入力	解ベクトルにおける指数 3 の要素数(≥ 0). デフォルトは iwork[6] = 0 です.
iwork[7]	入力	ステップ幅の制御に関するパラメタです. iwork[7] = 1 のとき, 修正予期コントローラ (Gustafsson)によりステップ幅を制御します. iwork[7] > 1 のとき, 標準的ステップ幅コントローラによりステップ幅を制御します. デフォルト(iwork[7] = 0)は前者の方法です. 前者の方法は, より安定的な制御法であることが実験により観察されています.後者の方法は簡単な問題に対しては, しばしば若干速い制御法となります.

iwork[8]と iwork[9]は，方程式が 1 階ではなく 2 階以上の方程式を解くときに有益な手段を与えます. よく知られるように 2 階以上の方程式は新しい未知関数を導入することで 1 階の方程式に変換できます. そのとき

y[i]' = y[i+M2]　i = 0 , 1,…,M1

のような特別な式を含むことになります. ここで M1 は M2 の整数倍です. このような場合に iwork[8], iwork[9] を設定することで，効率が著しく向上します.
例として, 2 階の常微分方程式 $\boldsymbol{p}'' = \boldsymbol{g}(x, \boldsymbol{p}, \boldsymbol{p}')$ は次のように 1 階の方程式に変換できます.

$$\boldsymbol{p}' = \boldsymbol{v}$$
$$\boldsymbol{v}' = \boldsymbol{g}(x, \boldsymbol{p}, \boldsymbol{v})$$

ここで, $\boldsymbol{p}, \boldsymbol{v}$ は共に n / 2 次元のベクトルで M1 = M2 = n / 2 となります.
このような特別の構造を活かして (M1 > 0) 本関数を利用する場合は，上記に述べた引数のいくつかは異なった設定が必要になります:

jac	入力	ヤコビ行列の非自明な要素だけを計算する関数とします.例えば, 上記の 2 階から 1 階に変換された方程式で言えば, $\begin{pmatrix} \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{p}} & \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{v}} \end{pmatrix}$ なる n / 2 × n の行列だけを計算します. 一般的な仕様は次のとおりです.以下, f は 1 階に変換したあとの右辺全体, y は全体の解ベクトルを示すものとします. mljac = n - M1 と設定すると, 非自明なヤコビ行列は密行列とみなし,

		dfy[(i-1)*ldfy+j-1] = $\frac{\partial f(i+M1)}{\partial y(j)}$, i = 1,..., n - M1, j = 1,..., n と代入します. 0 ≤ mljac < n - M1 の場合は, 非自明なヤコビ行列はバンド行列とみなし, M1 = M2 × MM なる整数 MM を使って dfy[(i-j+mujac)*ldfy+(j+k×M2-1)] = $\frac{\partial f(i+M1)}{\partial y(j+k\times M2)}$ i = 1,.., n - M1, j = 1,..., M2, k = 0,.., MM と代入します. ただしバンド行列として扱う場合は, n = M1 + M2 - 1 を満たす場合に限ります.
mljac	入力	mljac = n - M1 : ヤコビ行列の非自明な部分が密行列の場合. 0 ≤ mljac < n - M1: M1 = M2 × MM としたとき MM + 1 個の部分行列, $\frac{\partial f(i+M1)}{\partial y(j+k\times M2)}$, i = 1,..., n - M1 , j = 1,..., M2, k = 0,..., MM がバンド行列であり, mljac には最大の下バンド幅を設定.
mujac	入力	上記の部分行列について, 最大の上バンド幅を設定します. mujac = n - M1 の場合は設定する必要はありません.
mas	入力	imas = 0 のときは mas は単位行列とみなします. この場合は mas を計算する関数はダミー関数を用意してください. imas ≠ 0 のときは, 右下の角に位置する部分行列だけを mas では与えます. 例を以下に示します. $\boldsymbol{M}\boldsymbol{p}'' = \boldsymbol{g}(x, \boldsymbol{p}, \boldsymbol{p}')$ なる 2 階の方程式は, 次のように 1 階の問題に変換できます. $\boldsymbol{p}' = \boldsymbol{v}$ $\boldsymbol{M}\boldsymbol{v}' = \boldsymbol{g}(x, \boldsymbol{p}, \boldsymbol{v})$ これは $\begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{M} \end{pmatrix} \begin{pmatrix} \boldsymbol{p}' \\ \boldsymbol{v}' \end{pmatrix} = \begin{pmatrix} \boldsymbol{v} \\ \boldsymbol{g}(x, \boldsymbol{p}, \boldsymbol{v}) \end{pmatrix}$ と書けます. この場合, 左辺の係数行列は「(1)機能」で表現した **M** に相当します.改めて左辺の係数行列全体を **M** で表すと, mlmas = n - M1 ときは右下の角の部分行列は密行列とみなし, mas の引数 am には am[(i-1)*lmas+j-1] = M(j+M1,i+M1), i = 1,..., n - M1 , j = 1,..., n -M1 のように与えます. mlmas ≠ n - M1 のときは右下の角の部分行列はバンド行列とみなし,am[((i-j+mumas)*lmas+j-1] = M(j+M1,i+M1)のように与えます.
mlmas	入力	mlmas = n - M1 とき, **M** の非自明な部分行列は密行列. 0 ≤ mlmas < n - M1 のとき **M** の非自明な部分行列がバンド行列であり mlmas は下バンド幅を設定. mlmas ≤ mljac であること.
mumas	入力	0 ≤ mlmas < n - M1 のとき, **M** の非自明なバンド行列部分の上バンド幅を設定. mumas ≤ mujac であること. mlmas = n - M1 の場合は, mumas の設定は不要です.
iwork[8]	入力	M1 の値(≥ 0). デフォルトでは M1 = 0.
iwork[9]	入力	M2 の値(≥ 0). デフォルトでは M1 = 0 のとき M2 = n. M1 > 0 のとき M2 = M1. iwork[8] > 0 の場合は iwork[8] + iwork[9] ≤ n であること.

work[0]	入力	丸め誤差の単位 u. c_dmach( ) ≤ work[0] < 1.0 であること.デフォルトは c_dmach( ).
work[1]	入力	ステップ幅の予測に使う安全係数. デフォルトでは 0.9. デフォルト以外の値を設定する場合, 0.001 < work[1] < 1.0 であること.
work[2]	入力	ヤコビ行列を再計算するか否かを決める数値.デフォルトは 0.001. もしヤコビ行列の計算のコストが高い場合は例えば 0.1 程度に上げます.小さな問題を解く場合は, 0.001 くらいに小さくします. work[2]に負の値を入れると, 成功したステップ後に毎回ヤコビ行列を計算します. work[2] < 1.0 であること.
work[3]	入力	ニュートン法の収束判定値. 通常は 1.0 未満の値. 小さな値を入れるほど遅くなりますが安全になります. デフォルトは MAX(10u/TOLST,MIN(0.03, $\sqrt{\text{TOLST}}$))です.ここで u は丸め誤差の単位, TOLST = 0.1·rtol^(2/3)です. rtol がベクトルのときは rtol[0]が使われます. work[3] > u / TOLST であること.
work[4], work[5]	入力	ステップ幅を変更するか否かを決めるパラメタです. もし work[4] < HNEW / HOLD < work[5]のとき, ステップ幅は変更しません. work[4], work[5]の設定値によっては, (work[2]の設定とともに)大きな問題の場合, 内部での線形方程式解法の時間の節約になります. 小さな問題に対しては work[4] = 1.0, work[5] = 1.2 が適当であり, 大きな問題でかつヤコビ行列が密行列になると work[4] = 0.99, work[5] = 2.0 が適当です. デフォルトでは work[4] = 1.0, work[5] = 1.2 です. work[4] ≤ 1.0, work[5] ≥ 1.0 であること.
work[6]	入力	ステップ幅の上限. デフォルトでは $x_{end} - x_0$ です.
work[7], work[8]	入力	ステップ幅の選択についてのパラメタです. 新しいステップ幅は次の制約のもので決定されます. work[7] ≤ HNEW/HOLD ≤ work[8] デフォルトでは work[7] = 0.2, work[8] = 8.0 です. work[7] ≤ 1.0, work[8] ≥ 1.0 であること.

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.
100	関数 solout の中で引数 irtrn に負の値が設定された.	処理を打ち切る. 解は正常.
10000	iwork[1]で設定されたステップ数の上限に達した.	処理を打ち切る. xend における解は得られていない. iwork[1]により大きな整数を設定してやり直すと正常終了する可能性はある.
21000	ステップ幅が極度に小さくなった.	処理を打ち切る.
22000	内部で解かれる線形方程式の係数行列が繰り返し特異になった.	処理を打ち切る.
30000	入力した引数の値が正しくないものがあった.	処理を打ち切る.

3. 使用上の注意

a) 利用者関数 solout の役割

iout ≠ 0 のとき, 本関数は, x_0 から x_{end} に向かって積分していく過程で 1 ステップが完了するたびにその時点での x の値と数値解 $\boldsymbol{y}$ を solout へ渡します. 具体的には, 仮に $x_0 < x_{end}$ の場合,

$x_0 < x_1 < x_2 < \ldots < x_{end}$

のように 1 ステップ進むごとに x_i とその点での数値解を solout へ渡します(x_0 と x_{end} も含みます). x_i は要求精度を満たすべく行われるステップ幅制御の結果で決まります.

利用者が意図する点列での数値解を求めたい場合は, solout の中で密出力用の関数 c_dm_vcontr5 を利用することで可能となります. 例えば一定間隔を持った点列での解を求めたい場合は使用例 1 を参照してください.

意図する点列での解を求めるのに初期値と引数 xend を繰り返し変えて呼び出すことは本関数では適切ではありません.

b) 利用者関数内でのスレッド並列化

利用者関数である fcn, jac, mas および solout のいずれも必要なら OpenMP によるスレッド並列化することは可能です (本関数からは並列リージョンの外からこれらの関数を call します).

c) 微分代数方程式の「指数」と初期値

モデル $\boldsymbol{My}' = \boldsymbol{f}(x, \boldsymbol{y})$ において **M** が正則な場合は常微分方程式となり, 解ベクトル $\boldsymbol{y}$ の成分はすべて指数 1 となり iwork[4] ～ iwork[6]はデフォルトの 0 を設定します. またどんな初期値でも唯一の数値解が得られます.

これに対し, **M** が特異行列の場合は微分代数方程式となり, 指数と初期値について注意が必要になります. 以下, ガイドラインを述べます.

M が特異行列の場合, **M** は以下のように分解できます.

$$\boldsymbol{M} = \boldsymbol{S}\begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{pmatrix}\boldsymbol{T}$$

ここで **S**, **T** は n 次元の正則行列, **I** は n より小さな次元の単位行列です(そのような **S**, **T** は例えば, 完全ピボッティングによる **M** のガウス消去法で得られます).分解された **M** をモデルに代入し, 両辺に $\boldsymbol{S}^{-1}$ を乗じ,

$$\boldsymbol{Ty} = \begin{pmatrix} \boldsymbol{u} \\ \boldsymbol{w} \end{pmatrix}$$

とおけば, モデルは以下のような形に変換できます.

$$\begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{pmatrix}\begin{pmatrix} \boldsymbol{u}' \\ \boldsymbol{w}' \end{pmatrix} = \boldsymbol{S}^{-1}\boldsymbol{f}(x, \boldsymbol{T}^{-1}\begin{pmatrix} \boldsymbol{u} \\ \boldsymbol{w} \end{pmatrix}) =: \begin{pmatrix} \boldsymbol{g}(x, \boldsymbol{u}, \boldsymbol{w}) \\ \boldsymbol{h}(x, \boldsymbol{u}, \boldsymbol{w}) \end{pmatrix}$$

または

$$\boldsymbol{u}' = \boldsymbol{g}(x, \boldsymbol{u}, \boldsymbol{w})$$
$$\boldsymbol{0} = \boldsymbol{h}(x, \boldsymbol{u}, \boldsymbol{w})$$

このように微分方程式と代数方程式に分離できた形をヘッセンベルグ形式と言い, 実用問題ではよく現れます.代数式の拘束条件を持った微分方程式とみなすことができます. 以下, このヘッセンベルグ形式の代表的な指数 1, 指数 2, 指数 3 の問題を挙げます. 本関数の使用時に参考にしてください. なお, 以下では, 式を簡略化するために独立変数の表記は省略します.

a) 指数 1 の問題

いま

$$\boldsymbol{y}' = \boldsymbol{f}(\boldsymbol{y}, \boldsymbol{z}) \qquad (3.1a)$$

$$\mathbf{0} = \boldsymbol{g}(\boldsymbol{y},\boldsymbol{z}) \tag{3.1b}$$

を考えます. ここで $\boldsymbol{y},\boldsymbol{z}$ は未知なる関数ベクトルで長さの和は n とします.

(1.1)でいう $\mathbf{M}$ は

$$\boldsymbol{M} = \begin{pmatrix} \boldsymbol{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}$$

です. (3.1b)を微分し (3.1a)を使うと

$$\mathbf{0} = \boldsymbol{g}_y(\boldsymbol{y},\boldsymbol{z})\boldsymbol{f}(\boldsymbol{y},\boldsymbol{z}) + \boldsymbol{g}_z(\boldsymbol{y},\boldsymbol{z})\boldsymbol{z}' \tag{3.1c}$$

となります. ここで $\boldsymbol{g}_y(\boldsymbol{y},\boldsymbol{z}), \boldsymbol{g}_z(\boldsymbol{y},\boldsymbol{z})$ は各々 $\partial \boldsymbol{g}(\boldsymbol{y},\boldsymbol{z})/\partial \boldsymbol{y}, \partial \boldsymbol{g}(\boldsymbol{y},\boldsymbol{z})/\partial \boldsymbol{z}$ を意味します. もし $\boldsymbol{z}'$ の係数である $\boldsymbol{g}_z(\boldsymbol{y},\boldsymbol{z})$ が解の近傍で正則ならば,

$$\boldsymbol{z}' = -\boldsymbol{g}_z^{-1}(\boldsymbol{y},\boldsymbol{z})\boldsymbol{g}_y(\boldsymbol{y},\boldsymbol{z})\boldsymbol{f}(\boldsymbol{y},\boldsymbol{z})$$

が得られます. この場合, $\boldsymbol{y},\boldsymbol{z}$ 共に指数 1 となります. 初期値 $\boldsymbol{y}_0, \boldsymbol{z}_0$ は(3.1b)を満たす必要があります.

b) 指数 2 の問題

$$\boldsymbol{y}' = \boldsymbol{f}(\boldsymbol{y},\boldsymbol{z}) \tag{3.2a}$$

$$\mathbf{0} = \boldsymbol{g}(\boldsymbol{y}) \tag{3.2b}$$

のように代数式の中に $\mathbf{z}$ が含まれていない場合を考えます. ここでも

$$\boldsymbol{M} = \begin{pmatrix} \boldsymbol{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}$$

です. (3.2b)を微分すると

$$\mathbf{0} = \boldsymbol{g}_y(\boldsymbol{y})\boldsymbol{f}(\boldsymbol{y},\boldsymbol{z}) \tag{3.2c}$$

が得られます.さらに (3.2c)を微分すると $\mathbf{z}'$ の係数は

$$\boldsymbol{g}_y(\boldsymbol{y})\boldsymbol{f}_z(\boldsymbol{y},\boldsymbol{z}) \tag{3.2d}$$

となります.もし解の近傍で(3.2d)が正則であるならば, $\mathbf{y}$ は指数 1, $\mathbf{z}$ は指数 2 となります. 初期値 $\mathbf{y}_0, \mathbf{z}_0$ については (3.2b)だけでなく(3.2c)も満たすものである必要があります.

c) 指数 3 の問題

$$\boldsymbol{y}' = \boldsymbol{f}(\boldsymbol{y},\boldsymbol{z}) \tag{3.3a}$$

$$\boldsymbol{z}' = \boldsymbol{k}(\boldsymbol{y},\boldsymbol{z},\boldsymbol{u}) \tag{3.3b}$$

$$\mathbf{0} = \boldsymbol{g}(\boldsymbol{y}) \tag{3.3c}$$

を考えます. ここで解ベクトル $\mathbf{y}, \mathbf{z}, \mathbf{u}$ の長さの和は n とします. また $\boldsymbol{M}$ は

$$\boldsymbol{M} = \begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \end{pmatrix}$$

となります. (3.3c)を微分し, (3.3a)を使うと

$$\boldsymbol{0} = \boldsymbol{g}_y \boldsymbol{f} \tag{3.3d}$$

を得ます. これを微分して(3.3a,b) を使うと

$$\boldsymbol{0} = \boldsymbol{g}_{yy}(\boldsymbol{f}, \boldsymbol{f}) + \boldsymbol{g}_y \boldsymbol{f}_y \boldsymbol{f} + \boldsymbol{g}_y \boldsymbol{f}_z \boldsymbol{k} \tag{3.3e}$$

を得ます. ここで右辺第一項は, 行列 $\mathbf{g}_y$ を微分して得られる行列 $\mathbf{g}_{yy}$ とベクトル $\mathbf{f}$ の積を意味します. 続いて(3.3e)を微分すると $\mathbf{u}'$ が現れますがその係数である $\mathbf{g}_y\mathbf{f}_z\mathbf{k}_u$ が解の近傍で正則であるなら, (3.3a,b,c)の問題では, $\mathbf{y}$ は指数 1, $\mathbf{z}$ は指数 2, $\mathbf{u}$ は指数 3 となります. 初期値 $\mathbf{y}_0, \mathbf{z}_0$ および $\mathbf{u}_0$ は(3.3 c,d,e)を満たすものである必要があります.

4. 使用例

■使用例 1： $\mathbf{y}' = \mathbf{f}(x, \mathbf{y})$ の形の問題
常微分方程式の初期値問題,

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= \frac{((1-y_1^2)y_2 - y_1)}{\varepsilon}, \varepsilon = 10^{-6} \\ y_1(0) &= 2, y_2(0) = 0 \end{aligned}$$

において, $x = 1,2,\ldots,11$ での解を求め, 印刷するプログラムを以下に示します.

この問題のヤコビ行列 $\partial\mathbf{f}/\partial\mathbf{y}$ は

$$\begin{pmatrix} \frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial y_2} \\ \frac{\partial f_2}{\partial y_1} & \frac{\partial f_2}{\partial y_2} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ (-2y_1y_2 - 1)/\varepsilon & (1-y_1^2)/\varepsilon \end{pmatrix}$$

であり, 引数 jac に相当する関数 jvpol で右辺の行列を 2 次元配列 dfy に代入しています.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h"

#define     ND 2
#define     LWORK  (4 * ND * ND + 12 * ND + 20)  /* 60 */
#define     LIWORK (3 * ND + 20) /* 26 */

void  solout(int, double, double, double*, double*, int, int,
                        double*, int*, int*, double*, int*);
```

```
void  jvpol(int, double, double*, double*, int, double*, int*);
void  fvpol(int, double, double*, double*, double*, int*);
void  dummy(int, double*, int, double*, int*);

int MAIN__() {
  double  y[ND], work[LWORK];
  int  iwork[LIWORK];
  double  rpar[2];

  int  i, n, ijac, mljac, imas, itol, mujac, iout, icon, mlmas, mumas;
  int  ipar;
  double  x, xend, rtol, Atol, h;

  rpar[0] = 1.0e-6;
  rpar[1] = 0.2;
  n = ND;
  ijac = 1;
  mljac = n;
  imas = 0;
  iout = 1;
  x = 0.0;
  y[0] = 2.0;
  y[1] = -0.66;
  xend = 11.0;
  rtol = 1.0e-4;
  Atol = 1.0 * rtol;
  itol = 0;
  h = 1.0e-6;
  for (i = 0; i < 20; i++) {
    iwork[i] = 0;
    work[i] = 0.0;
  }
  c_dm_vradau5(n, fvpol, &x, y, xend, &h,
               rtol, Atol, itol,
               jvpol, ijac, mljac, mujac,
               dummy, imas, mlmas, mumas,
               solout, iout,
               work, LWORK, iwork, LIWORK,
               rpar, &ipar, &icon);
  printf(" ICON= %d\n", icon);
  printf(" X =%5.2lf    Y =%18.10e%18.10e\n", x, y[0], y[1]);
  return(0);
}

void solout(int nr, double xold, double x, double *y, double *cont,
            int lrc, int n, double *rpar, int *ipar, int *irtrn,
            double *work2, int *iwork2) {

  double      prm1, prm2;
```

```
  if (nr == 1) {
    printf(" X =%5.2lf    Y =%18.10le%18.10le    NSTEP =%4d\n",
           x, y[0], y[1], nr - 1);
  } else {
label_10:   ;
    if (x >= rpar[1]) {
/* --- CONTINUOUS OUTPUT FOR RADAU5 */
      prm1 = c_dm_vcontr5(1, rpar[1], cont, lrc, work2, iwork2);
      prm2 = c_dm_vcontr5(2, rpar[1], cont, lrc, work2, iwork2);
      printf(" X =%5.2lf    Y =%18.10le%18.10le    NSTEP =%4d\n",
             rpar[1], prm1, prm2, nr - 1);
      rpar[1] = rpar[1] + 0.2;
      goto label_10;
    }
  }
  return;
}

void fvpol(int n, double x, double *y, double *f, double *rpar, int *ipar) {

  f[0] = y[1];
  f[1] = ((1 - (y[0] * y[0])) * y[1] - y[0]) / rpar[0];
  return;
}

void jvpol(int n, double x, double *y, double *dfy, int ldfy, double *rpar,
           int *ipar) {

  dfy[0] = 0.0;
  dfy[1] = 1.0;
  dfy[ldfy] = (-2.0 * y[0] * y[1] - 1.0) / rpar[0];
  dfy[ldfy + 1] = (1.0 - (y[0] * y[0])) / rpar[0];
  return;
}

void dummy(int n, double *am, int lmas, double *rpar, int *ipar) {

  return;
}
```

■使用例 2： $\mathbf{y}' = \mathbf{f}(x, \mathbf{y})$ の形の問題. ヤコビ行列がバンド行列の場合.

以下に示す偏微分方程式を考えます. t は時刻, x はスカラーの空間変数とします.

$$\frac{\partial u}{\partial t} = A + u^2 v - (B+1)u + \alpha \frac{\partial^2 u}{\partial x^2}$$

$$\frac{\partial v}{\partial t} = Bu - u^2 v + \alpha \frac{\partial^2 v}{\partial x^2}$$

$0 \le x \le 1, A = 1, B = 3, \alpha = 1/50$

x についての境界条件 : $u(0,t) = u(1,t) = 1, v(0,t) = v(1,t) = 3$

t についての初期条件 : $u(x,0) = 1 + \frac{1}{2}\sin(2\pi x), v(x,0) = 3$

空間変数に関する 2 階微分を, $x_i = i/(N+1)$ $(1 \le i \le N)$, $\Delta x = 1/(N+1)$ から定まる N 点格子上の差分で置き換えることにより, 上記の偏微分方程式は以下に示す, 独立変数が t で $u_i = u(t, x_i), v_i = (t, x_i)$ を未知関数とする 2N 元の常微分方程式の初期値問題となります. ここで u' などは t についての微分を表すものとします.

$$u_i' = 1 + u_i^2 v_i - 4u_i + \alpha/(\Delta x)^2 (u_{i-1} - 2u_i + u_{i+1})$$

$$v_i' = 3u_i - u_i^2 v_i + \alpha/(\Delta x)^2 (v_{i-1} - 2v_i + v_{i+1})$$

$$u_0(t) = u_{N+1}(t) = 1, v_0(t) = v_{N+1}(t) = 3$$

$$u_i(0) = 1 + \frac{1}{2}\sin(2\pi x_i), v_i(0) = 3, i = 1,2,\ldots,N$$

この問題を本関数で解くにあたり, 解ベクトル $\boldsymbol{y}$ を

$\mathbf{y} = (u_1, v_1, u_2, v_2, \ldots, u_N, v_N)^{\mathrm{T}}$ とします. すると, ヤコビ行列は下バンド幅, 上バンド幅が共に 2 のバンド行列になります. このため引数 mljac と mujac は 2 と設定します. 以下のプログラムでは n = 500, xend = 10, iout = 0 (途中の解を solout へ渡さない)とし, xend での解ベクトルの一部の成分を印刷しています.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h"

#define  ND 1000
#define  NL 2
#define  NU 2
#define  LWORK  ((7 * NL + 4 * NU + 16) * ND + 20) /* 38020 */
#define  LIWORK (3 * ND + 20) /* 3020 */

void  fbrus(int, double, double*, double*, double*, int*);
void  jbrus(int, double, double*, double*, int, double*, int*);
void  solout(int, double, double, double*, double*, int, int,
             double*, int*, int*, double*, int*);
void  dummy(int, double*, int, double*, int*);

int MAIN__() {
  double  x, xend, y[ND], work[LWORK];
  int  iwork[LIWORK];
  double  rpar[2];
  int  ipar;
  double  pi, usdelq, gamma, gamma2, anp1, xi, rtol, Atol, h;
```

```
  int  i, n, n2, ijac, mljac, mujac, mlmas, mumas, imas, iout, itol, icon;

  pi = 3.14159265358979324;
  n = 500;
  n2 = 2 * n;
  usdelq = ((double)(n + 1)) * ((double)(n + 1));
  gamma = 0.02 * usdelq;
  gamma2 = 2.0 * gamma;
  rpar[0] = gamma;
  rpar[1] = gamma2;
  x = 0.0;
  xend = 10.0;
  anp1 = n + 1;
  for (i = 1; i <= n; i++) {
    xi = i / anp1;
    y[(2 * i) - 1] = 3.0;
    y[(2 * i) - 2]  = 1.0 + 0.5 * sin(2.0 * pi * xi);
  }
  ijac = 1;
/*  Jacobian is a banded matrix. */
  mljac = NL;
  mujac = NU;
  imas = 0;
/*  Output Routine is not used. */
  iout = 0;
  rtol = 1.0e-6;
  Atol = rtol;
  itol = 0;
  h = 1.0e-6;
  for (i = 0; i < 20; i++) {
    work[i] = 0.0;
    iwork[i] = 0;
  }
  mlmas = 0;
  mumas = 0;
  c_dm_vradau5(n2, fbrus, &x, y, xend, &h,
              rtol, Atol, itol,
              jbrus, ijac, mljac, mujac,
              dummy, imas, mlmas, mumas,
              solout, iout,
              work, LWORK, iwork, LIWORK,
              rpar, &ipar, &icon);
  printf(" ICON= %d\n", icon);
  printf(" %18.10e%18.10e%18.10e%18.10e\n", y[0], y[1], y[n2 - 2], y[n2 - 1]);
  return(0);
}

void solout(int nr, double xold, double x, double *y, double *cont,
            int lrc, int n, double *rpar, int *ipar, int *irtrn,
            double *work2, int *iwork2) {
  return;
```

```
}

void fbrus(int n2, double x, double *y, double *f, double *rpar, int *ipar) {
  int     i, n, iu, iv;
  double  gamma, ui, vi, uim, vim, uip, vip, prod;

  n = n2 / 2;
  gamma = rpar[0];
  i = 1;
  iu = 2 * i - 1;
  iv = 2 * i;
  ui = y[iu -1];
  vi = y[iv - 1];
  uim = 1.0;
  vim = 3.0;
  uip = y[iu + 1];
  vip = y[iv + 1];
  prod = ui * ui * vi;
  f[iu - 1] = 1.0 + prod - 4.0 * ui + gamma * (uim - 2.0 * ui + uip);
  f[iv - 1] = 3.0 * ui - prod + gamma * (vim - 2.0 * vi + vip);
  for (i = 2; i <= n-1; i++) {
    iu = 2 * i - 1;
    iv = 2 * i;
    ui = y[iu - 1];
    vi = y[iv - 1];
    uim = y[iu - 3];
    vim = y[iv - 3];
    uip = y[iu + 1];
    vip = y[iv + 1];
    prod = ui * ui * vi;
    f[iu - 1] = 1.0 + prod -4.0 * ui + gamma * (uim - 2.0 * ui + uip);
    f[iv - 1] = 3.0 * ui - prod + gamma * (vim - 2.0 * vi + vip);
  }
  i = n;
  iu = 2 * i - 1;
  iv = 2 * i;
  ui = y[iu - 1];
  vi = y[iv - 1];
  uim = y[iu - 3];
  vim = y[iv - 3];
  uip = 1.0;
  vip = 3.0;
  prod = ui * ui * vi;
  f[iu - 1] = 1.0 + prod - 4.0 * ui + gamma * (uim - 2.0 * ui + uip);
  f[iv - 1] = 3.0 * ui - prod + gamma * (vim - 2.0 * vi + vip);
  return;
}

void jbrus(int n2, double x, double *y, double *dfy, int ldfy, double *rpar,
          int *ipar) {
```

```
  int     i, n, iu, iv;
  double  gamma, gamma2, ui, ui2, vi, uivi;


  n = n2 / 2;
  gamma = rpar[0];
  gamma2 = rpar[1];
  for (i = 1; i <= n; i++) {
    iu = 2 * i - 1;
    iv = 2 * i;
    ui = y[iu - 1];
    vi = y[iv - 1];
    uivi = ui * vi;
    ui2 = ui * ui;
    dfy[(2 * ldfy) + (iu - 1)] = 2.0 * uivi - 4.0 - gamma2;
    dfy[ldfy + (iv - 1)] = ui2;
    dfy[(3 * ldfy) + (iu - 1)] = 3.0 - 2.0 * uivi;
    dfy[(2 * ldfy) + (iv - 1)] = -ui2 - gamma2;
    dfy[ldfy + (iu - 1)] = 0.0;
    dfy[(3 * ldfy) + (iv - 1)] = 0.0;
  }
  for (i = 1; i <= n2 - 2; i++) {
    dfy[i + 1] = gamma;
    dfy[(4 * ldfy) + (i - 1)] = gamma;
  }
  return;
}

void dummy(int n, double *am, int lmas, double *rpar, int *ipar) {

  return;
}
```

■使用例 3： $\mathbf{y}'' = \mathbf{f}(x, \mathbf{y}, \mathbf{y}')$ の形の 2 階の問題.

長方形領域$\Omega = \{(x, y); 0 \le x \le 2, 0 \le y \le 4/3\}$ で定義された以下に示す偏微分方程式を考えます.

$$\frac{\partial^2 u}{\partial t^2} + \omega \frac{\partial u}{\partial t} + \sigma \Delta \Delta u = f(x, y, t), \text{ ここで } \Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

境界条件: $u|_{\partial\Omega} = 0, \Delta u|_{\partial\Omega} = 0$

初期条件: $u(x, y, 0) = 0, \frac{\partial u}{\partial t}(x, y, 0) = 0$

長方形領域を x 方向, y 方向共に幅 h＝　　で分割してできる 8×5 個の内側格子点での解を求めます. まず,

$x_i = ih, y_j = jh, i = 1,2,\ldots,8, j = 1,2,\ldots,5$

$u_{ij} = u(x_i, y_j, t)$

として空間変数に関する微分を差分で置き換えます. 次に $v_{ij} = u'_{ij}$ と置けば,

$$u''_{ij} = v_{ij}$$
$$v'_{ij} = -\omega v_{ij} - \frac{\sigma}{h^4}(20u_{ij} - 8u_{i-1\,j} - 8u_{i+1\,j} + 2u_{i+1\,j+1} + 2u_{i+1\,j-1}$$
$$+ 2u_{i-1\,j-1} + 2u_{i-1\,j+1} + u_{i-2\,j} + u_{i+2\,j} + u_{i\,j-2} + u_{i\,j+2}) + f(x_i, y_j, t)$$

なる常微分方程式が得られます. ここで $k = i + 8(j-1)$ なる対応で (i, j) から k にマッピングし,

$y_k = u_{ij}, y_{k+40} = v_{ij}$ と置けば,

$$(y_1, y_2, \ldots, y_{40}, y_{41}, \ldots, y_{80})^{\mathrm{T}}$$

を解ベクトルとする常微分方程式となります. これを本関数で解く場合, `iwork[8]` = 40 と与えて, ヤコビ行列は非自明な部分だけを与えれば効率よく積分することができます. 以下のプログラムでは,
$\omega = 1000, \sigma = 100$

$$f(x,y,t) = \begin{cases} 200(\mathrm{e}^{-5(\mathrm{t}-\mathrm{x}-2)^2} + \mathrm{e}^{-5(\mathrm{t}-\mathrm{x}-5)^2}) & y = y_2 \text{または} y_4 \text{のとき} \\ 0 & y \neq y_2 \text{かつ} y \neq y_4 \text{のとき} \end{cases}$$

とします. また非自明なヤコビ行列はバンド行列 (上バンド幅, 下バンド幅共に 2×8)となります.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h"

#define  MX  8
#define  MY  5
#define  ND  (2 * MX * MY) /* 80 */
#define  LWORK  (4 * ND * ND + 12 * ND + 20) /* 26580 */
#define  LIWORK  (3 * ND + 20) /* 260 */

void  fplate(int, double, double*, double*, double*, int*);
void  jplatsb(int, double, double*, double*, int, double*, int*);
void  solout(int, double, double, double*, double*, int, int,
             double*, int*, int*, double*, int*);
void  dummy(int, double*, int, double*, int*);

int MAIN__() {
  double  y[ND], work[LWORK];
  int  iwork[LIWORK];

  double  rpar[4];
  int  ipar[9];
  int  i, k, n, nx, ny, nachs1, nachs2, nxm1, nym1, ndemi, imas, iout;
  int  itol, ijac, mljac, mujac, mlmas, mumas, icon;
  double  omega, stiffn, weight, denom, delx,  ush4, fac, x, rtol, Atol;
  double  h, xend;
```

```
  nx = MX;
  ny = MY;
  nachs1 = 2;
  nachs2 = 4;
  nxm1 = nx - 1;
  nym1 = ny - 1;
  ndemi = nx * ny;
  omega = 1000.0;
  stiffn = 100.0;
  weight = 200.0;
  denom = nx + 1;
  delx = 2.0 / denom;
  ush4 = 1.0 / ((delx * delx) * (delx * delx));
  fac = stiffn * ush4;
  n = ND;
  imas = 0;
/*  --- OUTPUT ROUTINE IS USED DURING INTEGRATION */
  iout = 1;
/*  --- INITIAL VALUES */
  x = 0.0;
  for (i = 0; i < n; i++) {
    y[i] = 0.0;
  }
/*  --- REQUIRED TOLERANCE */
  rtol = 1.0e-6;
  Atol = rtol * 1.0e-3;
  itol = 0;
/*  --- INITIAL STEP SIZE */
  h = 1.0e-2;
/*  --- SET DEFAULT VALUES */
  for (i = 0; i < 20; i++) {
    work[i] = 0.0;
    iwork[i] = 0;
  }
/*  --- SECOND ORDER OPTION AND BANDED */
  ijac = 1;
  iwork[8] = n / 2;
  mljac = 2 * MX;
  mujac = 2 * MX;
/*  --- ENDPOINT OF INTEGRATION */
  xend = 7.0;
/*  --- COMMUNICATION VALUES */
  ipar[0] = nx;
  ipar[1] = nxm1;
  ipar[2] = ny;
  ipar[3] = nym1;
  ipar[4] = ndemi;
  ipar[5] = nachs1;
  ipar[6] = nachs2;
```

```
  ipar[7] = mljac;
  ipar[8] = mujac;
  rpar[0] = omega;
  rpar[1] = delx;
  rpar[2] = fac;
  rpar[3] = weight;

/*  --- CALL OF THE FUNCTION RADAU5 */
  c_dm_vradau5(n, fplate, &x, y, xend, &h,
               rtol, Atol, itol,
               jplatsb, ijac, mljac, mujac,
               dummy, imas, mlmas, mumas,
               solout, iout,
               work, LWORK, iwork, LIWORK,
               rpar, ipar, &icon);
  printf(" ICON= %d\n", icon);
  for (k = 0; k < n; k++) {
    printf(" %-22.15le\n", y[k]);
  }
  return(0);
}

void solout(int nr, double xold, double x, double *y, double *cont,
            int lrc, int n, double *rpar, int *ipar, int *irtrn,
            double *work2, int *iwork2) {
  int  nhalf;

  nhalf = n / 2;
  printf(" X =%9.5lf Y(1) and Y(%3d)=%18.10lf%18.10lf NSTEP =%4d\n",
         x, nhalf, y[0], y[nhalf - 1], nr - 1);
  return;
}

void fplate(int n, double x, double *y, double *f, double *rpar, int *ipar) {
  int  i, j, k, nx, nxm1, ny, nym1, ndemi, nachs1, nachs2;
  double  omega, delx, fac, weight, uc, xi, force;

  nx = ipar[0];
  nxm1 = ipar[1];
  ny = ipar[2];
  nym1 = ipar[3];
  ndemi = ipar[4];
  nachs1 = ipar[5];
  nachs2 = ipar[6];
  omega = rpar[0];
  delx = rpar[1];
  fac = rpar[2];
  weight = rpar[3];
```

```
  for (i = 1; i <= nx; i++) {
    for (j = 1; j <= ny; j++) {
      k = i + nx * (j - 1);
/*  -------- SECOND DERIVATIVE ---- */
      f[k - 1] = y[(k - 1) + ndemi];
/*  ------ CENTRAL POINT--- */
      uc = 16.0 * y[k - 1];
      if (i > 1) {
        uc = uc + y[k - 1];
        uc = uc - 8.0 * y[k - 2];
      }
      if (i < nx) {
        uc = uc + y[k - 1];
        uc = uc - 8.0 * y[k];
      }
      if (j > 1) {
        uc = uc + y[k - 1];
        uc = uc - 8.0 * y[(k - 1) - nx];
      }
      if (j < ny) {
        uc = uc + y[k - 1];
        uc = uc - 8.0 * y[(k - 1) + nx];
      }
      if (i > 1 && j > 1)
        uc = uc + 2.0 * y[k - nx - 2];
      if (i < nx && j > 1)
        uc = uc + 2.0 * y[k - nx];
      if (i > 1 && j < ny)
        uc = uc + 2.0 * y[k + nx - 2];
      if (i < nx && j < ny)
        uc = uc + 2.0 * y[k + nx];
      if (i > 2)
        uc = uc + y[k - 3];
      if (i < nxm1)
        uc = uc + y[k + 1];
      if (j > 2)
        uc = uc + y[(k - 2 * nx) - 1];
      if (j < nym1)
        uc = uc + y[(k + 2 * nx) - 1];
      if (j == nachs1 || j == nachs2) {
        xi = i * delx;
        force = exp(-5.0 * ((x - xi - 2.0) * (x - xi - 2.0))) +
                exp(-5.0 * ((x - xi - 5.0) * (x - xi - 5.0)));
      } else {
        force = 0.0;
      }
      f[k + ndemi - 1] = -omega * y[k + ndemi - 1] - fac * uc + force * weight;
    }
  }
```

```
  return;
}

void jplatsb(int n, double x, double *y, double *dfy, int ldfy, double *rpar,
             int *ipar) {
  int  i, j, k, nx, nxm1, ny, nym1, ndemi, mu, mljac, mujac;
  double  omega, fac, fac2, fac8, fac16;

  nx = ipar[0];
  nxm1 = ipar[1];
  ny = ipar[2];
  nym1 = ipar[3];
  ndemi = ipar[4];
  mljac = ipar[7];
  mujac = ipar[8];
  omega = rpar[0];
  fac = rpar[2];

  for (i = 0; i < mljac + mujac + 1; i++) {
    for (j = 0; j < ldfy; j++) {
      dfy[(i * ldfy) + j] = 0.0;
    }
  }
  mu = 2 * nx + 1;
  fac2 = fac * 2.0;
  fac8 = fac * 8.0;
  fac16 = fac * 16.0;
  for (i = 1; i <= nx; i++) {
    for (j = 1; j <= ny; j++) {
      k = i + nx * (j - 1);
      dfy[((mu - 1) * ldfy) + (k - 1)] = -fac16;
      if (i > 1) {
        dfy[((mu - 1) * ldfy)  + (k - 1)] =
            dfy[((mu - 1) * ldfy) + (k - 1)] - fac;
        dfy[(mu * ldfy) + (k - 2)] = fac8;
      }
      if (i < nx) {
        dfy[((mu - 1) * ldfy)  + (k - 1)] =
            dfy[((mu - 1) * ldfy) + (k - 1)] - fac;
        dfy[((mu - 2) * ldfy) + k] = fac8;
      }
      if (j > 1) {
        dfy[((mu - 1) * ldfy)  + (k - 1)] =
            dfy[((mu - 1) * ldfy) + (k - 1)] - fac;
        dfy[((mu + nx - 1) * ldfy) + (k - nx - 1)] = fac8;
      }
      if (j < ny) {
        dfy[((mu - 1) * ldfy)  + (k - 1)] =
            dfy[((mu - 1) * ldfy) + (k - 1)] - fac;
```

```
          dfy[((mu - nx - 1) * ldfy) + (k + nx - 1)] = fac8;
        }
        if (i > 1 && j > 1)
          dfy[((mu + nx)  * ldfy) + (k - nx - 2)] = -fac2;
        if (i < nx && j > 1)
          dfy[((mu + nx - 2) * ldfy) + (k - nx)] = -fac2;
        if (i > 1 && j < ny)
          dfy[((mu - nx) * ldfy) + (k + nx - 2)] = -fac2;
        if (i < nx && j < ny)
          dfy[((mu - nx - 2) * ldfy) + (k + nx)] = -fac2;
        if (i > 2)
          dfy[((mu + 1) * ldfy) + (k - 3)] = -fac;
        if (i < nxm1)
          dfy[((mu - 3) * ldfy) + (k + 1)] = -fac;
        if (j > 2)
          dfy[((mu + 2 * nx - 1) * ldfy) + (k - 2 * nx - 1)] = -fac;
        if (j < nym1)
          dfy[((mu - 2 * nx - 1) * ldfy) + (k + 2 * nx - 1)] = -fac;
        dfy[((mu - 1) * ldfy) + (k + ndemi - 1)] = -omega;
      }
    }
    return;
}

void dummy(int n, double *am, int lmas, double *rpar, int *ipar) {

  return;
}
```

■使用例 4： $\mathbf{My}' = \mathbf{f}(x, \mathbf{y})$ の形の微分代数方程式.

t を独立変数とする 8 つの未知関数 $y_1, y_2, \ldots, y_8$ があって, 以下の方程式を満たすものとします.

$$
\begin{aligned}
&C_5(y_2' - y_1') = y_1/R_9 \\
&-C_5(y_2' - y_1') = \alpha f(y_4 - y_3) - U_b/R_8 + y_2/R_8 \\
&-C_4 y_3' = y_3/R_7 - f(y_4 - y_3) \\
&C_3(y_5' - y_4') = -U_b/R_6 + y_4(1/R_5 + 1/R_6) + (1-\alpha) f(y_4 - y_3) \\
&-C_3(y_5' - y_4') = -U_b/R_4 + y_5/R_4 + \alpha f(y_7 - y_6) \\
&-C_2 y_6' = y_6/R_3 - f(y_7 - y_6) \\
&C_1(y_8' - y_7') = -U_b/R_2 + y_7(1/R_1 + 1/R_2) + (1-\alpha) f(y_7 - y_6) \\
&C_1(y_7' - y_8') = y_8/R_0 - U_e(t)/R_0
\end{aligned}
$$

ここで

$C_k = k \cdot 10^{-6}, k = 1,2,\ldots,5$
$R_0 = 1000, R_k = 9000, k = 1,2,\ldots,9$
$f(y_i - y_j) = \beta(\mathrm{e}^{(y_i - y_j)/U_F} - 1)$
$U_F = 0.026, \alpha = 0.99, \beta = 10^{-6}, U_b = 6$
$U_e(t) = 0.1 \cdot \sin(200\pi t)$

$\mathbf{y} = (y_1, y_2, \ldots, y_8)^{\mathrm{T}}$ とすれば, 8 つの方程式の左辺は **M***y*'と表せます. ここで, **M** は次のような三重対角行列です.

$$\boldsymbol{M} = \begin{pmatrix} -C_5 & C_5 & & & & & & \\ C_5 & -C_5 & & & & & & \\ & & -C_4 & & & & & \\ & & & -C_3 & C_3 & & & \\ & & & C_3 & -C_3 & & & \\ & & & & & -C_2 & & \\ & & & & & & -C_1 & C_1 \\ & & & & & & C_1 & -C_1 \end{pmatrix}$$

M は明らかに階数 5 の特異行列となり, したがって方程式は微分代数方程式となります. かつ, 指数 1 の微分代数方程式です.
$t = 0$ から $t = 0.2$ まで積分するとします. このとき初期値 ***y***(0)は上記方程式の 8 つの右辺を成分とするベクトルが **M** の像空間に入るように選べば無矛盾な解を得ることができます. その初期値は次のとおりです.

$y_1(0) = 0, y_2(0) = U_b - y_1(0) \cdot R_8 / R_9, y_3(0) = y_4(0) = U_b / (R_6 / R_5 + 1)$
$y_5(0) = U_b, y_6(0) = y_7(0) = U_b / (R_2 / R_1 + 1), y_8(0) = 0$

8 つの方程式の右辺を成分とするベクトルのヤコビ行列は上バンド幅が 2, 下バンド幅が 1 のバンド行列となります. なお, この微分代数方程式では解の成分はすべて指標 1 であることが証明できます. 以下のプログラムでは, 時間幅 0.0025 ごとの解を印刷しています.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h"

#define  ND  8
#define  LJAC  4
#define  LMAS  3
#define  LE  5
#define  LWORK  (ND * (LJAC + LMAS + 3 * LE + 12) + 20) /* 292 */
#define  LIWORK  (3 * ND + 20) /* 44 */

void  fampl(int, double, double*, double*, double*, int*);
void  jbampl(int, double, double*, double*, int, double*, int*);
void  bbampl(int, double*, int, double*, int*);
void  solout(int, double, double, double*, double*, int, int,
             double*, int*, int*, double*, int*);

int MAIN__() {
```

```
double  y[ND], work[LWORK], rpar[16];
int  iwork[LIWORK];
double  ue, ub, uf, alpha, beta, r0, r1, r2, r3, r4, r5, r6, r7, r8, r9;
double  x, xend, rtol, Atol, h;
int  i, n, ijac, mljac, mujac, imas, mlmas, mumas, iout, itol, ipar;
int  icon;

ue = 0.1;
rpar[0] = ue;
ub = 6.0;
rpar[1] = ub;
uf = 0.026;
rpar[2] = uf;
alpha = 0.99;
rpar[3] = alpha;
beta = 1.0e-6;
rpar[4] = beta;
r0 = 1000.0;
rpar[5] = r0;
r1 = 9000.0;
rpar[6] = r1;
r2 = 9000.0;
rpar[7] = r2;
r3 = 9000.0;
rpar[8] = r3;
r4 = 9000.0;
rpar[9] = r4;
r5 = 9000.0;
rpar[10] = r5;
r6 = 9000.0;
rpar[11] = r6;
r7 = 9000.0;
rpar[12] = r7;
r8 = 9000.0;
rpar[13] = r8;
r9 = 9000.0;
rpar[14] = r9;
rpar[15] = 0.0025;
ipar = 0;
n = 8;
ijac = 1;
mljac = 1;
mujac = 2;
imas = 1;
mlmas = 1;
mumas = 1;
iout = 1;
x = 0.0;
y[0] = 0.0;
```

```
  y[1] = ub - y[0] * r8 / r9;
  y[2] = ub / (r6 / r5 + 1.0);
  y[3] = ub / (r6 / r5 + 1.0);
  y[4] = ub;
  y[5] = ub / (r2 / r1 + 1.0);
  y[6] = ub / (r2 / r1 + 1.0);
  y[7] = 0.0;
  xend = 0.2;
  rtol = 1.0e-5;
  Atol = 1.0e-6 * rtol;
  itol = 0;
  h = 1.0e-6;
  for (i = 0; i < 20; i++) {
    iwork[i] = 0;
    work[i] = 0.0;
  }
  c_dm_vradau5(n, fampl, &x, y, xend, &h,
              rtol, Atol, itol,
              jbampl, ijac, mljac, mujac,
              bbampl, imas, mlmas, mumas,
              solout, iout,
              work, LWORK, iwork, LIWORK, rpar, &ipar, &icon);
  printf(" ICON= %d\n", icon);
  printf(" X =%7.4lf    Y =%18.10le%18.10le\n", x, y[0], y[1]);
  return(0);
}

void solout(int nr, double xold, double x, double *y, double *cont,
            int lrc, int n, double *rpar, int *ipar, int *irtrn,
            double *work2, int  *iwork2) {
  double  prm1, prm2;

  if (nr == 1) {
    printf(" X =%7.4lf    Y =%18.10le%18.10le    NSTEP =%4d\n",
           x, y[0], y[1], nr - 1);
  } else {
Label_10:   ;
    if (x >= rpar[15]) {
      prm1 = c_dm_vcontr5(1, rpar[15], cont, lrc, work2, iwork2);
      prm2 = c_dm_vcontr5(2, rpar[15], cont, lrc, work2, iwork2);
      printf(" X =%7.4lf    Y =%18.10le%18.10le    NSTEP =%4d\n",
             rpar[15], prm1, prm2, nr - 1);
      rpar[15] = rpar[15] + 0.0025;
      goto Label_10;
    }
  }
  return;
}
```

```
void fampl(int n, double x, double *y, double *f, double *rpar, int *ipar) {
  double  ue, ub, uf, alpha, beta, r0, r1, r2, r3, r4, r5, r6, r7, r8, r9;
  double  w, uet, fac1, fac2;

  ue = rpar[0];
  ub = rpar[1];
  uf = rpar[2];
  alpha = rpar[3];
  beta = rpar[4];
  r0 = rpar[5];
  r1 = rpar[6];
  r2 = rpar[7];
  r3 = rpar[8];
  r4 = rpar[9];
  r5 = rpar[10];
  r6 = rpar[11];
  r7 = rpar[12];
  r8 = rpar[13];
  r9 = rpar[14];
  w = 2.0 * 3.141592654 * 100.0;
  uet = ue * sin(w * x);
  fac1 = beta * (exp((y[3] - y[2]) / uf) - 1.0);
  fac2 = beta * (exp((y[6] - y[5]) / uf) - 1.0);
  f[0] = y[0] / r9;
  f[1] = (y[1] - ub) / r8 + alpha * fac1;
  f[2] = y[2] / r7 - fac1;
  f[3] = y[3] / r5 + (y[3] - ub) / r6 + (1.0 - alpha) * fac1;
  f[4] = (y[4] - ub) / r4 + alpha * fac2;
  f[5] = y[5] / r3 - fac2;
  f[6] = y[6] / r1 + (y[6] - ub) / r2 + (1.0 - alpha) * fac2;
  f[7] = (y[7] - uet) / r0;
  return;
}

void jbampl(int n, double x, double *y, double *dfy, int ldfy, double *rpar,
            int *ipar) {
  double  uf, alpha, beta, r0, r1, r2, r3, r4, r5, r6, r7, r8, r9;
  double  fac14, fac27;
  int  j;

  uf = rpar[2];
  alpha = rpar[3];
  beta = rpar[4];
  r0 = rpar[5];
  r1 = rpar[6];
  r2 = rpar[7];
  r3 = rpar[8];
  r4 = rpar[9];
  r5 = rpar[10];
```

```
  r6 = rpar[11];
  r7 = rpar[12];
  r8 = rpar[13];
  r9 = rpar[14];
  fac14 = beta * exp((y[3] - y[2]) / uf) / uf;
  fac27 = beta * exp((y[6] - y[5]) / uf) / uf;
  for (j = 0; j < 8; j++) {
    dfy[j] = 0.0;
    dfy[ldfy + j] = 0.0;
    dfy[3 * ldfy + j] = 0.0;
  }
  dfy[2 * ldfy] = 1.0 / r9;
  dfy[2 * ldfy + 1] = 1.0 / r8;
  dfy[ldfy + 2] = -alpha * fac14;
  dfy[3] = alpha * fac14;
  dfy[2 * ldfy + 2] = 1.0 / r7 + fac14;
  dfy[ldfy + 3] = -fac14;
  dfy[2 * ldfy + 3] = 1.0 / r5 + 1.0 /r6 + (1.0 - alpha) * fac14;
  dfy[3 * ldfy + 2] = -(1.0 - alpha) * fac14;
  dfy[2 * ldfy + 4] = 1.0 / r4;
  dfy[ldfy + 5] = -alpha * fac27;
  dfy[6] = alpha * fac27;
  dfy[2 * ldfy + 5] = 1.0 / r3 + fac27;
  dfy[ldfy + 6] = -fac27;
  dfy[2 * ldfy + 6] = 1.0 / r1 + 1.0/ r2 + (1.0 - alpha) * fac27;
  dfy[3 * ldfy + 5] = -(1.0 - alpha) * fac27;
  dfy[2 * ldfy + 7] = 1.0 / r0;
  return;
}

void bbampl(int n, double *b, int lb, double *rpar, int *ipar) {
  int  i;
  double  c1, c2, c3, c4, c5;

  for (i = 0; i < 8; i++) {
    b[i] = 0.0;
    b[2 * lb + i] = 0.0;
  }
  c1 = 1.0e-6;
  c2 = 2.0e-6;
  c3 = 3.0e-6;
  c4 = 4.0e-6;
  c5 = 5.0e-6;
  b[lb] = -c5;
  b[1] = c5;
  b[2 * lb] = c5;
  b[lb + 1] = -c5;
  b[lb + 2] = -c4;
  b[lb + 3] = -c3;
```

```
    b[4] = c3;
    b[2 * lb + 3] = c3;
    b[lb + 4] = -c3;
    b[lb + 5] = -c2;
    b[lb + 6] = -c1;
    b[7] = c1;
    b[2 * lb + 6] = c1;
    b[lb + 7] = -c1;
    return;
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VRADAU5 の項目及び[33], [34], [35], [36]を参照してください.

c_dm_vrann3

正規乱数の生成
`ierr = c_dm_vrann3(dam, dsd, &ix, da, k, n, dwork, nwork, &icon);`

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, 擬似乱数を生成します.

$$f(x)=\frac{1}{\sigma\sqrt{2\pi}}\exp\left(\frac{-(x-m)^2}{2\sigma^2}\right) \tag{1}$$

2. 引数

呼出し形式:

```
ierr = c_dm_vrann3(dam, dsd, &ix, (double*)da, k, n, (double*)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 (>0).
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 のまま呼び出してください. 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ます. (“使用上の注意” a)参照)
		出力	0.
da	double da[NUMT][k]	出力	各スレッドごとに生成される異なる擬似正規乱数. ここで NUMT は本ルーチンで並列実行を行うスレッドの数. P 番目(0～NUMT–1)のスレッドで生成された n 個の擬似乱数が da[P][0], ... , da[P][n–1]に返却されます. (“使用上の注意” f)参照)
k	int	入力	配列 da の 2 次元目の大きさ (≥ n).
n	int	入力	da に返却される各スレッドで生成される正規分布擬似乱数の個数. (“使用上の注意” b)参照)
dwork	double dwork[NUMT][nwork]	作業領域	本関数を繰り返し呼び出す場合は, 内容および本関数で並列実行を行うスレッドの数 NUMT を変更してはなりません. dwork には, 本関数が現在の位置から再度呼び出される場合に必要な, すべての現情報が格納されます.
nwork	int	入力	配列 dwork の 2 次元目の大きさ. nwork ≥ 1156.
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	k < n または k < 1	処理を打ち切る.
30001	次のいずれかであった. • nwork が小さ過ぎた. • ix < 0 • dsd ≤ 0	
30002	内部エラー	
30003～30009	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.	
40000	ix が大き過ぎた.	

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場合は, 何かランダムな入力が必要です. したがって, 利用者が出発値 ix を与える必要があります. この出発値は"たね(*seed*)"と呼ばれています. 本関数の初回呼び出しでは, "たね" ix は正整数である必要があります(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 を指定してください. これは同じ乱数列から続いて乱数を求めるためです. 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 を返しています. 本関数は, *seed* = *seed* × omp_get_num_threads( )+omp_get_thread_num( )+1 のように, スレッド番号+1, omp_get_thread_num( )+1 を *seed* に付加しています. このように異なるスレッドで使用される *seed* は, 他のプロセッサでの *seed* とは全く別であることが保証され, 10^{18} 未満の長さの乱数列であればオーバーラップすることはありません.

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します. n ≤ 0 の時は疑似乱数を返却しません.

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)します. それは, 関数呼び出しのオーバーヘッドが減少するためです. 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 の 1 次元目の大きさ k を最大の n の大きさ以上に確保する必要があります.

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ます. 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りません.

d) nwork について

dwork[i][0], … , dwork[i][nwork–1](i = 0, … , NUMT–1)は, 本関数によって使用されます. nwork は, 本関数の各呼び出しで変えないでください. nwork には少なくとも 1156 以上, また十分大きな数(例えば, nwork = 100,000)を与えると効率的です.

e) 同じ乱数の再生成について

dwork[i][0], … , dwork[i][nwork–1](i = 0, … , NUMT–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ます.

f) NUMT について

本関数を並列実行するスレッド数 NUMT は, 環境変数 OMP_NUM_THREADS または実行環境関数 omp_set_num_threads( )で指定します.

実行環境関数 omp_set_num_threads( )で指定するときは, 2 回目以降の呼び出しの直前でも初回呼び出しと同じスレッド数を omp_set_num_threads( )で指定してください.

4. 使用例

10, 000, 000 × 4 個の擬似正規乱数を生成し, 平均と標準偏差を計算します.

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

#define min(a,b)  ((a) < (b) ? (a) : (b))

#define NUMT  4
#define NRAN  10000000
#define SEED  12345
#define NWMAX 100000
#define NBUF  120000
#define K     (NBUF)

int MAIN__()
{
  double da[NUMT][K], dwork[NUMT][NWMAX];
  double dsum, dsum2, dssum, dssum2, dmean, dsig, dam, dsd;
  int    ngen, ntot, krpt, ix, iz, i, j, n, nwork, icon;

  /* Initialize ix,n and nwork */
  ix    = SEED;
  n     = NBUF;
  nwork = NWMAX;
  dam   = 0.0;
  dsd   = 1.0;
  dsum  = 0.0;
  dssum = 0.0;

  printf("Seed               = %d\n", ix);
  printf("Mean               = %e\n", dam);
  printf("Standard deviation = %e\n", dsd);

  /* ngen counts down to 0 */
  ngen = NRAN;
  ntot = NRAN*NUMT;

  /* Generate ngen numbers with maximum NBUF at a time. */
  krpt = (NRAN+NBUF-1)/NBUF;

  printf("Generating %d numbers with %d calls to c_dm_vrann3 on %d threads.\n",
         ntot, krpt, NUMT);

  omp_set_num_threads(NUMT);

  for (iz=0; iz<krpt; iz++) {
    n = min(NBUF,ngen);
    c_dm_vrann3(dam, dsd, &ix, (double*)da, K, n, (double*)dwork, nwork, &icon);

    if(icon != 0) printf("c_dm_vrann3 : icon = %d\n", icon);

    /* Accumulate sum of numbers */
    dsum2 = 0.0;
    for (j=0; j<NUMT; j++) {
      for (i=0; i<n; i++) {
        dsum2 += da[j][i];
      }
    }
```

```
      /* Accumulate sum of numbers globally. */
      dssum2 = 0.0;
      for (j=0; j<NUMT; j++) {
        for (i=0; i<n; i++) {
          dssum2 += da[j][i]*da[j][i];
        }
      }

      dsum  += dsum2;
      dssum += dssum2;

      /* Count down numbers still to generate on each processor */
      ngen -= n;
    }

    /* Compute overall mean. */
    dmean = dsum / (double)ntot;
    printf("Sample mean %e\n", dmean);

    /* Compute overall sample standard deviation. */
    dsig = dssum / (double)ntot;
    printf("Sample standard deviation %e\n", dsig);
    return(0);
  }
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VRANN3 の項目を参照してください.

c_dm_vrann4

正規乱数の生成 (Wallace 法)
`ierr = c_dm_vrann4(dam, dsd, &ix, da, k, n, dwork, nwork, &icon);`

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, 擬似乱数を生成します.

$$f(x)=\frac{1}{\sigma\sqrt{2\pi}}\exp\left(\frac{-(x-m)^2}{2\sigma^2}\right) \tag{1}$$

2. 引数

呼出し形式:

```
ierr = c_dm_vrann4(dam, dsd, &ix, (double*)da, k, n, (double*)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 (>0).
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 のまま呼び出してください. 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ます. (“使用上の注意” a)参照)
		出力	0.
da	double da[NUMT][k]	出力	各スレッドごとに生成される異なる擬似正規乱数. ここで NUMT は本ルーチンで並列実行を行うスレッドの数. P 番目(0～NUMT–1)のスレッドで生成された n 個の擬似乱数が da[P][0], ... , da[P][n–1]に返却されます. (“使用上の注意” f)参照)
k	int	入力	配列 da の 2 次元目の大きさ (≥ n).
n	int	入力	da に返却される各スレッドで生成される正規分布擬似乱数の個数. (“使用上の注意” b)参照)
dwork	double dwork[NUMT][nwork]	作業領域	本関数を繰り返し呼び出す場合は, 内容および本関数で並列実行を行うスレッドの数 NUMT を変更してはなりません. dwork には, 本関数が現在の位置から再度呼び出される場合に必要な, すべての現情報が格納されます. (“使用上の注意” c), f)参照)
nwork	int	入力	配列 dwork の 2 次元目の大きさ. nwork ≥ 1350.
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	k < n または k < 1	処理を打ち切る.
30001	次のいずれかであった. • nwork が小さ過ぎた. • ix < 0 • dsd ≤ 0	
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 が大き過ぎた.	
40001～40002	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場合は, 何かランダムな入力が必要です. したがって, 利用者が出発値 ix を与える必要があります. この出発値は“たね(*seed*)”と呼ばれています. 本関数の初回呼び出しでは, “たね” ix は正整数である必要があります(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 を指定してください. これは同じ乱数列から続いて乱数を求めるためです. 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 を返しています.

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します. n ≤ 0 の時は疑似乱数を返却しません.

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)します. それは, 関数呼び出しのオーバーヘッドが減少するためです. 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 の 1 次元目の大きさ k を最大の n の大きさ以上に確保する必要があります.

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ます. 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りません.

d) nwork について

dwork[i][0], … , dwork[i][nwork–1](i = 0, … , NUMT–1)は, 本関数によって使用されます. nwork は, 本関数の各呼び出しで変えないでください. nwork には少なくとも 1350 以上, また十分大きな数(例えば nwork = 500,000)を与えると効率的です.

e) 同じ乱数の再生成について

dwork[i][0], … , dwork[i][nwork–1](i = 0, … , NUMT–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ます.

f) NUMT について

本関数を並列実行するスレッド数 NUMT は, 環境変数 OMP_NUM_THREADS または実行環境関数 omp_set_num_threads( )で指定します.

実行環境関数 omp_set_num_threads( )で指定するときは, 2 回目以降の呼び出しの直前でも初回呼び出しと同じスレッド数を omp_set_num_threads( )で指定してください.

g) Wallace 法について

本ルーチンで使われている Wallace 法は, c_dm_vrann3 で使われている Polar 法に比べて約 3 倍高速です.

4. 使用例

$10,000,000 \times 4$ 個の擬似正規乱数を生成し, 平均と標準偏差を計算します.

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

#define min(a,b)  ((a) < (b) ? (a) : (b))

#define NUMT  4
#define NRAN  10000000
#define SEED  12345
#define NWMAX 100000
#define NBUF  120000
#define K     (NBUF)

int MAIN__()
{
  double da[NUMT][K], dwork[NUMT][NWMAX];
  double dsum, dsum2, dssum, dssum2, dmean, dsig, dam, dsd;
  int    ngen, ntot, krpt, ix, iz, i, j, n, nwork, icon;

  /* Initialize ix,n and nwork */
  ix    = SEED;
  n     = NBUF;
  nwork = NWMAX;
  dam   = 0.0;
  dsd   = 1.0;
  dsum  = 0.0;
  dssum = 0.0;

  printf("Seed               = %d\n", ix);
  printf("Mean               = %e\n", dam);
  printf("Standard deviation = %e\n", dsd);

  /* ngen counts down to 0 */
  ngen = NRAN;
  ntot = NRAN*NUMT;

  /* Generate ngen numbers with maximum NBUF at a time. */
  krpt = (NRAN+NBUF-1)/NBUF;

  printf("Generating %d numbers with %d calls to c_dm_vrann4 on %d threads.\n",
         ntot, krpt, NUMT);

  omp_set_num_threads(NUMT);

  for (iz=0; iz<krpt; iz++) {
    n = min(NBUF,ngen);
    c_dm_vrann4(dam, dsd, &ix, (double*)da, K, n, (double*)dwork, nwork, &icon);

    if(icon != 0) printf("c_dm_vrann4 : icon = %d\n", icon);

    /* Accumulate sum of numbers */
    dsum2 = 0.0;
    for (j=0; j<NUMT; j++) {
      for (i=0; i<n; i++) {
        dsum2 += da[j][i];
      }
    }

    /* Accumulate sum of numbers globally. */
```

```
      dssum2 = 0.0;
      for (j=0; j<NUMT; j++) {
        for (i=0; i<n; i++) {
          dssum2 += da[j][i]*da[j][i];
        }
      }

      dsum  += dsum2;
      dssum += dssum2;

      /* Count down numbers still to generate on each processor */
      ngen -= n;
    }

    /* Compute overall mean. */
    dmean = dsum / (double)ntot;
    printf("Sample mean %e\n", dmean);

    /* Compute overall sample standard deviation. */
    dsig = dssum / (double)ntot;
    printf("Sample standard deviation %e\n", dsig);
    return(0);
  }
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VRANN4 の項目を参照してください.

c_dm_vranu4

一様乱数 [0, 1)の生成
`ierr = c_dm_vranu4(&ix, da, k, n, dwork, nwork, &icon);`

1. 機能

各スレッドで, 区間[0, 1)上での一様分布から各々異なる擬似乱数列を生成します.

2. 引数

呼出し形式:

```
ierr = c_dm_vranu4(&ix, (double*)da, k, n, (double*)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 のまま呼び出してください. 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ます. ix < 8000000. (“使用上の注意” a)参照)
		出力	0.
da	double da[NUMT][k]	出力	区間[0, 1)で各スレッドごとに生成される異なる一様な擬似乱数. ここで NUMT は本ルーチンで並列実行を行うスレッドの数. P 番目(0〜NUMT–1)のスレッドで生成された n 個の擬似乱数が da[P][0], … , da[P][n–1]に返却されます.
k	int	入力	配列 da の 2 次元目の大きさ (≥ n).
n	int	入力	da に返却される各スレッドで生成される一様分布擬似乱数の個数. (“使用上の注意” b)参照)
dwork	double dwork[NUMT][nwork]	作業領域	本関数を繰り返し呼び出す場合は, 内容および本関数で並列実行を行うスレッドの数 NUMT を変更してはなりません. dwork には, 本関数が現在の位置から再度呼び出される場合に必要な, すべての現情報が格納されます.
nwork	int	入力	配列 dwork の 2 次元目の大きさ. nwork ≥ 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.
30000	k < n または k < 1	処理を打ち切る.
30001	nwork が小さ過ぎた.	
30002	内部チェックで誤りを検出した.	
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 が大き過ぎた.	

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場合は, 何かランダムな入力が必要です. したがって, 利用者が出発値 ix を与える必要があります. この出発値は“たね(*seed*)”と呼ばれています. 本関数の初回呼び出しでは, “たね” ix は正整数である必要があります(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 を指定してください. これは同じ乱数列から続いて乱数を求めるためです. 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 を返しています. 本関数は, *seed* = *seed* × omp_get_num_threads( )+omp_get_thread_num( )+1 のように, スレッド番号+1, omp_get_thread_num( )+1 を *seed* に付加しています. このように異なるスレッドで使用される *seed* は, 他のプロセッサでの *seed* とは全く別であることが保証され, 10^{18} 未満の長さの乱数列であればオーバーラップすることはありません.

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します. n ≤ 0 の時は疑似乱数を返却しません.

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)します. それは, 関数呼び出しのオーバーヘッドが減少するためです. 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 の 1 次元目の大きさ k を最大の n の大きさだけ確保する必要があります.

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ます. 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りません.

d) nwork について

dwork[i][0], … , dwork[i][nwork−1](i = 0, … , NUMT−1)は, 本関数によって使用されます. nwork は, 本関数の各呼び出しで変えてはなりません. nwork には少なくとも 388 以上を与えると効率的です.

e) 同じ乱数の再生成について

dwork[i][0], … , dwork[i][nwork−1](i = 0, … , NUMT−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ます.

f) NUMT について

本関数を並列実行するスレッド数 NUMT は, 環境変数 OMP_NUM_THREADS または実行環境関数 omp_set_num_threads( )で指定します.

実行環境関数 omp_set_num_threads( )で指定するときは, 2 回目以降の呼び出しの直前でも初回呼び出しと同じスレッド数を omp_set_num_threads( )で指定してください.

4. 使用例

1,000,000×4 個の擬似一様乱数を計算し, 平均と標準偏差を計算します.

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

#define min(a,b)  ((a) < (b) ? (a) : (b))
#define NT        (4)
#define RAN       (1000000)
```

```
#define NWMAX     (5000)
#define BUF       (25000)

MAIN__()
{
  double da[NT][BUF], dwork[NT][NWMAX];
  double sum, sum2, mean, sig;
  unsigned int gen, tot, rpt, i, j;
  int    tno, ix, n, nwork, icon, ierr;

  /* Initialize ix, n and nwork */
  ix = 123;
  printf("Seed = %d\n", ix);

  nwork = NWMAX;
  sum   = 0.0;

  /* gen counts down to 0 */
  gen = RAN;
  tot = RAN*NT;

  /* Generate ngen numbers on each thread with maximum BUF at a time */
  rpt = (RAN+BUF-1)/BUF;
  printf("Generating %d numbers with %d calls to c_dm_vranu4 on %d threads.\n",
         tot, rpt, NT);

  for(j=0; j<rpt; j++) {
    n    = min(BUF,gen);
    sum2 = 0.0;

    omp_set_num_threads(NT);
    ierr = c_dm_vranu4(&ix, (double*)da, BUF, n, (double*)dwork, nwork, &icon);

    if (icon != 0) {
      printf("ERROR: c_dm_vranu4 failed with icon = %d\n", icon);
      exit(1);
    }

    /* Accumulate sum of numbers locally */
    for(tno=0; tno<NT; tno++)
      for(i=0; i<n; i++) sum2 += da[tno][i];

    /* Accumulate sum of numbers globally */
    sum += sum2;

    /* Count down numbers still to generate on each processor */
    gen -= n;
  }

  /* Compute overall mean */
  mean = sum/tot;
  printf("mean = %e\n", mean);

  /* Compute deviation from 0.5 normalized by expected value 1/sqrt(12*ntot).     */
  /* This should be (approximately) normally distributed with mean 0, variance 1. */
  sig = (mean-0.50)*sqrt(12.0*tot);
  printf("Normalized deviation = %e\n", sig);
  return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VRANU4 の項目及び[4], [9], [10], [24], [42], [53]を参照してください.

c_dm_vranu5

一様乱数 [0, 1)の生成 (MRG8)
`ierr = c_dm_vranu5(&ix, da, n, j, dwork, &icon);`

1. 機能

区間[0, 1)上での一様分布から擬似乱数列を 8 次の原始多項式による多重再帰ジェネレータ(MRG8:Multiple Recursive Generator with 8th-order full primitive polynomials)で生成します.

本関数では使用するスレッド数に関係なく同じ疑似乱数列を生成します. 乱数の再現性が必要な場合には本関数をご利用ください. このため, c_dm_vranu4 とはインタフェースが異なります.

また, 本関数では疑似乱数列の j 個先までジャンプする機能をサポートしています. この機能は並列に乱数を生成する場合に利用できます.

従来関数の c_dm_vranu4 は本関数よりも高速に疑似乱数を生成します. 乱数生成の性能を優先する場合は c_dm_vranu4 をご利用ください.

本関数と c_dm_vranu4 は両方とも著名な乱数検定プログラム TESTU01 の bigCrush テストをパスします.

2. 引数

呼出し形式:

```
ierr = c_dm_vranu5(&ix, da, n, j, dwork, &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 のまま呼び出してください. 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ます. (“使用上の注意” a)参照)
		出力	0.
`da`	`double da[n]`	出力	区間[0, 1)で生成される一様な擬似乱数.
`n`	`int`	入力	`da` に返却される擬似乱数の個数.
`j`	`long`	入力	疑似乱数列の先頭を `j` 個先までジャンプします. 疑似乱数列の最初から必要な場合は 0 を設定してください. (“使用上の注意” b)参照)
`dwork`	`double dwork[8]`	作業領域	本関数を繰り返し呼び出す場合は, 内容を変更しないでください. `dwork` には, 本関数が現在の位置から再度呼び出される場合に必要な, すべての現情報が格納されます. (“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	`ix` < `0`, `n` < `1` または `j` < 0	処理を打ち切る.

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場合は, 何かランダムな入力が必要です. したがって, 利用者が出発値 ix を与える必要があります. この出発値は“たね(*seed*)”と呼ばれています. 本関数の初回呼び出しでは, “たね” ix は正整数である必要があります(例外事項については, d)を参照). 2 回目以降の呼び出しでは ix には 0 を指定してください. これは同じ乱数列から続いて乱数を求めるためです. 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 を返しています.

b) j について

本関数では, パラメタ j に 0 以上の値を入力することで, 乱数列を j 個先までジャンプすることができます. プロセス並列で計算するとき, プロセスごとに同じ ix を与え, j を変えることにより, プロセスごとに異なる疑似乱数を生成することができます. (“ 使用例” 使用例 2, 3 参照).

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ます. 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りません.

d) 同じ乱数の再生成について

dwork[i] (i = 0, ... ,7)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ます.

4. 使用例

使用例 1
1,000,000 個の擬似一様乱数を生成し, 平均値を計算します. 出発値は 123 とします.
(並列実行を行うスレッド数は環境変数(OMP_NUM_THREADS)などで指定できます. 例えば, 4 つのスレッドで並列実行するときは,OMP_NUM_THREADS を 4 に設定して実行します.)

```
/* **EXAMPLE 1** */
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cssl.h"

#define  NRAN 10000000
#define  NSEED 123
#define  NBUF 25000

#define min(x,y) ((x)>(y)?(y):(x))

int MAIN__() {

  double  da[NBUF];
  double  dwork[8];
  double  dsum, dsum2;
  double  dmean;
  int  ix, n, icon;
  int  i, j;

/* Generate NRAN numbers with maximum NBUF at a time */
  ix = NSEED;
  printf(" Seed %d\n", ix);
  printf(" Generating %d numbers\n", NRAN);

  dsum = 0.0;
  for (j = 1; j <= NRAN; j += NBUF) {
    n = min(NBUF, NRAN - j + 1);
    c_dm_vranu5(&ix, da, n, (long)0, dwork, &icon);
```

```
    if (icon != 0) {
      printf(" Error return ICON %d\n", icon);
    }
    dsum2 = 0.0;
    for (i = 0; i < n; i++) {
      dsum2 += da[i];
    }
    dsum += dsum2;
  }
/* Compute mean */
  dmean = dsum / (double)NRAN;
  printf(" Mean %20.16lf\n", dmean);

  return(0);
}
```

使用例 2

MPI で並列化されたプログラムで, 4 つのプロセスでそれぞれ異なる 100,000 個の擬似一様乱数を生成し, 全体の平均値を計算します. 出発値は 123 とします.

このプログラムでは, j に 2^{31}-1 を与えています. 生成する疑似乱数が 1 プロセスあたり 2^{31}-1 個以下なら重複することはありません.

```
/* **EXAMPLE 2** */
 #include <stdlib.h>
 #include <stdio.h>
 #include <math.h>
 #include <mpi.h>
 #include "cssl.h"

 #define  N 10000

 int MAIN__(int argc, char *argv[]) {

   const long  jump = (long)2147483647;  /* =2**31-1 */
   double  x[N];
   double  dnall;
   int  irank, np;
   int  ix, icon;
   int  i;
   long  j;
   double  work[8];
   double  dsum, dsumall, dmean;

   MPI_Init(&argc, &argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &irank);
   MPI_Comm_size(MPI_COMM_WORLD, &np);

   ix = 123;
   j = irank * jump;
   c_dm_vranu5(&ix, x, N, j, work, &icon);
   if (icon != 0) {
     printf("C_DM_VRANU5 ERROR ICON= %d\n", icon);
   }
```

```
  dsum = 0.0;
  for (i = 0; i < N; i++) {
    dsum += x[i];
  }
  MPI_Reduce(&dsum, &dsumall, 1, MPI_DOUBLE, MPI_SUM, 0,
             MPI_COMM_WORLD);
/* Compute overall mean */
  dnall = (double)N * (double)np;
  if (irank == 0) {
    dmean = dsumall / dnall;
    printf(" Mean %19.16lf\n", dmean);
  }

  MPI_Finalize();
  return(0);
}
```

使用例 3

MPI で並列化されたプログラムで, 4 つのプロセスで合わせて 1,000,000 個の擬似一様乱数を 2 つ生成し, 全体の平均値を計算します. 出発値は 123 とします.

このプログラムでは 1,000,000 個の疑似乱数を全プロセスで等分割して生成します. プロセス数を変数 NP で指定しています. プロセス数を変更して実行する場合, NP の値を変更してください. このとき, 全プロセスで生成した 1,000,000 個の疑似乱数はプロセス数に関係なく同じ値になります.

```
/* **EXAMPLE 3** */
 #include <stdlib.h>
 #include <stdio.h>
 #include <math.h>
 #include "cssl.h"
 #include <mpi.h>

 #define  NX  100000
 #define  NY  100000
 #define  NP  4  /* NUMBER OF PROCESS */

 #define min(x,y) ((x)>(y)?(y):(x))

 int MAIN__(int argc, char *argv[]) {

   double  x[(NX + NP - 1) / NP], y[(NY + NP - 1) / NP];
   int  irank, nsize;
   int  ix, nl, icon, jump;
   int  i;
   long  j0, j;
   double  work[8];
   double  dsum, dsumall, dmean;

   MPI_Init(&argc, &argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &irank);
```

```
  MPI_Comm_size(MPI_COMM_WORLD, &nsize);
  if (NP != nsize) {
    MPI_Finalize();
    return(-1);
  }

  ix = 123;
  jump = (NX + NP - 1) / NP;
  j = min(irank * jump, NX);
  nl = min(jump, NX - j);
  if (nl >= 1) {
    c_dm_vranu5(&ix, x, nl, j, work, &icon);
    if (icon != 0) {
      printf("DM_VRANU5 ERROR ICON= %d\n", icon);
    }
    j0 = NX - (j + nl);
  } else {
    j0 = NX;
  }

  dsum = 0.0;
  for (i = 0; i < nl; i++) {
    dsum += x[i];
  }
  MPI_Reduce(&dsum, &dsumall, 1, MPI_DOUBLE, MPI_SUM, 0,
             MPI_COMM_WORLD);

/* Compute overall mean of X */
  if (irank == 0) {
      dmean = dsumall / (double)NX;
      printf(" Mean of X %19.16lf\n", dmean);
  }

  jump = (NY + NP - 1) / NP;
  j = min(irank * jump, NY);
  nl = min(jump, NY - j);
  j += j0;
  if (nl >= 1) {
    c_dm_vranu5(&ix, y, nl, j, work, &icon);
    if (icon != 0) {
      printf("C_DM_VRANU5 ERROR ICON= %d\n", icon);
    }
  }

  dsum = 0.0;
  for (i = 0; i < nl; i++) {
    dsum += y[i];
  }
  MPI_Reduce(&dsum, &dsumall, 1, MPI_DOUBLE, MPI_SUM, 0,
```

```
                MPI_COMM_WORLD);

  /* Compute overall mean of Y */
    if (irank == 0) {
      dmean = dsumall / (double) NY;
      printf(" Mean of Y %19.16lf\n", dmean);
    }

    MPI_Finalize();
    return(0);
  }
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VRANU5 の項目及び[82], [83], [84]を参照してください.

c_dm_vschol

正値対称スパース行列の LDL^T 分解 (Left-looking な方法)

`ierr = c_dm_vschol(a, nz, nrow, nfcnz, n,`
`iordering, nperm, isw, &epsz,`
`nassign, &nsupnum, nfcnzfactor,`
`panelfactor, &nsizefactor,`
`nfcnzindex, npanelindex,`
`&nsizeindex, ndim, nposto, w, iw1,`
`iw2, iw3, &icon);`

1. 機能

n 次の正値対称スパース行列 **A** を, 変形コレスキー分解法により LDL^T 分解します.

$$\mathbf{QPAP^TQ^T=LDL^T}$$

ただし, **P** は ordering による行列要素の並び換えを示す置換行列, **Q** は post order による行列要素の並び換えを示す置換行列を示します. **P**, **Q** は直交行列です.
L は単位下三角行列, **D** は対角行列です.

2. 引数

呼出し形式:

```
ierr = c_dm_vschol(a, nz, nrow, nfcnz, n, iordering, nperm, isw, &epsz,
            nassign, &nsupnum,nfcnzfactor, panelfactor, &nsizefactor,
            nfcnzindex, npanelindex, &nsizeindex,(int*)ndim, nposto, w, iw1,
            iw2, iw3, &icon);
```

引数の説明:

a	double a[nz]	入力	係数行列 **A** のスパースな正値対称行列の下三角行列部分 $\{a_{ij}\|i\geq j\}$ を圧縮列格納法で a[*i*], *i*=0,…,nz-1 に格納します. 圧縮列格納法については, c_dm_vmvscc の図 c_dm_vmvscc-1 参照.
nz	int	入力	係数行列 **A** のスパースな正値対称行列の下三角行列部分にある非零要素の総数.
nrow	int nrow[nz]	入力	圧縮列格納法で使用される行指標で **A** に格納される要素が何番目の行ベクトルに属するかを示します.
nfcnz	int nfcnz[n+1]	入力	行列 **A** の各列の非零要素を列方向に圧縮して順次配列 a に格納するとき,対応する列の最初の非零要素が格納される位置を表します. nfcnz[n]=nz+1.
n	int	入力	行列 **A** の次数 *n*.
iordering	int	入力	ordering を表す置換行列 **P** で $\mathbf{PAP^T}$ と変換した行列を LDL^T 分解するかを指定します. 置換行列は直交行列です. 1 のとき : $\mathbf{PAP^T}$ と変換したものを LDL^T 分解します. 1 以外のとき : 行列 **A** をそのまま LDL^T 分解します.
nperm	int nperm[n]	入力	iordering=1 のとき使用する置換行列をベクトルで指定

			します. (“使用上の注意” a)参照)
isw	int	入力	呼び出しに関する制御情報を示します. 1　初回の呼び出し. 2　1 回目の呼び出しでは panelfactor または npanelindex の大きさが不足していた. icon=31000 で終了した.このとき, nsizefactor,または nsizeindex に返却された必要な大きさで panelfactor または npanelindex を確保し直して再度呼び出しします. さらに a, nz, nrow, nfcnz, n, iordering, nperm, nassign, nsupnum, nfcnzfactor, nfcnzindex, npanelindex, nposto, ndim, w, iw1, iw2, iw3 に格納されている値を変更してはなりません. 3　同じ非零パターンを持つ次数の同じ行列で,行列要素の値が異なる行列に対して symbolic decomposition の解析結果や必要な大きさが同じになる配列 panelfactor, npanelindex を再利用して LDL^T 分解することを指定します.行列の値を配列要素に格納し直して呼び出します. このとき, nrow の値を変えずに配列 a に行列要素の値を格納し直すか, 別の配列 b に格納し引数 a として受け渡さなければなりません. さらに nz, nrow, nfcnz, n, iordering, nperm, nassign, nsupnum, nfcnzfactor, nsizefactor, nfcnzindex, npanelindex, nsizeindex, nposto, ndim, w, iw1, iw2, iw3 に格納されている値を変更してはなりません.
epsz	double	入力	ピボットの相対判定値 (≥0.0)
		出力	0.0 のときは標準値が設定されます. (“使用上の注意” b)参照)
nassign	int nassign[n]	出力	各 supernode は複数の列ベクトルからなり,分解結果に関して共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します.この panel を panelfactor の 1 次元部分配列として順番に割り付けたとき何番目になるかを示します. j=nassign[i-1]のとき, *i* 番目の supernode を *j* 番目に割り付けたことを示します.
		入力	isw≠1 のとき,初回呼び出しの値を再利用します. 結果の格納方法については図 c_dm_vschol-1 を参照. (“使用上の注意” c)参照)
nsupnum	int	出力	supernode の総数.
		入力	isw≠1 のとき,初回呼び出しの値を再利用します. (≤n)
nfcnzfactor	long long int nfcnzfactor [n+1]	出力	各 supernode は複数の列ベクトルからなり,分解結果に関して共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します.この panel を panelfactor の 1 次元部分配列として順番に割り付けたとき, *i* 番目の panel の先頭要素 panel[0][0]が 1 次元配列の panelfactor の何番目の要素になるかを示します. 結果の格納方法については図 c_dm_vschol-1 を参照.
		入力	isw≠1 のとき,初回呼び出しの値を再利用します.

panelfactor	double panelfactor [nsizefactor]	出力	各 supernode は複数の列ベクトルからなり,分解結果に関して共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します. panel を順番に割り付けます. *i* 番目の supernode に対応する panel が何番目の位置に割り当てられるかは j=nassign[i-1]から分かります.その先頭位置は nfcnzfactor[j-1]に格納されています. panel ごとに分解結果が格納されます. *i* 番目に割付けられた panel の大きさは ndim[i-1][1] ×ndim[i-1][0]の 2 次元配列と見なせます. *i* 番目の panel の panel[t-1][s-1], s>t, s=1,…,ndim[i-1][0], t=1,…,ndim[i-1][1]に単位下三角行列 **L** の対角要素を除いた対応部分が転置した形で格納されます.対角部分 panel[t-1][t-1]には対角行列 **D** 対応部分が格納されます. 結果の格納方法については図 c_dm_vschol-1 を参照. ("使用上の注意" d)参照)
nsizefactor	long long int	入力	panelfactor の大きさを示す. ("使用上の注意" d)参照)
		出力	panelfactor の大きさとして必要な大きさが返却されます.
nfcnzindex	long long int nfcnzindex [n+1]	出力	各 supernode は複数の列ベクトルからなり,分解結果に関して共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します.この行指標ベクトルを npanelindex に順番に割り付けたとき *i* 番目の行指標ベクトルの先頭要素が 1 次元配列である npanelindex の何番目の要素になるかを示します.
		入力	isw≠1 のとき,初回呼び出しの値を再利用します. 結果の格納方法については図 c_dm_vschol-1 を参照.
npanelindex	int npanelindex [nsizeindex]	出力	各 supernode は複数の列ベクトルからなり,分解結果に関して共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します.この行指標ベクトルを npanelindex に順番に割り付けます. *i* 番目の supernode に対応する panel の行の指標ベクトルが何番目に割り当てられるかは j=nassign[i-1]から分かります.その先頭位置は nfcnzindex[j-1]に格納されています. panel ごとの行の指標ベクトルが格納されます. この行指標は行列 **A** を post order で並び換えた行列 $\mathbf{QAQ}^{\mathrm{T}}$ での行の番号です. 結果の格納方法については図 c_dm_vschol-1 を参照. ("使用上の注意" d)参照)
nsizeindex	long long int	入力	npanelindex の大きさを示す. ("使用上の注意" d)参照)
		出力	必要な大きさが返却されます.
ndim	int ndim[n][2]	出力	ndim[i-1][0]および ndim[i-1][1] は *i* 番目に割り付けられた panel の 1 次元目と 2 次元目の大きさを示します.
		入力	isw≠1 のとき,初回呼び出しの値を再利用します. 結果の格納方法については図 c_dm_vschol-1 を参照.
nposto	int nposto[n]	出力	post order で *i* 番目の node が行列 **A** の何番目の列番号に対応するかを表す 1 次元ベクトル.
		入力	isw≠1 のとき,初回呼び出しの値を再利用します.

			(“使用上の注意” e)参照)
w	double w[*Iwlen1*]	作業域 出力/入力	isw=1, 2, 3 で続けて呼び出すとき,呼び出しの間で必要なデータを受け渡すために使われます.呼び出しの間で値を変更してはいけません. 1) iordering=1 のとき *Iwlen1*=nz 2) iordering≠1 のとき *Iwlen1*=1
iw1	int iw1[*Iwlen2*]	作業域 出力/入力	isw=1, 2, 3 で続けて呼び出すとき,呼び出しの間で必要なデータを受け渡すために使われます.呼び出しの間で値を変更してはいけません. 1) iordering=1 のとき *Iwlen2*=nz+n+1 2) iordering≠1 のとき *Iwlen2*=1
iw2	int iw2[nz+n+1]	作業域 出力/入力	isw=1, 2, 3 で続けて呼び出すとき,呼び出しの間で必要なデータを受け渡すために使われます.呼び出しの間で値を変更してはいけません.
iw3	int iw3[n*35+35]	作業域 出力/入力	isw =1, 2, 3 で続けて呼び出すとき,呼び出しの間で必要なデータを受け渡すために使われます.呼び出しの間で値を変更してはいけません.
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.
20000	ピボットが相対的にゼロになった. 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 • nz < 0 • nfcnz[n]≠nz+1 • nsizefactor<1 • nsizeindex<1 • epsz<0.0 • isw<0 • isw>3	
30100	nperm に指定した置換行列が正しくない.	
30200	nrow[j-1]に格納された *i* 列目の行指標 k が k<i または k>n.	
30300	*i* 列目の行指標の数 nfcnz[i]-nfcnz[i-1]>n-i+1	
30400	対角要素が格納されていない列がある.	
31000	panelfactor の大きさ nsizefactor または npanelindex の大きさ nsizeindex が小さすぎる.	nsizefactor または nsizeindex で指定された 大きさで panelfactor または npanelindex を割り付けて isw=2 として再度呼び出す.

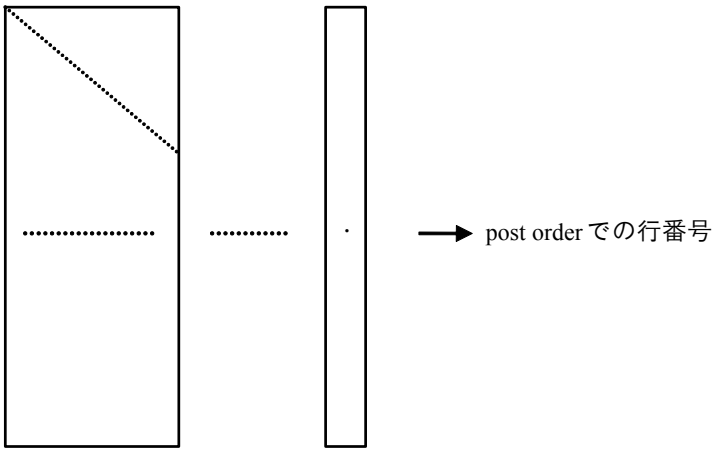


図 c_dm_vschol-1 分解結果の格納概念図

`j = nassign[i-1]` → *i* 番目の supernode は *j* 番目に格納されます.
`p = nfcnzfactor[j-1]` → *j* 番目の `panel` は `panelfactor` の *p* 番目の要素から `ndim[j-1][1]`×`ndim[j-1][0]`の長さを占めます.
`q = nfcnzindex[j-1]` → *j* 番目の `panel` の行指標を表すベクトルは `npanelindex` の *q* 番目の要素から `ndim[j-1][0]`の長さを占めます.

`panel` は大きさ `ndim[j-1][1]`×`ndim[j-1][0]`の配列と見なせます.
`panel[t-1][s-1]` , `s > t` , `s=1` , … , `ndim[j-1][0]`,
`t=1` , … , `ndim[j-1][1]`
に分解結果の単位下三角行列 **L** の対角要素を除いた部分が転置した形で格納されます.
`panel[t-1][t-1]`に対角行列 **D** の対応部分が格納されます.
行指標の値は行列 **A** の列番号をもつ node を post order で並べ換えた $\mathbf{QAQ}^{\mathrm{T}}$ での列番号を表します.

3. 使用上の注意

a) `nperm` について

直交行列である置換行列 **P** の要素 $\mathbf{p_{ij}=1}$ のとき, `nperm[i-1]=j` と表現します.
逆は以下のようにして求めることができます.

```
for(i=1; i<=n; i++){
   j=nperm[i-1];
   nperminv[j-1]=i;
}
```

b) `epsz`

ピボットの相対零判定値にある値を設定したとすると, この値は次の意味を持っています. すなわち, 変形コレスキー分解法による $\mathrm{LDL}^{\mathrm{T}}$ 分解の過程で, ピボットの絶対値が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見なし, `icon`=20000 として処理を打ち切ります. `epsz` の標準値は, 丸め誤差の単位を u としたとき, `epsz`=16u です.

なお, ピボットの絶対値が小さくなっても, 処理を続行させたい場合には, `epsz` に極小の値を与えれば良いのですが, 結果の精度は保証されません.

分解の途中でピボットが負となった場合, 係数行列は正値ではありません. このとき, icon=10000 として処理は続行します. ただし, ピボッティングを行っていないため, 計算誤差は大きい可能性があります.

c) c_dm_vscholx について

分解結果に関するデータを nassign, nsupnum, nfcnzfactor, nsizefactor, nfcnzindex, npanelindex, nsizeindex, nposto, ndim, iw3 などをルーチン c_dm_vscholx に受け渡して引き続き呼び出すことで,連立 1 次方程式 **Ax=b** を変形した $\mathbf{LDL^TPQx=PQb}$ を解くことができます.

d) nsizefactor と nsizeindex について

分解結果を格納する配列 panelfactor, npanelindex の必要な大きさは,事前には分かりません.十分大きな配列を割り当てるか,本ルーチンを呼び出して symbolic decomposition を行った解析の結果を使って割り当てを行うことができます.

例えば,大きさ 1 の 1 次元配列などを割付けます.そしてその大きさ 1 などの小さな値を nsizefactor, nsizeindex に指定して, isw=1 で呼び出します.

symbolic decomposition を行い, icon=31000 で終了し, nsizefactor と nsizeindex に必要な大きさが返却されます.必要な大きさの配列を割付け直して, isw=2 で呼び出すことで, symbolic decomposition 以降の処理を続けることができます.

e) nposto について

列番号に対応するノードを考えます.これを post order で順番を並び換えたときの番号の対応関係が nposto に格納されています. post order の *i* 番目のノードが並び換える前の何番目のノードに対応するかを表します. j=nposto[i-1]は *j* 番目であることを表します.

a)と同様にこれは直交行列である置換行列 **Q** を表し,行列 **A** を $\mathbf{QAQ^T}$ と並び換えることに相当します. 逆変換 $\mathbf{Q^T}$ は以下のようにして求めることができます.

```
for(i=1; i<=n; i++){
    j= nposto [i-1];
    npostoinv [j-1]=i;
}
```

4. 使用例

連立 1 次方程式 **Ax = f** を解きます. 行列 **A** は境界条件として立方体の境界で 0 となる楕円型の偏微分方程式に有限差分法を適用して生成されます.

$$-\Delta u + a\nabla u + cu = f$$

ここで $a = (a_1, a_2, a_3)$, a_1, a_2, a_3 および c はゼロで,ラプラシアンになり行列 **A** は正値対称です.行列 **A** は関数 init_mat_diag によって生成されます. これを圧縮列格納法に変換します. (並列実行を行うスレッド数は環境変数(OMP_NUM_THREADS)などで指定できます. 例えば, 4 プロセッサーのシステムで 4 つのスレッドで並列実行するときは, OMP_NUM_THREADS を 4 に設定して実行します.)

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

#define NORD   (39)
#define NX     (NORD)
#define NY     (NORD)
#define NZ     (NORD)
#define N      (NX*NY*NZ)
```

```
#define K      (N+1)
#define NDIAG  (7)
#define NDIAGH (4)

MAIN__()
{
  int    ierr, icon, iguss, iter, itmax;
  int    nord, n, l, i, j, k;
  int    nx, ny, nz, nnz, nnzc;
  int    length, nbase, ndiag, ntopcfgc;
  int    numnz, numnzc, nsupnum, ntopcfg, ncol;
  int    iordering, isw;
  int    *npanelindex;
  int    ndummyi;
  int    nofst[NDIAG];
  int    nrow[NDIAG*K];
  int    nrowc[NDIAG*K];
  int    nfcnz[N+1];
  int    nfcnzc[N+1];
  int    nperm[N];
  int    nassign[N];
  int    nposto[N];
  int    ndim[N][2];
  int    iw1[N*NDIAGH+N+1];
  int    iw2[N*NDIAGH+N+1];
  int    iw3[N*35+35];
  int    iwc[NDIAG*K][2];

  double err, epsz;
  double t0, t1, t2;
  double va1, va2, va3, vc;
  double xl, yl, zl;
  double dummyf;
  double *panelfactor;
  double diag[NDIAG][K];
  double diag2[NDIAG][K];
  double a[N*NDIAGH];
  double b[N];
  double c[NDIAG*K];
  double w[N*NDIAGH];
  double wc[NDIAG*K];
  double x[N];
  double solex[N];

  long long int nsizefactor;
  long long int nsizeindex;
  long long int nfcnzfactor[N+1];
  long long int nfcnzindex[N+1];

  void init_mat_diag(double va1, double va2, double va3, double vc,
              double d_l[], int offset[], int nx, int ny, int nz,
              double xl, double yl,double zl, int ndiag, int len, int ndivp);

  double errnrm(double *x1, double *x2, int len);

  nord=NORD, nx=NX, ny=NY, nz=NZ, n=N, k=K, ndiag=NDIAG;

  printf("     LEFT-LOOKING MODIFIED CHOLESKY METHOD\n");
  printf("     FOR SPARSE POSITIVE DEFINITE MATRICES\n");
  printf("     IN COMPRESSED COLUMN STORAGE\n");
  printf("\n");

  for (i=1; i<=n; i++){
    solex[i-1]=1.0;
  }
  printf("     EXPECTED SOLUTIONS\n");
  printf("     X(1) = %.15lf  X(N) = %.15lf\n", solex[0], solex[n-1]);
  printf("\n");

  va1 = 0.0;
  va2 = 0.0;
  va3 = 0.0;
  vc = 0.0;
  xl = 1.0;
  yl = 1.0;
  zl = 1.0;
  init_mat_diag(va1, va2, va3, vc, (double*)diag, (int*)nofst,
```

```
                        nx, ny, nz, xl, yl, zl, ndiag, n, k);

        for (i=1; i<=ndiag; i++){
          if (nofst[i-1] < 0){
            nbase=-nofst[i-1];
            length=n-nbase;
            for (j=1; j<=length; j++){
              diag2[i-1][j-1]=diag[i-1][nbase+j-1];
            }
          }
          else{
            nbase=nofst[i-1];
            length=n-nbase;
            for (j=nbase+1; j<=n; j++){
              diag2[i-1][j-1]=diag[i-1][j-nbase-1];
            }
          }
        }

        numnzc=1;
        numnz=1;
        for (j=1; j<=n; j++){
          ntopcfgc = 1;
          ntopcfg = 1;
          for (i=ndiag; i>=1; i--){
            if (diag2[i-1][j-1]!=0.0){
              ncol=j-nofst[i-1];
              c[numnzc-1]=diag2[i-1][j-1];
              nrowc[numnzc-1]=ncol;
              if (ncol>=j){
                a[numnz-1]=diag2[i-1][j-1];
                nrow[numnz-1]=ncol;
              }
              if (ntopcfgc==1){
                nfcnzc[j-1]=numnzc;
                ntopcfgc=0;
              }
              if (ntopcfg==1){
                nfcnz[j-1]=numnz;
                ntopcfg=0;
              }
              if (ncol>=j){
                numnz=numnz+1;
              }
              numnzc=numnzc+1;
            }
          }
        }


        nfcnzc[n]=numnzc;
        nnzc=numnzc-1;
        nfcnz[n]=numnz;
        nnz=numnz-1;

        ierr=c_dm_vmvscc(c, nnzc, nrowc, nfcnzc, n, solex, b, wc, (int*)iwc, &icon);

        for(i=1; i<=n; i++){
           x[i-1]=b[i-1];
        }
        iordering=0;
        isw=1;
        epsz=0;
        nsizefactor=1;
        nsizeindex=1;

        ierr=c_dm_vschol(a, nnz, nrow, nfcnz, n, iordering, nperm, isw, &epsz, nassign,
&nsupnum, nfcnzfactor, &dummyf, &nsizefactor, nfcnzindex, &ndummyi, &nsizeindex,
(int*)ndim, nposto, w, iw1, iw2, iw3, &icon);

        printf("\n");
        printf("     ICON = %d  NSIZEFACTOR = %lld NSIZEINDEX = %lld\n", icon,
nsizefactor, nsizeindex);
        printf("\n");

        panelfactor = (double *)malloc(sizeof(double)*nsizefactor);
        npanelindex = (int *)malloc(sizeof(int)*nsizeindex);
```

```
        isw=2;

        ierr=c_dm_vschol(a, nnz, nrow, nfcnz, n, iordering, nperm, isw, &epsz, nassign,
&nsupnum, nfcnzfactor, panelfactor, &nsizefactor, nfcnzindex, npanelindex, &nsizeindex,
(int*)ndim, nposto, w, iw1, iw2, iw3, &icon);

        ierr=c_dm_vscholx(n, iordering, nperm, x, nassign, nsupnum, nfcnzfactor,
panelfactor, nsizefactor, nfcnzindex, npanelindex, nsizeindex,
                          (int*)ndim, nposto, iw3, &icon);

        err = errnrm(solex,x,n);

        printf("     COMPUTED VALUES\n");
        printf("     X(1) = %.15lf  X(N) = %.15lf\n", x[0], x[n-1]);
        printf("\n");
        printf("     ICON = %d\n", icon);
        printf("\n");
        printf("     N = %d  :: NX = %d  NY = %d  NZ = %d\n",n,nx,ny,nz);
        printf("\n");
        printf("     ERROR = %.15e\n",err);
        printf("\n");
        printf("\n");
        if (err<(1.0e-8) && icon==0){
          printf("     ********** OK **********\n");
        }
        else{
          printf("     ********** NG **********\n");
        }
          free(panelfactor);
          free(npanelindex);
          return 0;
      }

      void init_mat_diag(double va1, double va2, double va3, double vc,
               double d_l[], int offset[], int nx, int ny, int nz,
               double xl, double yl, double zl, int ndiag, int len, int ndivp)
      {
        int i, l, j;
        int length, numnz, js;
        int i0, j0, k0;
        int ndiag_loc;
        int nxy;

        double hx, hy, hz;
        double x1, x2;
        double base;
        double ret, remark;

        if (ndiag<1){
          printf("FUNCTION INIT_MAT_DIAG:\n");
          printf("NDIAG SHOULD BE GREATER THAN OR EQUAL TO 1\n");
          return;
        }
        ndiag_loc = ndiag;
        if (ndiag>7){
          ndiag_loc=7;
        }

        hx = xl / (nx + 1);
        hy = yl / (ny + 1);
        hz = zl / (nz + 1);

        for (i=1; i<=ndivp; i++){
          for (j=1; j<=ndiag; j++){
            d_l[i-1+(j-1)*ndivp]= 0.;
          }
        }

        nxy = nx * ny;
        l = 1;
        if (ndiag_loc >= 7) {
          offset[l-1] = -nxy;
          ++l;
        }
        if (ndiag_loc >= 5) {
          offset[l-1] = -nx;
          ++l;
```

```
}
if (ndiag_loc >= 3) {
  offset[l-1] = -1;
  ++l;
}
offset[l-1] = 0;
++l;
if (ndiag_loc >= 2) {
  offset[l-1] = 1;
  ++l;
}
if (ndiag_loc >= 4) {
  offset[l-1] = nx;
  ++l;
}
if (ndiag_loc >= 6) {
  offset[l-1] = nxy;
}

for (j = 1; j <= len; ++j) {
  js=j;
  k0 = (js - 1) / nxy + 1;
  if (k0 > nz) {
    printf("ERROR; K0.GH.NZ\n");
    return;
  }
  j0 = (js - 1 - nxy * (k0 - 1)) / nx + 1;
  i0 = js - nxy * (k0 - 1) - nx * (j0 - 1);

  l = 1;
  if (ndiag_loc >= 7) {
    if (k0 > 1) {
      d_l[j-1+(l-1)*ndivp] = -(1.0/hz+va3*0.5)/hz;
    }
    ++l;
  }

  if (ndiag_loc >= 5) {
    if (j0 > 1) {
      d_l[j-1+(l-1)*ndivp] = -(1.0/hy+va2*0.5)/hy;
    }
    ++l;
  }

  if (ndiag_loc >= 3) {
    if (i0 > 1) {
      d_l[j-1+(l-1)*ndivp] = -(1.0/hx+va1*0.5)/hx;
    }
    ++l;
  }

  d_l[j-1+(l-1)*ndivp] = 2.0/(hx*hx)+vc;
  if (ndiag_loc >= 5) {
    d_l[j-1+(l-1)*ndivp] += 2.0/(hy*hy);
    if (ndiag_loc >= 7) {
      d_l[j-1+(l-1)*ndivp] += 2.0/(hz*hz);
    }
  }
  ++l;
  if (ndiag_loc >= 2) {
    if (i0 < nx) {
      d_l[j-1+(l-1)*ndivp] = -(1.0/hx-va1*0.5)/hx;
    }
    ++l;
  }

  if (ndiag_loc >= 4) {
    if (j0 < ny) {
      d_l[j-1+(l-1)*ndivp] = -(1.0/hy-va2*0.5)/hy;
    }
    ++l;
  }

  if (ndiag_loc >= 6) {
    if (k0 < nz) {
      d_l[j-1+(l-1)*ndivp] = -(1.0/hz-va3*0.5)/hz;
    }
```

```
    }
  }
  return;
}

double errnrm(double *x1, double *x2, int len)
{
  double ret_val;

  int i;
  double s, ss;

  s = 0.;
  for (i = 1; i <= len; ++i) {
    ss = x1[i-1] - x2[i-1];
    s += ss * ss;
  }
  ret_val = sqrt(s);
  return ret_val;
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VSCHOL の項目及び[19]を参照してください.

c_dm_vscholx

LDL^T分解された正値対称スパース行列の連立 1 次方程式
ierr = c_dm_vscholx(n, iordering, nperm, b, nassign, nsupnum, nfcnzfactor, panelfactor, nsizefactor, nfcnzindex, npanelindex, nsizeindex,ndim, nposto, iw3, &icon);

1. 機能

変形コレスキー分解法によりLDL^T分解された$n \times n$の正値対称スパース行列の連立1次方程式を解きます.

$$\mathbf{LDL^TQPx=QPb}$$

ただし, **P**はorderingによる行列要素の並び換えを示す置換行列, **Q**はpost orderによる行列要素の並び換えを示す置換行列を示します. **P**,**Q**は直交行列です.
Lは単位下三角行列, **D**は対角行列です.
bは右辺定数ベクトル, **x**は解ベクトルです.

2. 引数

呼出し形式:

```
ierr = c_dm_vscholx(n, iordering, nperm, b, nassign, nsupnum, nfcnzfactor,
            panelfactor, nsizefactor, nfcnzindex, npanelindex, nsizeindex,
           (int*)ndim, nposto, iw3, &icon);
```

引数の説明:

n	int	入力	行列 A の次数 *n*.
iordering	int	入力	LDL^T 分解を行うときに, ordering を表す直交行列である置換行列 **P** で $\mathbf{PAP^T}$ と変換したかを示します. 1 のとき, $\mathbf{PAP^T}$ と変換したものを LDL^T 分解した. 1 以外のとき,行列 **A** をそのまま LDL^T 分解した.
nperm	int nperm[n]	入力	iordering=1 のとき使用した置換行列をベクトルで指定します. (“使用上の注意” a)参照)
b	double b[n]	入力	**Ax=b** の右辺定数ベクトル.
		出力	解ベクトル.
nassign	int nassign[n]	入力	各 supernode は複数の列ベクトルからなり,分解結果に関して共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します.この panel を panelfactor の 1 次元部分配列として順番に割り付けたとき何番目になるかを示します. j=nassign[i-1]のとき, *i* 番目の supernode を *j* 番目に割り付けたことを示します. 分解結果の格納方法については図 c_dm_vscholx-1 を参照.
nsupnum	int	入力	supernode の総数.
nfcnzfactor	long long int nfcnzfactor [n+1]	入力	各 supernode は複数の列ベクトルからなり,分解結果に関して共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します.この panel を panelfactor の 1 次元部分配列として順番に割り付けたとき, *i* 番目の panel の先頭要素

			panel[0][0]が 1 次元配列の panelfactor の何番目の要素になるかを示します. 分解結果の格納方法については図 c_dm_vscholx-1 を参照. (“使用上の注意” c)参照)
panelfactor	double panelfactor [nsizefactor]	入力	各 supernode は複数の列ベクトルからなり,分解結果に関して共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します. panel を順番に割り付けます. *i* 番目の supernode に対応する panel が何番目の位置に割り当てられるかは j=nassign[i-1]から分かります.その先頭位置は nfcnzfactor[j-1]に格納されています. panel ごとに分解結果が格納されます. *i* 番目に割付けられた panel の大きさは ndim[i-1][1]× ndim[i-1][0]の 2 次元配列と見なせます. *i* 番目の panel の panel[t-1][s-1], s>t,s=1,…,ndim[i-1][0], t=1,…,ndim[i-1][1]に単位下三角行列 **L** の対角要素を除いた対応部分が転置した形で格納されます.対角部分 panel[t-1][t-1]には対角行列 **D** 対応部分を格納します. 分解結果の格納方法については図 c_dm_vscholx-1 を参照.
nsizefactor	long long int	入力	panelfactor の大きさを示す.
nfcnzindex	long long int nfcnzindex [n+1]	入力	各 supernode は複数の列ベクトルからなり,分解結果に関して共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します.この行指標ベクトルを npanelindex に順番に割り付けたとき *i* 番目の行指標ベクトルの先頭要素が 1 次元配列である npanelindex の何番目の要素になるかを示します. 分解結果の格納方法については図 c_dm_vscholx-1 を参照.
npanelindex	int npanelindex [nsizeindex]	入力	各 supernode は複数の列ベクトルからなり,分解結果に関して共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します.この行指標ベクトルを npanelindex に順番に割り付けます. *i* 番目の supernode に対応する panel の行の指標ベクトルが何番目に割り当てられるかは j=nassign[i-1]から分かります.その先頭位置は nfcnzindex[j-1]に格納されています. panel ごとの行の指標ベクトルを格納します. この行指標は行列 **A** を post order で並び換えた行列 $\mathbf{QAQ}^{\mathrm{T}}$ での行の番号です. 結果の格納方法については図 c_dm_vscholx-1 を参照.
nsizeindex	long long int	入力	npanelindex の大きさを示す.
ndim	int ndim[n][2]	入力	ndim[i-1][0]および ndim[i-1][1] は *i* 番目に割り付けられた panel の 1 次元目と 2 次元目の大きさを示します. 分解結果の格納方法については図 c_dm_vscholx-1 を参照.
nposto	int nposto[n]	入力	post order で *i* 番目の node が行列 **A** の何番目の列番号に対応するかを表す 1 次元ベクトル. (“使用上の注意” b)参照)
iw3	int iw3[n*35+35]	入力	ルーチン c_dm_vschol の本ルーチンの呼び出しに先立つ呼び出しで使った作業域 iw3 の内容を変更せず指定します.
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	次のいずれかであった. • n < 1 • nsizefactor<1 • nsizeindex<1 • nsupnum<1	処理を打ち切る.
30100	nperm に指定した置換行列が正しくない.	

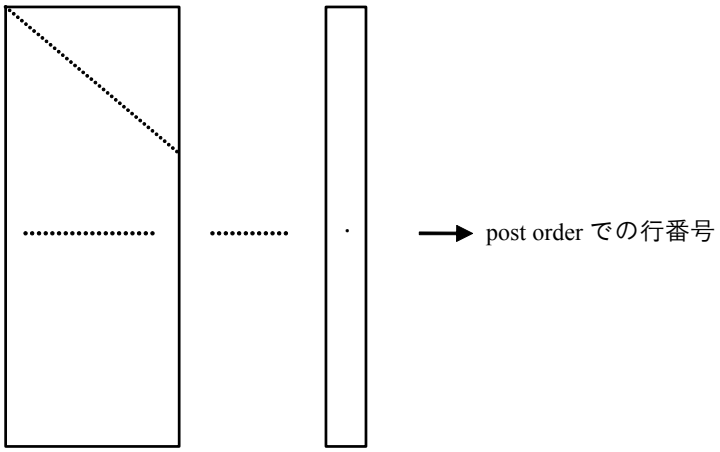


図 c_dm_vscholx-1 分解結果の格納概念図

```
j = nassign[i-1]      → i番目の supernode は j 番目に格納されます.
p = nfcnzfactor[j-1] → j番目の panel は panelfactor の p 番目の要素から
                        ndim[j-1][1]×ndim[j-1][0]の長さを占めます.
q = nfcnzindex[j-1]  → j番目の panel の行指標を表すベクトルは
                        npanelindex の q 番目の要素から ndim[j-1][0]
                        の長さを占めます.
```

panel は大きさ ndim[j-1][1]×ndim[j-1][0]の配列と見なせます.
panel[t-1][s-1] , s > t , s=1 , … , ndim[j-1][0],
t=1 , … , ndim[j-1][1]
に分解結果の単位下三角行列 **L** の対角要素を除いた部分が転置した形で格納されます.
panel[t-1][t-1]に対角行列 **D** の対応部分が格納されます.
行指標の値は行列 **A** の列番号をもつ node を post order で並べ換えた $\mathbf{QAQ}^{\mathrm{T}}$ での列番号を表します.

3. 使用上の注意

a) nperm について

直交行列である置換行列 **P** の要素 p_{ij}=**1** のとき, nperm[i-1]=j と表現します.
逆は以下のようにして求めることができます.

```
for(i=1; i<=n; i++){
   j=nperm[i-1];
   nperminv[j-1]=i;
}
```

b) nposto について

列番号に対応するノードを考えます.これを post order で順番を並び換えたときの番号の対応関係が nposto に格納されています. post order の *i* 番目のノードが並び換える前の何番目のノードに対応するかを表します. j=nposto[i-1]は *j* 番目であることを表します.

a)と同様にこれは直交行列である置換行列 **Q** を表し,行列 **A** を $\mathbf{QAQ}^T$ と並び換えることに相当します. 逆変換 $\mathbf{Q}^T$は以下のようにして求めることができます.

```
for(i=1; i<=n; i++){
   j=nposto[i-1];
   npostoinv[j-1]=i;
}
```

c) 連立 1 次方程式

LDL^T分解の結果データは, c_dm_vschol を呼び出した結果を指定して本ルーチンを呼び出すことで連立 1 次方程式を解くことができます.

4. 使用例

連立 1 次方程式 **Ax** = **f** を解きます. 行列 **A** は境界条件として立方体の境界で 0 となる楕円型の偏微分方程式に有限差分法を適用して生成されます.

$$-\Delta u + a\nabla u + cu = f$$

ここで $a = (a_1, a_2, a_3)$, a_1, a_2, a_3および c はゼロで,ラプラシアンになり行列 **A** は正値対称です.行列 **A** は関数 init_mat_diag によって生成されます. これを圧縮列格納法に変換します.

(並列実行を行うスレッド数は環境変数(OMP_NUM_THREADS)などで指定できます. 例えば, 4 プロセッサーのシステムで 4 つのスレッドで並列実行するときは, OMP_NUM_THREADS を 4 に設定して実行します.)

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

#define NORD   (39)
#define NX     (NORD)
#define NY     (NORD)
#define NZ     (NORD)
#define N      (NX*NY*NZ)
#define K      (N+1)
#define NDIAG  (7)
#define NDIAGH (4)

MAIN__()
{
  int    ierr, icon, iguss, iter, itmax;
```

```
int    nord, n, l, i, j, k;
int    nx, ny, nz, nnz, nnzc;
int    length, nbase, ndiag, ntopcfgc;
int    numnz, numnzc, nsupnum, ntopcfg, ncol;
int    iordering, isw;
int    *npanelindex;
int    ndummyi;
int    nofst[NDIAG];
int    nrow[NDIAG*K];
int    nrowc[NDIAG*K];
int    nfcnz[N+1];
int    nfcnzc[N+1];
int    nperm[N];
int    nassign[N];
int    nposto[N];
int    ndim[N][2];
int    iw1[N*NDIAGH+N+1];
int    iw2[N*NDIAGH+N+1];
int    iw3[N*35+35];
int    iwc[NDIAG*K][2];

double err, epsz;
double t0, t1, t2;
double va1, va2, va3, vc;
double xl, yl, zl;
double dummyf;
double *panelfactor;
double diag[NDIAG][K];
double diag2[NDIAG][K];
double a[N*NDIAGH];
double b[N];
double c[NDIAG*K];
double w[N*NDIAGH];
double wc[NDIAG*K];
double x[N];
double solex[N];

long long int nsizefactor;
long long int nsizeindex;
long long int nfcnzfactor[N+1];
long long int nfcnzindex[N+1];

void init_mat_diag(double va1, double va2, double va3, double vc,
            double d_l[], int offset[], int nx, int ny, int nz,
            double xl, double yl,double zl, int ndiag, int len, int ndivp);

double errnrm(double *x1, double *x2, int len);

nord=NORD, nx=NX, ny=NY, nz=NZ, n=N, k=K, ndiag=NDIAG;

printf("     LEFT-LOOKING MODIFIED CHOLESKY METHOD\n");
printf("     FOR SPARSE POSITIVE DEFINITE MATRICES\n");
printf("     IN COMPRESSED COLUMN STORAGE\n");
printf("\n");

for (i=1; i<=n; i++){
  solex[i-1]=1.0;
}
printf("     EXPECTED SOLUTIONS\n");
printf("     X(1) = %.15lf  X(N) = %.15lf\n", solex[0], solex[n-1]);
printf("\n");

va1 = 0.0;
va2 = 0.0;
va3 = 0.0;
vc = 0.0;
xl = 1.0;
yl = 1.0;
zl = 1.0;
init_mat_diag(va1, va2, va3, vc, (double*)diag, (int*)nofst,
                    nx, ny, nz, xl, yl, zl, ndiag, n, k);

for (i=1; i<=ndiag; i++){
  if (nofst[i-1] < 0){
    nbase=-nofst[i-1];
    length=n-nbase;
    for (j=1; j<=length; j++){
```

```
              diag2[i-1][j-1]=diag[i-1][nbase+j-1];
            }
          }
          else{
            nbase=nofst[i-1];
            length=n-nbase;
            for (j=nbase+1; j<=n; j++){
              diag2[i-1][j-1]=diag[i-1][j-nbase-1];
            }
          }
        }

        numnzc=1;
        numnz=1;
        for (j=1; j<=n; j++){
          ntopcfgc = 1;
          ntopcfg = 1;
          for (i=ndiag; i>=1; i--){
            if (diag2[i-1][j-1]!=0.0){
              ncol=j-nofst[i-1];
              c[numnzc-1]=diag2[i-1][j-1];
              nrowc[numnzc-1]=ncol;
              if (ncol>=j){
                a[numnz-1]=diag2[i-1][j-1];
                nrow[numnz-1]=ncol;
              }
              if (ntopcfgc==1){
                nfcnzc[j-1]=numnzc;
                ntopcfgc=0;
              }
              if (ntopcfg==1){
                nfcnz[j-1]=numnz;
                ntopcfg=0;
              }
              if (ncol>=j){
                numnz=numnz+1;
              }
              numnzc=numnzc+1;
            }
          }
        }


        nfcnzc[n]=numnzc;
        nnzc=numnzc-1;
        nfcnz[n]=numnz;
        nnz=numnz-1;

        ierr=c_dm_vmvscc(c, nnzc, nrowc, nfcnzc, n, solex, b, wc, (int*)iwc, &icon);

        for(i=1; i<=n; i++){
           x[i-1]=b[i-1];
        }
        iordering=0;
        isw=1;
        epsz=0;
        nsizefactor=1;
        nsizeindex=1;

        ierr=c_dm_vschol(a, nnz, nrow, nfcnz, n, iordering, nperm, isw, &epsz, nassign,
&nsupnum, nfcnzfactor, &dummyf, &nsizefactor, nfcnzindex, &ndummyi, &nsizeindex,
(int*)ndim, nposto, w, iw1, iw2, iw3, &icon);

        printf("\n");
        printf("     ICON = %d  NSIZEFACTOR = %lld NSIZEINDEX = %lld\n", icon,
nsizefactor, nsizeindex);
        printf("\n");

        panelfactor = (double *)malloc(sizeof(double)*nsizefactor);
        npanelindex = (int *)malloc(sizeof(int)*nsizeindex);
        isw=2;

        ierr=c_dm_vschol(a, nnz, nrow, nfcnz, n, iordering, nperm, isw, &epsz, nassign,
&nsupnum, nfcnzfactor, panelfactor, &nsizefactor, nfcnzindex, npanelindex, &nsizeindex,
(int*)ndim, nposto, w, iw1, iw2, iw3, &icon);
```

```
        ierr=c_dm_vscholx(n, iordering, nperm, x, nassign, nsupnum, nfcnzfactor,
panelfactor, nsizefactor, nfcnzindex, npanelindex, nsizeindex,
                          (int*)ndim, nposto, iw3, &icon);

        err = errnrm(solex,x,n);

        printf("     COMPUTED VALUES\n");
        printf("     X(1) = %.15lf  X(N) = %.15lf\n", x[0], x[n-1]);
        printf("\n");
        printf("     ICON = %d\n", icon);
        printf("\n");
        printf("     N = %d  :: NX = %d  NY = %d  NZ = %d\n",n,nx,ny,nz);
        printf("\n");
        printf("     ERROR = %.15e\n",err);
        printf("\n");
        printf("\n");
        if (err<(1.0e-8) && icon==0){
          printf("     ********** OK **********\n");
        }
        else{
          printf("     ********** NG **********\n");
        }
          free(panelfactor);
          free(npanelindex);
          return 0;
      }

      void init_mat_diag(double va1, double va2, double va3, double vc,
               double d_l[], int offset[], int nx, int ny, int nz,
               double xl, double yl, double zl, int ndiag, int len, int ndivp)
      {
        int i, l, j;
        int length, numnz, js;
        int i0, j0, k0;
        int ndiag_loc;
        int nxy;

        double hx, hy, hz;
        double x1, x2;
        double base;
        double ret, remark;

        if (ndiag<1){
          printf("FUNCTION INIT_MAT_DIAG:\n");
          printf("NDIAG SHOULD BE GREATER THAN OR EQUAL TO 1\n");
          return;
        }
        ndiag_loc = ndiag;
        if (ndiag>7){
          ndiag_loc=7;
        }

        hx = xl / (nx + 1);
        hy = yl / (ny + 1);
        hz = zl / (nz + 1);

        for (i=1; i<=ndivp; i++){
          for (j=1; j<=ndiag; j++){
            d_l[i-1+(j-1)*ndivp]= 0.;
          }
        }

        nxy = nx * ny;
        l = 1;
        if (ndiag_loc >= 7) {
          offset[l-1] = -nxy;
          ++l;
        }
        if (ndiag_loc >= 5) {
          offset[l-1] = -nx;
          ++l;
        }
        if (ndiag_loc >= 3) {
          offset[l-1] = -1;
          ++l;
        }
        offset[l-1] = 0;
```

```
  ++l;
  if (ndiag_loc >= 2) {
    offset[l-1] = 1;
    ++l;
  }
  if (ndiag_loc >= 4) {
    offset[l-1] = nx;
    ++l;
  }
  if (ndiag_loc >= 6) {
    offset[l-1] = nxy;
  }

  for (j = 1; j <= len; ++j) {
    js=j;
    k0 = (js - 1) / nxy + 1;
    if (k0 > nz) {
      printf("ERROR; K0.GH.NZ\n");
      return;
    }
    j0 = (js - 1 - nxy * (k0 - 1)) / nx + 1;
    i0 = js - nxy * (k0 - 1) - nx * (j0 - 1);

    l = 1;
    if (ndiag_loc >= 7) {
      if (k0 > 1) {
        d_l[j-1+(l-1)*ndivp] = -(1.0/hz+va3*0.5)/hz;
      }
      ++l;
    }

    if (ndiag_loc >= 5) {
      if (j0 > 1) {
        d_l[j-1+(l-1)*ndivp] = -(1.0/hy+va2*0.5)/hy;
      }
      ++l;
    }

    if (ndiag_loc >= 3) {
      if (i0 > 1) {
        d_l[j-1+(l-1)*ndivp] = -(1.0/hx+va1*0.5)/hx;
      }
      ++l;
    }

    d_l[j-1+(l-1)*ndivp] = 2.0/(hx*hx)+vc;
    if (ndiag_loc >= 5) {
      d_l[j-1+(l-1)*ndivp] += 2.0/(hy*hy);
      if (ndiag_loc >= 7) {
        d_l[j-1+(l-1)*ndivp] += 2.0/(hz*hz);
      }
    }
    ++l;
    if (ndiag_loc >= 2) {
      if (i0 < nx) {
        d_l[j-1+(l-1)*ndivp] = -(1.0/hx-va1*0.5)/hx;
      }
      ++l;
    }

    if (ndiag_loc >= 4) {
      if (j0 < ny) {
        d_l[j-1+(l-1)*ndivp] = -(1.0/hy-va2*0.5)/hy;
      }
      ++l;
    }

    if (ndiag_loc >= 6) {
      if (k0 < nz) {
        d_l[j-1+(l-1)*ndivp] = -(1.0/hz-va3*0.5)/hz;
      }
    }
  }
  return;
}

double errnrm(double *x1, double *x2, int len)
```

```
{
  double ret_val;

  int i;
  double s, ss;

  s = 0.;
  for (i = 1; i <= len; ++i) {
    ss = x1[i-1] - x2[i-1];
    s += ss * ss;
  }
  ret_val = sqrt(s);
  return ret_val;
}
```

c_dm_vsclu

複素スパース行列の LU 分解
ierr = c_dm_vsclu(za, nz, nrow, nfcnz, n, ipledsm, mz, isclitermax, &iordering, nperm, isw, nrowsym, nfcnzsym, nassign, &nsupnum, nfcnzfactorl, zpanelfactorl, &nsizefactorl, nfcnzindexl, npanelindexl, &nsizeindexl, ndim, nfcnzfactoru, zpanelfactoru, &nsizefactoru, nfcnzindexu, npanelindexu, &nsizeindexu, nposto, sclrow, sclcol, &epsz, &thepsz, ipivot, istatic, &spepsz, nfcnzpivot, npivotp, npivotq, zw, w, iw1, iw2, &icon);

1. 機能

$n \times n$ の複素スパース行列 **A** に対角要素に大きな要素を並べる並び換えを行い. 行および列の均衡化を行うスケーリングを行います. スーパーノードの対角ブロック内で指定した Pivot をとり, LU 分解します.

要素の並び換えや スケーリングおよび Pivot では, 複素数の大きさを表す絶対値は, 実部の絶対値と虚部の絶対値を加えたもので近似します.
複素スパース行列 **A** は以下のように変換できます.

$$\mathbf{A_1} = \mathbf{D_r A P_c D_c}$$

ここで $\mathbf{P_c}$ は列の入れ換えを行う直交行列, $\mathbf{D_r}$ は行のスケーリングを行う対角行列, $\mathbf{D_c}$ は列のスケーリングを行う対角行列です.

$$\mathbf{A_2} = \mathbf{Q P A_1 P^T Q^T}$$

$\mathbf{A_2}$ は, スーパーノードの対角ブロックで指定された Pivot を行い, 行および列の入れ換えを行い LU 分解されます.
ただし, **P** は, $\mathbf{SYM} = \mathbf{A_1} + \mathbf{A_1^T}$ の非ゼロパターンに対して求めた ordering による行列要素の並び換えを示す置換行列で, **Q** は **SYM** に対する post order による行列要素の並び換えを示す置換行列を示します. **P**, **Q** は直交行列です.

L は下三角行列, **U** は単位上三角行列です.

Pivot 処理で, 閾値 thepsz より絶対値が大きな Pivot を見つけることが出来なかったとき, 対角ブロック内の絶対値が最大の要素を Pivot の候補とします. この値が小さ過ぎるときに Static Pivot を与えて近似的に LU 分解を続けることができます.

2. 引数

呼出し形式:

```
ierr = c_dm_vsclu(za, nz, nrow, nfcnz, n, ipledsm, mz, isclitermax,
            &iordering, nperm, isw, nrowsym, nfcnzsym, nassign, &nsupnum,
            nfcnzfactorl, zpanelfactorl, &nsizefactorl, nfcnzindexl,
            npanelindexl, &nsizeindexl, (int *)ndim, nfcnzfactoru,
            zpanelfactoru, &nsizefactoru, nfcnzindexu, npanelindexu,
            &nsizeindexu, nposto, sclrow, sclcol, &epsz, &thepsz, ipivot,
            istatic, &spepsz, nfcnzpivot, npivotp, npivotq, zw, w, iw1, iw2,
            &icon);
```

引数の説明:

za	dcomplex za[nz]	入力	複素スパースな係数行列 **A** を圧縮列格納法で格納します. 圧縮列格納法については, 実スパース行列と実ベクトル(圧縮列格納法), c_dm_vmvscc の図 c_dm_vmvscc-1 を参照してください. 実数型の配列 a を複素数型の配列に変えたものを利用します. (“使用上の注意” e)参照)
nz	int	入力	複素スパースな係数行列 **A** の非零要素の総数.
nrow	int nrow[nz]	入力	圧縮列格納法で使用される行指標で za に格納される要素が何番目の行ベクトルに属するかを示します.
nfcnz	int nfcnz[n+1]	入力	行列 **A** の各列の非零要素を列方向に圧縮して順次配列 za に格納するとき, 対応する列の最初の非零要素が格納される位置を表します. nfcnz[n] = nz + 1.
n	int	入力	行列 **A** の次数 n.
ipledsm	int	入力	対角要素に大きな要素を並べる列の入れ換えを行うかを指定します. 1 のとき, 列の入れ換えを求めて入れ換えます. 1 以外のとき, 列の入れ換えを行いません.
mz	int mz[n]	出力	ipledsm に 1 が指定されたとき, 列の入れ換えを表す. mz[i-1] = j は行列 $\mathbf{a}_{ij}$ の属する *j* 番目の列を *i* 番目の列に移動する. $\mathbf{a}_{ij}$ は対角要素に並べる大きな要素を表す.
isclitermax	int	入力	係数行列のスケーリングを行う対角行列 $\mathbf{D}_r$ と $\mathbf{D}_c$ を反復して求める反復回数. isclitermax ≤ 0 のときスケーリングは行わずに, $\mathbf{D}_r$ および $\mathbf{D}_c$ は単位行列が設定されます. isclitermax ≥ 10 のときは 10 回を上限値とします.
iordering	int	入力	ordering を表す置換行列 **P** で $\mathbf{PA}_1\mathbf{P}^T$ と変換した行列を LU 分解するかを指定します. 置換行列は直交行列です. 10 のとき, isw = 1 で呼び出すと ordering を求める $\mathbf{A}_1$ に関する情報を求めて nrowsym, nfcnzsym に設定します. 11 のとき, isw = 1 で 10 を指定して呼び出して得た行列を対称化した情報 nrowsym, nfcnzsym を使って決めた ordering を nperm に指定して同じく isw = 1 で呼び出し, 計算を続けることを示します. $\mathbf{PA}_1\mathbf{P}^T$ を LU 分解する処理を続けます.

			10,11 以外のとき, ordering は指定せずに行列 $\mathbf{A_1}$ をそのまま LU 分解します.
		出力	isw = 1 と iordering = 10 を指定して呼び出した後, iordering に 11 が設定されます. そのため, ordering を nperm に指定して再び呼び出すときに特に 11 を指定する必要はありません. ("使用上の注意" a)参照)
nperm	int nperm[n]	入力	iordering = 11 のとき使用する置換行列をベクトルで指定します. ("使用上の注意" a)参照)
isw	int	入力	呼び出しに関する制御情報を示します. 1) 1 のとき 対称化および symbolic decomposition を行い, 必要な領域が割り当てられているか調べ計算を行います. iordering = 10 で呼び出し, ordering を求めるための情報が nrowsym, nfcnzsym に出力されます. これらを使って **SYM** に対する ordering を求めた後, nperm に指定して iordering = 11 を指定してもう 1 回 isw = 1 で呼び出します. iordering が 10, 11 以外のときは ordering は行ないません. 2) 2 のとき isw = 1 で呼び出したとき, zpanelfactorl, zpanelfactoru, npanelindexl または npanelindexu の大きさが不足して icon = 31000 で終了した処理を継続します. このとき, nsizefactorl, nsizefactoru, nsizeindexl または nsizeindexu に返却された必要な大きさで zpanelfactorl, zpanelfactoru, npanelindexl または npanelindexu を確保し直して指定しなおして再度呼び出します. これらの引数と実行を制御する引数 isw 以外の引数に格納されている値は変更してはいけません.
nrowsym	int nrowsym[nz+n]	出力	iordering = 10 で呼び出したとき, 対称化した **SYM** = $\mathbf{A_1} + \mathbf{A_1}^T$ の非ゼロパターンの下三角行列部分の行指標を列圧縮したものが返却されます. ipledsm = 1 のときは, $\mathbf{A_1} = \mathbf{D_r A P_c D_c}$ です.
nfcnzsym	int nfcnzsym[n+1]	出力	iordering = 10 で呼び出したとき, 行列 SYM の下三角部分の各列の非零要素の行指標を列方向に圧縮して順次配列 nrowsym に格納するとき,対応する列の最初の非零要素が格納される位置を表します. nfcnzsym[n] = nsymz + 1 nsymz は, 総要素数を表します.
nassign	int nassign[n]	出力	各 supernode に対する L, U は圧縮して 2 次元の panel に格納します. この panel を zpanelfactorl および zpanelfactoru の 1 次元部分配列として順番に割り付けたとき何番目になるかを示します. これらの指標ベクトルも同じように npanelindexl, npanelindexu に割り付けます.

			j = nassign[i-1]のとき, *i* 番目の supernode を *j* 番目に割り付けたことを示します. 分解結果の格納方法については図 c_dm_vsclu-1 を参照してください.
		入力	isw ≠ 1 のとき, 初回呼び出しの値を再利用します.
nsupnum	int	出力	supernode の総数.
		入力	isw ≠ 1 のとき, 初回呼び出しの値を再利用します. (≤n)
nfcnzfactorl	long nfcnzfactorl[n+1]	出力	複素スパース行列の LU 分解の結果の行列 **L** および **U** はスーパーノードに対応しておのおの求めます. 各 supernode に対応する **L** の列ベクトルは, 共通の行指標ベクトルを持つように圧縮してブロック対角部分に **U** の対応部分を含めて 2 次元の panel に格納します. この panel を zpanelfactorl の 1 次元部分配列として順番に割り付けたとき, *i* 番目の panel の先頭要素 panel[0][0]が 1 次元配列の zpanelfactorl の何番目の要素になるかを示します. 結果の格納方法については図 c_dm_vsclu-1 を参照してください.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します.
zpanelfactorl	dcomplex zpanelfactorl [nsizefactorl]	出力	各 supernode の分解結果に対応する行列 **L** の複数の列ベクトルと行列 **U** のブロック対角部分について, 共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します. panel を順番に割り付けます. *i* 番目の supernode に対応する panel が何番目の位置に割り当てられるかは j = nassign[i-1]から分かります. その先頭位置は nfcnzfactorl[j-1]に格納されています. panel ごとに分解結果が格納されます. *i* 番目に割付けられた panel の大きさは ndim[i-1][0] × ndim[i-1][1]の 2 次元配列と見なせます. i 番目の panel の panel[t-1][s-1], s ≥ t, s = 1,…, ndim[i-1][0], t = 1,…, ndim[i-1][1]に下三角行列 **L** が格納されます. panel の panel[t-1][s-1], s < t, t = 1,…, ndim[i-1][1]には単位上三角行列 **U** の対角要素を除いたブロック対角部分が格納されます. 結果の格納方法については図 c_dm_vsclu-1 を参照してください. (“使用上の注意” c)参照)
nsizefactorl	long	入力	zpanelfactorl の大きさを示す.
		出力	zpanelfactorl の大きさとして必要な大きさが返却されます. (“使用上の注意” c)参照)
nfcnzindexl	long nfcnzindexl[n+1]	出力	各 supernode の分解結果に対応する行列 **L** の複数の列ベクトルと行列 **U** のブロック対角部分について, 共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します. この行指標ベクトルを npanelindexl に順番に割り付けたとき i 番目の行指標ベクトルの先頭要素が 1 次元配列である npanelindexl の何番目の要素になるかを示します. 結果の格納方法については図 c_dm_vsclu-1 を参照して

			ください.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します.
npanelindexl	int npanelindexl [nsizeindexl]	出力	各 supernode の分解結果に対応する行列 **L** の複数の列ベクトルと行列 **U** のブロック対角部分について, 共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します. この行指標ベクトルを npanelindexl に順番に割り付けます. i 番目の supernode に対応する panel の行の指標ベクトルが何番目に割り当てられるかは j = nassign[i-1]から分かります. その先頭位置は nfcnzindexl[j-1]に格納されています. この行指標は行列 **SYM** に対する post order で並び換えたときの行の番号です. 結果の格納方法については図 c_dm_vsclu-1 を参照してください. ("使用上の注意" c)参照)
nsizeindexl	long	入力	npanelindexl の大きさを示す.
		出力	npanelindexl の大きさとして必要な大きさが返却されます. ("使用上の注意" c)参照)
ndim	int ndim[n][3]	出力	ndim[i-1][0]および ndim[i-1][1]は行列 **L** に関して *i* 番目に割り付けられた panel の 1 次元目と 2 次元目の大きさを示します. ndim[i-1][2]は行列 **U** に関して割り付けられた panel を転置したときの 1 次元目の大きさに対角ブロックの大きさを加えた大きさを示します. 結果の格納方法については図 c_dm_vsclu-1 を参照してください.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します.
nfcnzfactoru	long nfcnzfactoru[n+1]	出力	複素スパース行列の LU 分解の結果の行列 **U** に関しては, 各 supernode に対応する **U** の行ベクトルは, ブロック対角部分を除いて共通の列指標ベクトルを持つように圧縮し, 転置したものを 2 次元の panel に格納します. この panel を zpanelfactoru の 1 次元部分配列として順番に割り付けたとき, *i* 番目の panel の先頭要素 panel[0][0]が 1 次元配列の zpanelfactoru の何番目の要素になるかを示します. 結果の格納方法については図 c_dm_vsclu-1 を参照してください.
		入力	isw ≠ 1 のとき, 初回呼び出しの値を再利用します.
zpanelfactoru	dcomplex zpanelfactoru [nsizefactoru]	出力	各 supernode の分解結果に対応する行列 **U** のブロック対角部分を除いた複数の行ベクトルについて, 共通の列指標ベクトルを持つように圧縮し, 転置して 2 次元の panel に格納します. panel を順番に割り付けます. *i* 番目の supernode に対応する panel が何番目の位置に割り当てられるかは j = nassign[i-1]から分かります. その先頭位置は nfcnzfactoru[j-1]に格納されています. panel ごとに分解結果が格納されます. *i* 番目に割付けられた panel の大きさは{ndim[i-1][2] – ndim[i-1][1] } × ndim[i-1][1]の 2 次元配列と見なせます. *i* 番目の panel の panel[t-

			1][s-1], s = 1,…, ndim[i-1][2] – ndim[i-1][1], t = 1,…, ndim[i-1][1]に単位上三角行列 **U** のブロック対角部分を除いた部分が行ベクトルを圧縮し, 転置して格納します. 結果の格納方法については図 c_dm_vsclu-1 を参照してください. (“使用上の注意” c)参照)
nsizefactoru	long	入力	zpanelfactoru の大きさを示す.
		出力	zpanelfactoru の大きさとして必要な大きさが返却されます. (“使用上の注意” c)参照)
nfcnzindexu	long nfcnzindexu[n+1]	出力	各 supernode の分解結果に対応する行列 **U** は対角ブロック部分を除いて複数の行ベクトルについて, 共通の列指標ベクトルを持つように圧縮し, 転置して 2 次元の panel に格納します. 対角ブロック部分も含んだこの列指標ベクトルを npanelindexu に順番に割り付けたとき *i* 番目の列指標ベクトルの先頭要素が 1 次元配列である npanelindexu の何番目の要素になるかを示します. 結果の格納方法については図 c_dm_vsclu-1 を参照してください.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します.
npanelindexu	int npanelindexu [nsizeindexu]	出力	各 supernode の分解結果に対応する行列 **U** の対角ブロックを除いた部分を転置して圧縮して 2 次元の panel に格納します. この列指標ベクトルを対角ブロック部分を含めて npanelindexu に順番に割り付けます. *i* 番目の supernode に対応する panel の行の指標ベクトルが何番目に割り当てられるかは j = nassign[i-1]から分かります. その先頭位置は nfcnzindexu[j-1]に格納されています. この行指標は行列 **SYM** に対する post order で並び換えたときの列の番号です. 結果の格納方法については図 c_dm_vsclu-1 を参照してください. (“使用上の注意” c)参照)
nsizeindexu	long	入力	npanelindexu の大きさを示す.
		出力	npanelindexu の大きさとして必要な大きさが返却されます. (“使用上の注意” c)参照)
nposto	int nposto[n]	出力	post order で *i* 番目の node が行列 **A** の何番目の列番号に対応するかを表す 1 次元ベクトル.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します. (“使用上の注意” d)参照)
sclrow	double sclrow[n]	出力	スケーリング対角行列 $\mathbf{D}_r$. 対角要素が 1 次元配列に格納されます.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します.
sclcol	double sclcol[n]	出力	スケーリング対角行列 $\mathbf{D}_c$. 対角要素が 1 次元配列に格納されます.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します.
epsz	double	入力	分解過程で対角要素の大きさの絶対値の相対判定値
		出力	epsz ≤ 0.0 のときは標準値が設定されます. (“使用上の注意” b)参照)
thepsz	double	入力	ピボットの判定での閾値(threshold). これより大きな値

			はピボットとして採用します. 見つかればその時点で, ピボット検索は打ち切ります. 例えば 10^{-2} 程度.
		出力	thepsz ≤ 0.0 のときは 10^{-2} が設定されます. epsz ≥ thepsz > 0.0 のときは, epsz が設定されます.
ipivot	int	入力	スーパーノード内でピボットを行うか, 行う場合どのようなピボットを行うかを指定します. 例えば, 完全ピボットとして 40 を指定. ipivot < 10 : または ipivot ≥ 50: ピボットなし. 10 ≤ ipivot < 20 : 部分ピボット 20 ≤ ipivot < 30 : 対角ピボット 21 : スーパーノード内で対角ピボットがとれないとき, Rook ピボットに変更します. 22 : スーパーノード内で対角ピボットがとれないときは, Rook ピボットに, Rook ピボットが取れないときは完全ピボットに変更します. 30 ≤ ipivot < 40 :Rook ピボット 32 : スーパーノード内で Rook ピボットがとれないとき, 完全ピボットをとります. 40 ≤ ipivot < 50 : 完全ピボット
istatic	int	入力	Pivot を指定したとき, Static Pivot を行うかを示します. 1) = 1 のとき スーパーノード内で指定したピボットが spepsz より大きくないとき, ピボットとして大きさ spepsz の複素数で近似します. ピボットの値が 0.0 なら spepsz に近似します. このとき, 以下を設定しなければなりません. a) epsz は epsz の標準値以下にしなければなりません. b) isclitermax は 10 を設定しスケーリングを行う必要があります. c)thepsz ≥ spepsz でなければなりません. 2) ≠ 1 のとき Static Pivot は行いません.
spepsz	double	入力	istatic = 1 のとき, Statc Pivot として使う値. thepsz ≥ spepsz ≥ epsz でなければなりません.
		出力	spepsz < epsz のときは, 10^{-10} が設定されます.
nfcnzpivot	int nfcnzpivot [nsupnum+1]	出力	スーパーノード内の相対的な位置でのピボットの行, 列の入れ換えの履歴を格納する位置を示す. *i* 番目の supernode に関する情報が何番目の位置に割り当てられるかは j = nassign[i-1]から分かります.その先頭位置は nfcnzpivot[j-1]に格納されています. *i* 番目のスーパーノードの入れ換え情報は, npivotp, npivotq の is = nfcnzpivot[j-1],…, ie = nfcnzpivot[j-1] + ndim[j-1][1] - 1 番目の要素に格納されます.
npivotp	int npivotp[n]	出力	スーパーノード内の行の入れ換えに関する情報を格納する.
npivotq	int npivotq[n]	出力	スーパーノード内の列の入れ換えに関する情報を格納

			する.
zw	dcomplex zw[2*nz]	作業域	isw = 1, 2 で続けて呼び出すとき, 呼び出しの間で必要なデータを受け渡すために使われます. 呼び出しの間で値を変更してはいけません.
w	double w[4*nz+6*n]	作業域	isw = 1, 2 で続けて呼び出すとき, 呼び出しの間で必要なデータを受け渡すために使われます. 呼び出しの間で値を変更してはいけません.
iw1	int iw1[2*nz+2*(n+1)+16*n]	作業域	isw = 1, 2 で続けて呼び出すとき, 呼び出しの間で必要なデータを受け渡すために使われます. 呼び出しの間で値を変更してはいけません.
iw2	int iw2[47*n+47+nz+4*(n+1)+2*(nz+n)]	作業域	isw = 1, 2 で続けて呼び出すとき, 呼び出しの間で必要なデータを受け渡すために使われます. 呼び出しの間で値を変更してはいけません.
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	istatic = 1 のとき小さすぎるピボットを spepsz で置き換える Static Pivot を行った.	処理は続行する.
20000	ピボットが相対的に零になった.	処理を打ち切る.
20100	ipledsm = 1 を指定して, 対角要素に絶対値が大きな要素を並べるため maximum matching を求める処理で, 長さ n の maximum matching がみつからなかった. 行列が特異である可能性がある.	
20200	行および列の均衡化を行う対角行列の対角要素を求める過程で, 元の行列 A の行もしくは列にゼロベクトルがあった. 行列が特異である可能性がある.	
30000	次のいずれかであった: • n < 1 • nz < 0 • nfcnz[n] ≠ nz + 1 • nsizefactorl < 1 • nsizefactoru < 1 • nsizeindexl < 1 • nsizeindexu < 1 • isw < 1 • isw > 2	
30100	nperm に指定した置換行列が正しくない.	
30200	nrow[j-1]に格納された *i* 列目の行指標 k が k<1 または k>n	
30300	*i* 列目の行指標の数 nfcnz[i] – nfcnz[i-1] > n	

コード	意　味	処 理 内 容
30500	istatic = 1 のとき満たすべき条件をみたしていない. epsz は標準値 16***u*** より大きかった. または isclitermax < 10 であった. または spepsz > thepsz であった.	
31000	zpanelfactorl の大きさ nsizefactorl または npanelindexl の大きさ nsizeindexl または zpanelfactoru の大きさ nsizefactoru または npanelindexu の大きさ nsizeindexu が小さすぎます.	nsizefactorl または nsizeindexl または nsizefactoru または nsizeindexu で指定された 大きさで zpanelfactorl または npanelindexl または zpanelfactoru または npanelindexu を割り付けて isw = 2 として再度呼び出す.

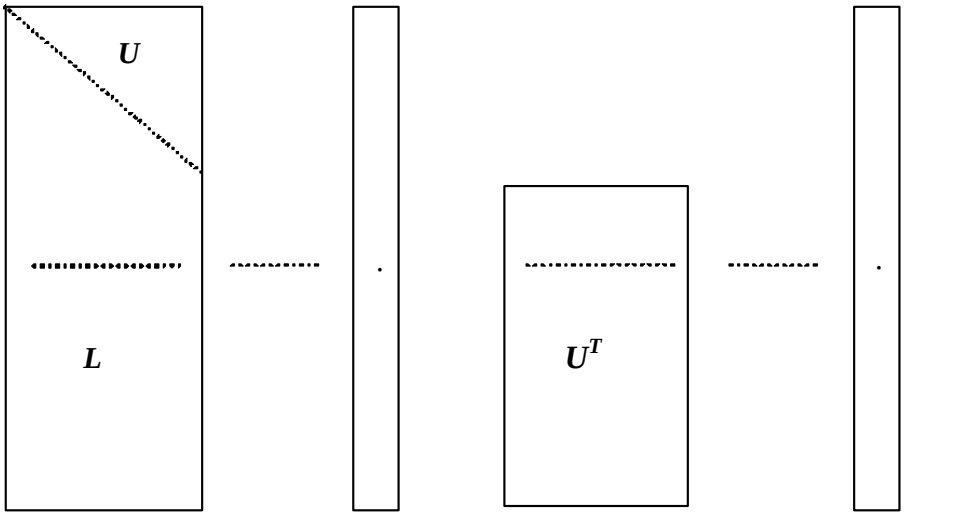


図 c_dm_vsclu-1 分解結果の格納概念図

j = nassign[i-1]　→ *i* 番目の supernode は *j* 番目に格納されます.

p = nfcnzfactorl[j-1] → *j* 番目の panel は zpanelfactorl の *p* 番目の要素から ndim[j-1][0] × ndim[j-1][1] の長さを占めます.

q = nfcnzindexl[j-1] → *j* 番目の panel の行指標を表すベクトルは npanelindexl の *q* 番目の要素から ndim[j-1][0] の長さを占めます.

panel は大きさ ndim[j-1][0] × ndim[j-1][1]の配列と見なせます.

panel[t-1][s-1] , s ≥ t, s = 1,…, ndim[j-1][0], t = 1,…, ndim[j-1][1] に分解結果の下三角行列 **L** が格納されます.

panel[t-1][s-1], s < t, s = 1,…, ndim[j-1][1], t = 1,…, ndim[j-1][1] に単位上三角行列 **U** の対角ブロック部分が対角要素を除き格納されます.

u = nfcnzfactoru[j-1] → *j* 番目の panel は zpanelfactoru の *u* 番目の要素から(ndim[j-1][2] – ndim[j-1][1]) × ndim[j-1][1] の長さを占めます.

v = nfcnzindexu[j-1] → *j* 番目の panel の列指標を表すベクトルは npanelindexu の *v* 番目の要素から ndim[j-1][2] の長さを占めます.

panel は大きさ(ndim[j-1][2] – ndim[j-1][1]) × ndim[j-1][1] の配列と見なせます.

panel[y-1][x-1], x = 1,…, ndim[j-1][2] - ndim[j-1][1], y = 1,…, ndim[j-1][1]

に分解結果の単位上三角行列 **U** の転置行列 $\mathbf{U}^{\mathrm{T}}$ の対角ブロック部分を除いた部分が格納されます.

指標の値は行列 **A** の列番号をもつ node を post order で並び換えた $\mathbf{QAQ}^{\mathrm{T}}$ での列番号を表します.

3. 使用上の注意

a)

直交行列である置換行列 **P** の要素 $\mathbf{p}_{ij} = \mathbf{1}$ のとき, nperm[i-1] = j と表現します.
逆は以下のようにして求めることができます.

```
for (i = 1; i <= n; i++) {
 j = nperm[i-1];
 nperminv[j-1] = i;
}
```

Fill-Reducing Ordering は METIS などを使って求めることができます.
詳細は“付録 参考文献一覧表”の[43], [44]を参照願います.

b)

分解過程で対角要素の大きさの絶対値の相対零判定値にある値を設定したとすると, この値は次の意味を持っています. すなわち, LU 分解の過程で, 対角要素の大きさの絶対値がその値より小さくなった場合に, その値を相対的に零と見なし, icon = 20000 として処理を打ち切ります. epsz の標準値は, 丸め誤差の単位を *u* としたとき, epsz = 16*u* です.
Pivot では, 複素数の大きさを表す絶対値は, 実部の絶対値と虚部の絶対値を加えたもので近似します.
なお, 対角要素の大きさの絶対値が小さくなっても, 処理を続行させたい場合には, epsz に極小の値を与えれば良いのですが, 結果の精度は保証されません.
Static Pivot を指定した場合, 対角要素が spepsz より小さいとき, 大きさ spepsz の複素数で近似して LU 分解を行います.

c)

分解結果を格納する配列 zpanelfactorl, npanelindexl, zpanelfactoru, npanelindexu の必要な大きさは, 事前には分かりません. 十分大きな配列を割り当てるか, 本関数を呼び出して symbolic decomposition を行った解析の結果を使って割り当てを行うことができます.
例えば, 大きさ 1 の 1 次元配列などを割付けます. そしてその大きさ 1 などの小さな値を nsizefactorl, nsizeindexl, nsizefactoru, nsizeindexu に指定して, isw = 1 で呼び出します.
symbolic decomposition を行い, icon = 31000 で終了し, nsizefactorl, nsizeindexl, nsizefactoru, nsizeindexu に必要な大きさが返却されます. 必要な大きさの配列を割付け直して引数に指定して, isw = 2 で呼び出すことで, symbolic decomposition 以降の処理を続けることができます. 使用例を参照願います.

LU 分解の結果を利用して c_dm_vsclux を利用して連立 1 次方程式を解く上での注意は“使用上の注意” e)参照.

d)

列番号に対応するノードを考えます. これを post order で順番を並び換えたときの番号の対応関係が nposto に格納されています. post order の *i* 番目のノードが並び換える前の何番目のノードに対応するかを表します. *j* = nposto[i-1]は *j* 番目であることを表します.
a)同様にこれは直交行列である置換行列 **Q** を表し, 行列 **A** を $\mathbf{QAQ}^T$ と並び換えることに相当します.
逆変換 $\mathbf{Q}^T$ は以下のようにして求めることができます.

```
for (i = 1; i <= n; i++) {
 j = nposto[i-1];
 npostoinv[j-1] = i;
}
```

e)

本関数で得られた LU 分解の結果を使い, c_dm_vsclux を引き続いて呼びだすことで連立 1 次方程式 **Ax** = **b** を解くことができます.
このとき, c_dm_vsclu で利用した以下の引数を指定します. 使用例を参照願います.
za, nz, nrow, nfcnz, n,
ipledsm, mz, iordering, nperm,
nassign, nsupnum,
nfcnzfactorl, zpanelfactorl,
nsizefactorl, nfcnzindexl, npanelindexl,
nsizeindexl, ndim,
nfcnzfactoru, zpanelfactoru, nsizefactoru,
nfcnzindexu, npanelindexu, nsizeindexu, nposto,
sclrow, sclcol,
nfcnzpivot,
npivotp, npivotq, iw2

4. 使用例

連立 1 次方程式 **Ax** = **f** を解きます.
行列は境界条件として立方体の境界で 0 となる楕円型の偏微分方程式に有限差分法を適用して生成されたものを利用します.

$$-\Delta u + a\nabla u + cu = f$$

ここで $a = (a_1, a_2, a_3)$.
行列は関数 init_mat_diag によって生成されます.
対角形式格納法で生成し, 対角ベクトルの下 6 本を圧縮列格納法で格納します. 結果生成された非対称な行列 **A** に関する連立 1 次方程式を解きます.
(並列実行を行うスレッド数は環境変数(OMP_NUM_THREADS)などで指定できます.
例えば, 4 プロセッサーのシステムで 4 つのスレッドで並列実行するときは, OMP_NUM_THREADS を 4 に設定して実行します.)

```
/* **EXAMPLE** */
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <malloc.h>
#include <omp.h>
#include "cssl.h"
```

```
#define  NORD  40
#define  KX  NORD
#define  KY  NORD
#define  KZ  NORD
#define  N  KX * KY * KZ
#define  NBORDER  (N + 1)
#define  NOFFDIAG  6
#define  K  (N + 1)
#define  NDIAG  7
#define  NALL  NDIAG * N

#define  ZWL  2 * NALL
#define  WL  4 * NALL + 6 * N
#define  IW1L  2 * NALL + 2 * (N + 1) + 16 * N
#define  IW2L  47 * N + 47 + 4 * (N + 1) + NALL + 2 * (NALL + N)

void init_mat_diag(double, double, double, double, double*, int*, int, int, int,
                   double, double, double, int, int, int);
double errnrm(dcomplex*, dcomplex*, int);
dcomplex comp_sub(dcomplex, dcomplex);

int MAIN__() {

  int  nofst[NDIAG];
  double  diag[NDIAG][K], diag2[NDIAG][K];
  dcomplex  za[K * NDIAG], zwc[K * NDIAG],
            zw[ZWL], zone;
  int  nrow[K * NDIAG], nfcnz[N + 1],
       nrowsym[K * NDIAG + N], nfcnzsym[N + 1],
       iwc[K * NDIAG][2];
  int  nperm[N],
       nposto[N], ndim[N][3],
       nassign[N],
       mz[N],
       iw1[IW1L], iw2[IW2L];
  double  w[WL];
  dcomplex  *zpanelfactorl, *zpanelfactoru;
  int  *npanelindexl, *npanelindexu;
  dcomplex  zdummyfl, zdummyfu;
  int  ndummyil,
       ndummyiu;
  long  nsizefactorl,
        nsizeindexl,
        nsizeindexu,
        nsizefactoru,
        nfcnzfactorl[N + 1],
        nfcnzfactoru[N + 1],
        nfcnzindexl[N + 1],
```

```
      nfcnzindexu[N + 1];
dcomplex  zb[N], zsolex[N];
double  epsz, thepsz, spepsz,
        sclrow[N], sclcol[N];

int  ipivot, istatic, nfcnzpivot[N + 1],
     npivotp[N], npivotq[N],
     irefine, itermax, iter, ipledsm;
double  err, va1, va2, va3, vc, xl, yl, zl, epsr;
int  i, j, nbase, length, numnz, ntopcfg, ncol, nz, icon, iordering,
     isclitermax, isw, nsupnum;

zone.re = 1.0;
zone.im = 0.0;

printf("    LU DECOMPOSITION METHOD\n");
printf("    FOR SPARSE UNSYMMETRIC COMPLEX MATRICES\n");
printf("    IN COMPRESSED COLUMN STORAGE\n\n");

for (i = 0; i < N; i++) {
  zsolex[i] = zone;
}
printf("    EXPECTED SOLUTIONS\n");
printf("    X(1) = (%lf,%lf) X(N) = (%lf,%lf)\n\n",
       zsolex[0].re, zsolex[0].im, zsolex[N - 1].re, zsolex[N - 1].im);

va1 = 1.0;
va2 = 2.0;
va3 = 3.0;
vc = 4.0;
xl = 1.0;
yl = 1.0;
zl = 1.0;
init_mat_diag(va1, va2, va3, vc, (double *)diag, nofst,
              KX, KY, KZ, xl, yl, zl, NDIAG, N, K);

for (i = 0; i < NDIAG; i++) {
  for (j = 0; j < K; j++) {
    diag2[i][j] = 0;
  }
}

for (i = 0; i < NDIAG; i++) {

  if (nofst[i] < 0) {
    nbase = -nofst[i];
    length = N - nbase;
    for (j = 0; j < length; j++) {
      diag2[i][j] = diag[i][nbase + j];
```

```
      }
    } else {
      nbase = nofst[i];
      length = N - nbase;
      for (j = 0; j < length; j++) {
        diag2[i][nbase + j] = diag[i][j];
      }
    }

  }

  numnz = 1;

  for (j = 0; j < N; j++) {
    ntopcfg = 1;

    for (i = NDIAG - 1; i >= 0; i--) {

      if (ntopcfg == 1) {
        nfcnz[j] = numnz;
        ntopcfg = 0;
      }

      if (j + 1 < NBORDER && i + 1 > NOFFDIAG) {
        continue;
      } else {

        if (diag2[i][j] != 0.0) {

          ncol = (j + 1) - nofst[i];
          za[numnz - 1].re = diag2[i][j];
          za[numnz - 1].im = 0.0;
          nrow[numnz - 1] = ncol;

          numnz++;

        }
      }
    }
  }

  nfcnz[N] = numnz;
  nz = numnz - 1;

  c_dm_vmvsccc(za, nz, nrow, nfcnz, N, zsolex,
               zb, zwc, (int *)iwc, &icon);

/* INITIAL CALL WITH IORDER=1 */
```

```
iordering = 0;
ipledsm = 1;
isclitermax = 10;
isw = 1;
nsizefactorl = 1;
nsizefactoru = 1;
nsizeindexl = 1;
nsizeindexu = 1;
epsz = 1.0e-16;
thepsz = 1.0e-2;
spepsz = 0.0;
ipivot = 40;
istatic = 0;
irefine = 1;
epsr = 0.0;
itermax = 10;

c_dm_vsclu(za, nz, nrow, nfcnz, N,
          ipledsm, mz, isclitermax, &iordering,
          nperm, isw,
          nrowsym, nfcnzsym,
          nassign,
          &nsupnum,
          nfcnzfactorl, &zdummyfl,
          &nsizefactorl,
          nfcnzindexl,
          &ndummyil, &nsizeindexl,
          (int *)ndim,
          nfcnzfactoru, &zdummyfu,
          &nsizefactoru,
          nfcnzindexu,
          &ndummyiu, &nsizeindexu,
          nposto,
          sclrow, sclcol,
          &epsz, &thepsz,
          ipivot, istatic, &spepsz, nfcnzpivot,
          npivotp, npivotq,
          zw, w, iw1, iw2, &icon);

printf("ICON=%d NSIZEFACTORL=%d NSIZEFACTORU=%d NSIZEINDEXL=%d",
       icon, nsizefactorl, nsizefactoru, nsizeindexl);
printf(" NSIZEINDEXU=%d NSUPNUM=%d\n", nsizeindexu, nsupnum);

zpanelfactorl = (dcomplex *)malloc(nsizefactorl * sizeof(dcomplex));
zpanelfactoru = (dcomplex *)malloc(nsizefactoru * sizeof(dcomplex));
npanelindexl = (int *)malloc(nsizeindexl * sizeof(int));
npanelindexu = (int *)malloc(nsizeindexu * sizeof(int));

isw = 2;
```

```
c_dm_vsclu(za, nz, nrow, nfcnz, N,
          ipledsm, mz, isclitermax, &iordering,
          nperm, isw,
          nrowsym, nfcnzsym,
          nassign,
          &nsupnum,
          nfcnzfactorl, zpanelfactorl,
          &nsizefactorl,
          nfcnzindexl,
          npanelindexl, &nsizeindexl,
          (int *)ndim,
          nfcnzfactoru, zpanelfactoru,
          &nsizefactoru,
          nfcnzindexu,
          npanelindexu, &nsizeindexu,
          nposto,
          sclrow, sclcol,
          &epsz, &thepsz,
          ipivot, istatic, &spepsz, nfcnzpivot,
          npivotp, npivotq,
          zw, w, iw1, iw2, &icon);

c_dm_vsclux(N,
            iordering,
            nperm,
            zb,
            nassign,
            nsupnum,
            nfcnzfactorl, zpanelfactorl,
            nsizefactorl,
            nfcnzindexl,
            npanelindexl, nsizeindexl,
            (int *)ndim,
            nfcnzfactoru, zpanelfactoru,
            nsizefactoru,
            nfcnzindexu,
            npanelindexu, nsizeindexu,
            nposto,
            ipledsm, mz,
            sclrow, sclcol,
            nfcnzpivot,
            npivotp, npivotq,
            irefine, epsr, itermax, &iter,
            za, nz, nrow, nfcnz,
            iw2, &icon);

err = errnrm(zsolex, zb, N);
```

```
  printf("    COMPUTED VALUES\n");
  printf("    X(1) = (%lf,%lf) X(N) = (%lf,%lf)\n\n", zb[0], zb[N - 1]);
  printf("    ICON = %d\n\n", icon);
  printf("    N = %d\n\n", N);
  printf("    ERROR = %lf\n", err);
  printf("    ITER=%d\n\n\n", iter);

  if (err < 1.0e-8 && icon == 0) {
    printf("********** OK **********\n");
  } else {
    printf("********** NG **********\n");
  }

  free(zpanelfactorl);
  free(zpanelfactoru);
  free(npanelindexl);
  free(npanelindexu);

  return(0);
}

/* ========================================
      INITIALIZE COEFFICIENT MATRIX
  ======================================== */
void init_mat_diag(double va1, double va2, double va3, double vc,
                   double *d_l, int *offset,
                   int nx, int ny, int nz, double xl, double yl, double zl,
                   int ndiag, int len, int ndivp) {

  if (ndiag < 1) {
    printf("FUNCTION INIT_MAT_DIAG:\n");
    printf(" NDIAG SHOULD BE GREATER THAN OR EQUAL TO 1\n");
    return;
  }

#pragma omp parallel default(shared)
{
  int i, j, l, ndiag_loc, nxy, js, k0, j0, i0;
  double hx, hy, hz, hx2, hy2, hz2;

/* NDIAG CANNOT BE GREATER THAN 7 */
  ndiag_loc = ndiag;
  if (ndiag > 7)
    ndiag_loc = 7;

/* INITIAL SETTING */
  hx = xl / (nx + 1);
  hy = yl / (ny + 1);
  hz = zl / (nz + 1);
```

```
#pragma omp for
  for (i = 0; i < ndivp; i++) {
    for (j = 0; j < ndiag; j++) {
      d_l[(j * ndivp) + i] = 0.0;
    }
  }

  nxy = nx * ny;

/* OFFSET SETTING */
#pragma omp single
  {
    l = 0;
    if (ndiag_loc >= 7) {
      offset[l] = -nxy;
      l++;
    }
    if (ndiag_loc >= 5) {
      offset[l] = -nx;
      l++;
    }
    if (ndiag_loc >= 3) {
      offset[l] = -1;
      l++;
    }
    offset[l] = 0;
    l++;
    if (ndiag_loc >= 2) {
      offset[l] = 1;
      l++;
    }
    if (ndiag_loc >= 4) {
      offset[l] = nx;
      l++;
    }
    if (ndiag_loc >= 6) {
      offset[l] = nxy;
    }
  }

/* MAIN LOOP */
#pragma omp for
  for (j = 0; j < len; j++) {
    js = j + 1;

/* DECOMPOSE JS-1 = (K0-1)*NX*NY+(J0-1)*NX+I0-1 */
    k0 = (js -1) / nxy + 1;
    if (k0 > nz) {
```

```
      printf("ERROR; K0.GH.NZ \n");
      goto label_100;
    }
    j0 = (js - 1 - nxy * (k0 - 1)) / nx + 1;
    i0 = js - nxy * (k0 - 1) - nx * (j0 - 1);
    l = 0;

    if (ndiag_loc >= 7) {
      if (k0 > 1) d_l[(l * ndivp) + j] = -(1.0 / hz + 0.5 * va3) / hz;
      l++;
    }
    if (ndiag_loc >= 5) {
      if (j0 > 1) d_l[(l * ndivp) + j] = -(1.0 / hy + 0.5 * va2) / hy;
      l++;
    }
    if (ndiag_loc >= 3) {
      if (i0 > 1) d_l[(l * ndivp) + j] = -(1.0 / hx + 0.5 * va1) / hx;
      l++;
    }
    hx2 = hx * hx;
    hy2 = hy * hy;
    hz2 = hz * hz;
    d_l[(l * ndivp) + j] = 2.0 / hx2 + vc;
    if (ndiag_loc >= 5) {
      d_l[(l * ndivp) + j] += 2.0 / hy2;
      if (ndiag_loc >= 7) {
        d_l[(l * ndivp) + j] += 2.0 / hz2;
      }
    }
    l++;
    if (ndiag_loc >= 2) {
      if (i0 < nx) d_l[(l * ndivp) + j] = -(1.0 / hx - 0.5 * va1) / hx;
      l++;
    }
    if (ndiag_loc >= 4) {
      if (j0 < ny) d_l[(l * ndivp) + j] = -(1.0 / hy - 0.5 * va2) / hy;
      l++;
    }
    if (ndiag_loc >= 6) {
      if (k0 < nz) d_l[(l * ndivp) + j] = -(1.0 / hz - 0.5 * va3) / hz;
    }
label_100: ;
  }

}

  return;
}
```

```
/* ======================================
  * SOLUTE ERROR
  * | Z1 - Z2 |
   ====================================== */
double errnrm(dcomplex *z1, dcomplex *z2, int len) {
  double  rtc, s;
  dcomplex  ss;
  int i;

  s = 0.0;
  for (i = 0; i < len; i++) {
    ss = comp_sub(z1[i], z2[i]);
    s += ss.re * ss.re + ss.im * ss.im;
  }

  rtc = sqrt(s);
  return(rtc);
}

dcomplex comp_sub(dcomplex so1, dcomplex so2) {

  dcomplex obj;

  obj.re = so1.re - so2.re;
  obj.im = so1.im - so2.im;
  return obj;
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VSCLU の項目及び[2], [13], [17], [19], [22], [23], [48], [57], [63], [68], [69]を参照してください.

c_dm_vsclux

LU 分解された複素スパース行列の連立 1 次方程式
ierr = c_dm_vsclux(n, iordering, nperm, zb, nassign, nsupnum, nfcnzfactorl, zpanelfactorl, nsizefactorl, nfcnzindexl, npanelindexl, nsizeindexl, ndim, nfcnzfactoru, zpanelfactoru, nsizefactoru, nfcnzindexu, npanelindexu, nsizeindexu, nposto, ipledsm, mz, sclrow, sclcol, nfcnzpivot, npivotp, npivotq, irefine, epsr, itermax, &iter, za, nz, nrow, nfcnz, iw2, &icon);

1. 機能

$n \times n$ の複素スパース行列 **A** に対角要素に大きな要素を並べる並び換えを行い, 行および列の均衡化を行うスケーリングを行い, そしてスーパーノード内で Pivot をとり, 以下のように LU 分解されています. LU 分解の結果を利用して以下の連立 1 次方程式を解きます.
要素の並び換えや スケーリングおよび Pivot では, 複素数の大きさを表す絶対値は, 実部の絶対値と虚部の絶対値を加えたもので近似します.

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

行列 **A** は以下のように分解されています.

$$\mathbf{P_{rs}QPD_rAP_cD_cP^TQ^TP_{cs}} = \mathbf{LU}$$

ここで, 複素スパース行列 **A** は以下のように変換されています.

$$\mathbf{A_1} = \mathbf{D_rAP_cD_c}$$

ここで $\mathbf{P_c}$ は列の入れ換えを行う直交行列, $\mathbf{D_r}$ は行のスケーリングを行う対角行列, $\mathbf{D_c}$ は列のスケーリングを行う対角行列です.

$$\mathbf{A_2} = \mathbf{QPA_1P^TQ^T}$$

$\mathbf{A_2}$ は, スーパーノード内に閉じて指定された Pivot を行い, 行および列の入れ換えを行い LU 分解されています.
$\mathbf{P_{rs}}$, $\mathbf{P_{cs}}$ はそれぞれ行の入れ換えおよび列の入れ換えを示す直交行列です. 実際の入れ換えはスーパーノードに属する限られた行列の部分で行われます.
ただし, **P** は, $\mathbf{SYM} = \mathbf{A_1} + \mathbf{A_1^T}$ の非ゼロパターンに対して求めた `ordering` による行列要素の並び換えを示す置換行列で, **Q** は **SYM** に対する post order による行列要素の並び換えを示す置換行列を示します. **P**, **Q** は直交行列です.

L は下三角行列, **U** は単位上三角行列です.
解の反復改良で精度を改良することを指定できます.

2. 引数

呼出し形式:

```
ierr = c_dm_vsclux(n, iordering, nperm, zb, nassign, nsupnum, nfcnzfactorl,
            zpanelfactorl, nsizefactorl, nfcnzindexl, npanelindexl,
            nsizeindexl, (int *)ndim, nfcnzfactoru, zpanelfactoru,
            nsizefactoru, nfcnzindexu, npanelindexu, nsizeindexu, nposto,
            ipledsm, mz, sclrow, sclcol, nfcnzpivot, npivotp, npivotq,
            irefine, epsr, itermax, &iter, za, nz, nrow, nfcnz, iw2, &icon);
```

引数の説明:

n	int	入力	行列 **A** の次数 n.
iordering	int	入力	11 のとき, nperm に ordering を指定して LU 分解されたことを示します. $\mathbf{PA_1P^T}$ を LU 分解しました. 11 以外のとき, ordering は指定されていません. (“使用上の注意” a)参照)
nperm	int nperm[n]	入力	iordering = 11 のとき使用する置換行列をベクトルで指定します. (“使用上の注意” b)参照)
zb	dcomplex zb[n]	入力 出力	**Ax** = **b** の右辺定数ベクトル. 解ベクトル.
nassign	int nassign[n]	入力	各 supernode に対する **L**, **U** は圧縮して 2 次元の panel に格納します. この panel を zpanelfactorl および zpanelfactoru の 1 次元部分配列として順番に割り付けたとき何番目になるかを示します. これらの指標ベクトルも同じように npanelindexl, npanelindexu に割り付けます. j = nassign[i-1] のとき, *i* 番目の supernode を *j* 番目に割り付けたことを示します. 分解結果の格納方法については図 c_dm_vsclux-1 を参照してください.
nsupnum	int	入力	supernode の総数. (≤ n)
nfcnzfactorl	long nfcnzfactorl[n+1]	入力	複素スパース行列の LU 分解の結果の行列 **L** および **U** はスーパーノードに対応しておのおの求めます. 各 supernode に対応する **L** の列ベクトルは, 共通の行指標ベクトルを持つように圧縮してブロック対角部分に **U** の対応部分を含めて 2 次元の panel に格納します. この panel を zpanelfactorl の 1 次元部分配列として順番に割り付けたとき, *i* 番目の panel の先頭要素 panel[0][0]が 1 次元配列の zpanelfactorl の何番目の要素になるかを示します. 結果の格納方法については図 c_dm_vsclux-1 を参照してください.
zpanelfactorl	dcomplex zpanelfactorl [nsizefactorl]	入力	各 supernode の分解結果に対応する行列 **L** の複数の列ベクトルと行列 **U** のブロック対角部分について, 共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します. panel を順番に割り付けます. *i* 番目の supernode に対応する panel が何番目の位置に割り当てられるかは j = nassign[i-1]から分かります. その先頭位置は nfcnzfactorl[j-1]に格納さ

			れています. panel ごとに分解結果が格納されます. *i* 番目に割付けられた panel の大きさは ndim[i-1][0] × ndim[i-1][1]の 2 次元配列と見なせます. *i* 番目の panel の panel[t-1][s-1], s ≥ t, s = 1,…, ndim[i-1][0], t = 1,…, ndim[i-1][1]に下三角行列 **L** が格納されます. panel の panel[t-1][s-1], s < t, t = 1,…, ndim[i-1][1]には単位上三角行列 **U** の対角要素を除いたブロック対角部分が格納されます. 結果の格納方法については図 c_dm_vsclux-1 を参照してください.
nsizefactorl	long	入力	zpanelfactorl の大きさを示す.
nfcnzindexl	long nfcnzindexl[n+1]	入力	各 supernode の分解結果に対応する行列 **L** の複数の列ベクトルと行列 **U** のブロック対角部分について, 共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します. この行指標ベクトルを npanelindexl に順番に割り付けたとき *i* 番目の行指標ベクトルの先頭要素が 1 次元配列である npanelindexl の何番目の要素になるかを示します.
npanelindexl	int npanelindexl [nsizeindexl]	入力	各 supernode の分解結果に対応する行列 **L** の複数の列ベクトルと行列 **U** のブロック対角部分について, 共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します. この行指標ベクトルを npanelindexl に順番に割り付けます. i 番目の supernode に対応する panel の行の指標ベクトルが何番目に割り当てられるかは j = nassign[i-1]から分かります. その先頭位置は nfcnzindexl[j-1]に格納されています. この行指標は行列 **SYM** に対する post order で並び換えたときの行の番号です. 結果の格納方法については図 c_dm_vsclux-1 を参照してください.
nsizeindexl	long	入力	npanelindexl の大きさを示す.
ndim	int ndim[n][3]	入力	ndim[i-1][0]および ndim[i-1][1]は行列 **L** に関して *i* 番目に割り付けられた panel の 1 次元目と 2 次元目の大きさを示します. ndim[i-1][2]は行列 **U** に関して割り付けられた panel を転置したときの 1 次元目の大きさに対角ブロックの大きさを加えた大きさを示します. 結果の格納方法については図 c_dm_vsclux-1 を参照してください.
nfcnzfactoru	long nfcnzfactoru[n+1]	入力	複素スパース行列の LU 分解の結果の行列 **U** に関しては, 各 supernode に対応する **U** の行ベクトルは, ブロック対角部分を除いて共通の列指標ベクトルを持つように圧縮し, 転置して 2 次元の panel に格納します. この panel を zpanelfactoru の 1 次元部分配列として順番に割り付けたとき, i 番目の panel の先頭要素 panel[0][0]が 1 次元配列の zpanelfactoru の何番目の要素になるかを示します. 結果の格納方法については図 c_dm_vsclux-1 を参照し

			てください.
zpanelfactoru	dcomplex zpanelfactoru [nsizefactoru]	入力	各 supernode の分解結果に対応する行列 **U** のブロック対角部分を除いた複数の行ベクトルについて, 共通の列指標ベクトルを持つように圧縮し, 転置して 2 次元の panel に格納します. panel を順番に割り付けます. *i* 番目の supernode に対応する panel が何番目の位置に割り当てられるかは j = nassign[i-1]から分かります. その先頭位置は nfcnzfactoru[j-1]に格納されています. panel ごとに分解結果が格納されます. *i* 番目に割付けられた panel の大きさは{ndim[i-1][2] – ndim[i-1][1]} × ndim[i-1][1]の 2 次元配列と見なせます. *i* 番目の panel の panel[t-1][s-1], s = 1,…, ndim[i-1][2] – ndim[i-1][1], t = 1,…, ndim[i-1][1]に単位上三角行列 **U** のブロック対角部分を除いた部分が行ベクトルを圧縮し, 転置したものが格納されます. 結果の格納方法については図 c_dm_vsclux-1 を参照してください.
nsizefactoru	long	入力	zpanelfactoru の大きさを示す.
nfcnzindexu	long nfcnzindexu[n+1]	入力	各 supernode の分解結果に対応する行列 **U** は対角ブロック部分を除いて複数の行ベクトルについて, 共通の列指標ベクトルを持つように圧縮し, 転置した形で 2 次元の panel に格納します. 対角ブロック部分も含んだこの列指標ベクトルを npanelindexu に順番に割り付けたとき *i* 番目の列指標ベクトルの先頭要素が 1 次元配列である npanelindexu の何番目の要素になるかを示します. 結果の格納方法については図 c_dm_vsclux-1 を参照してください.
npanelindexu	int npanelindexu [nsizeindexu]	入力	各 supernode の分解結果に対応する行列 **U** の対角ブロックを除いた部分を転置して圧縮して 2 次元の panel に格納します. 対応する列指標ベクトルには対角ブロック部分を含めたものを npanelindexu に順番に割り付けます. *i* 番目の supernode に対応する panel の行の指標ベクトルが何番目に割り当てられるかは j = nassign[i-1]から分かります. その先頭位置は nfcnzindexu[j-1]に格納されています. この行指標は行列 **SYM** に対する post order で並び換えたときの列の番号です. 結果の格納方法については図 c_dm_vsclux-1 を参照してください.
nsizeindexu	long	入力	npanelindexu の大きさを示す.
nposto	int nposto[n]	入力	post order で *i* 番目の node が行列 **A** の何番目の列番号に対応するかを表す 1 次元ベクトル. (“使用上の注意” c)参照)
ipledsm	int	入力	LU 分解を行うときに対角要素に大きな要素を並べる列の入れ換えを行ったかを指定します. 1 のとき, 列の入れ換えを行いました. 1 以外のとき, 列の入れ換えを行っていません.

mz	int mz[n]	入力	ipledsm に 1 が指定されたとき, 列の入れ換えを表す. mz[i-1] = *j* は行列 $\mathbf{a}_{ij}$ の属する *j* 番目の列を *i* 番目の列に移動する. $\mathbf{a}_{ij}$ は対角要素に並べる大きな要素を表す.
sclrow	double sclrow[n]	入力	スケーリング対角行列 $\mathbf{D}_r$. 対角要素が 1 次元配列に格納されます.
sclcol	double sclcol[n]	入力	スケーリング対角行列 $\mathbf{D}_c$. 対角要素が 1 次元配列に格納されます.
nfcnzpivot	int nfcnzpivot [nsupnum+1]	入力	スーパーノード内の相対的な位置でのピボットの行, 列の入れ換えの履歴を格納する位置を示す. *i* 番目の supernode に関する情報が何番目の位置に割り当てられるかは j = nassign[i-1]から分かります.その先頭位置は nfcnzpivot[j-1]に格納されています. *i* 番目のスーパーノードの入れ換え情報は, npivotp, npivotq の is = nfcnzpivot[j-1],..., ie = nfcnzpivot[j-1] + ndim[j-1][1] - 1 番目の要素に格納されます.
npivotp	int npivotp[n]	入力	スーパーノード内の行の入れ換えに関する情報を格納する.
npivotq	int npivotq[n]	入力	スーパーノード内の列の入れ換えに関する情報を格納する.
irefine	int	入力	LU 分解結果を利用して解を求めるときに, 解の反復改良を行うかどうかを示す. 残差ベクトルを計算するときに 4 倍精度で計算します. =1 : 解の反復改良を行う. 反復改良して得られる残差ベクトル $\mathbf{r}_k$ の絶対値の差分がひとつ前の差分の 1/4 より大きくなるまで, 反復改良を行います. ≠1: 解の反復改良を行わない.
epsr	double	入力	解の残差ベクトル **b-Ax** の絶対値が, **b** の絶対値に対して十分小さいかを判定する判定値 epsr ≤ 0.0 のとき 10^{-6} が設定されます.
itermax	int	入力	(≥ 1) 反復改良を行うときの最大反復回数.
iter	int	出力	反復改良を行った回数.
za	dcomplex za[nz]	入力	複素スパースな係数行列 **A** を圧縮列格納法で格納します. 圧縮列格納法については, 実スパース行列と実ベクトル(圧縮列格納法), c_dm_vmvscc の図 c_dm_vmvscc-1 を参照してください. 実数型の配列 a を複素数型の配列に変えたものを利用します.
nz	int	入力	複素スパースな係数行列 **A** の非零要素の総数.
nrow	int nrow[nz]	入力	圧縮列格納法で使用される行指標で za に格納される要素が何番目の行ベクトルに属するかを示します.
nfcnz	int nfcnz[n+1]	入力	行列 **A** の各列の非零要素を列方向に圧縮して順次配列 za に格納するとき, 対応する列の最初の非零要素が格納される位置を表します. nfcnz[n] = nz + 1.
iw2	int iw2[47*n+47+nz+4*(n+1)+2*(nz+n)]	作業域	複素スパース行列の LU 分解を行う c_dm_vsclu からのデータの受け渡しに使われます. 呼び出しの間で値を変更してはいけません.
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.
20400	LU 分解された行列の対角要素にゼロがあった	処理を打ち切る.
20500	求めた解ベクトルに対する残差ベクトルのノルムが方程式 **Ax** = **b** の右辺ベクトルのノルムの epsr 倍より大きい. 係数行列は特異に近い可能性がある.	
30000	次のいずれかであった: • n < 1 • nz < 0 • nfcnz[n] ≠ nz + 1 • nsizefactorl < 1 • nsizefactoru < 1 • nsizeindexl < 1 • nsizeindexu < 1 • irefine = 1 のとき itermax < 1	
30100	nperm に指定した置換行列が正しくない.	
30200	nrow[j-1]に格納された *i* 列目の行指標 k が k < 1 または k > n	
30300	*i* 列目の行指標の数 nfcnz[i] – nfcnz[i-1] > n	

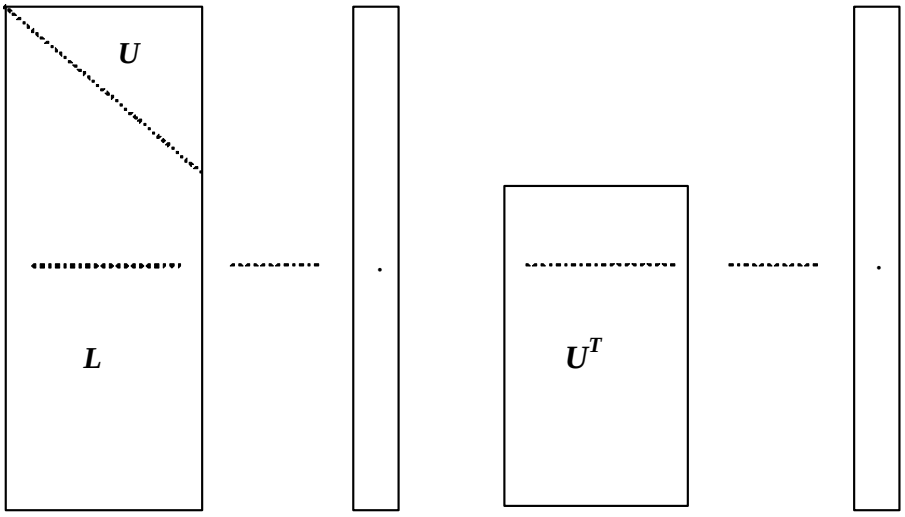


図 c_dm_vsclux-1 分解結果の格納概念図

j = nassign[i-1]　→ *i* 番目の supernode は *j* 番目に格納されます.

p = nfcnzfactorl[j-1] → *j* 番目の panel は zpanelfactorl の *p* 番目の要素から ndim[j-1][0] × ndim[j-1][1] の長さを占めます.

q = nfcnzindexl[j-1] → *j* 番目の panel の行指標を表すベクトルは npanelindexl の *q* 番目の要素から ndim[j-1][0] の長さを占めます.

panel は大きさ ndim[j-1][0] × ndim[j-1][1]の配列と見なせます.

panel[t-1][s-1], s ≥ t, s = 1,… , ndim[j-1][0], t = 1,…, ndim[j-1][1] に分解結果の下三角行列 **L** が格納されます.

panel[t-1][s-1], s < t, s = 1,…, ndim[j-1][1], t = 1,…, ndim[j-1][1] に単位上三角行列 **U** の対角ブロック部分が対角要素を除き格納されます.

u = nfcnzfactoru[j-1] → *j* 番目の panel は zpanelfactoru の *u* 番目の要素から(ndim[j-1][2] – ndim[j-1][1]) × ndim[j-1][1] の長さを占めます.

v = nfcnzindexu[j-1] → *j* 番目の panel の列指標を表すベクトルは npanelindexu の *v* 番目の要素から ndim[j-1][2]の長さを占めます.

panel は大きさ(ndim[j-1][2] – ndim[j-1][1]) × ndim[j-1][1] の配列と見なせます.

panel[y-1][x-1], x = 1,…, ndim[j-1][2] - ndim[j-1][1], y = 1,…, ndim[j-1][1]

に分解結果の単位上三角行列 **U** の転置行列 $\mathbf{U}^{\mathrm{T}}$ の対角ブロック部分を除いた部分が格納されます.

指標の値は行列 **A** の列番号をもつ node を post order で並び換えた $\mathbf{QAQ}^{\mathrm{T}}$ での列番号を表します.

3. 使用上の注意

a)

c_dm_vsclu で LU 分解を行った結果を利用します.
c_dm_vsclu の“使用上の注意” e)参照や c_dm_vsclux の使用例を参照願います.

b)

直交行列である置換行列 **P** の要素 $p_{ij} = 1$ のとき, nperm[i-1] = j と表現します.
逆は以下のようにして求めることができます.

```
for (i = 1; i <= n; i++) {
 j = nperm[i-1];
 npermінv[j-1] = i;
}
```

c)

列番号に対応するノードを考えます. これを post order で順番を並び換えたときの番号の対応関係が nposto に格納されています. post order の *i* 番目のノードが並び換える前の何番目のノードに対応するかを表します. j=nposto[i-1]は *j* 番目であることを表します.
b)同様にこれは直交行列である置換行列 **Q** を表し, 行列 **A** を $\mathbf{QAQ}^{\mathrm{T}}$ と並び換えることに相当します.
逆変換 $\mathbf{Q}^{\mathrm{T}}$ は以下のようにして求めることができます.

```
for (i = 1; i <= n; i++) {
 j = nposto[i-1];
 npostoinv[j-1] = i;
}
```

4. 使用例

連立 1 次方程式 **Ax** = **f** を解きます.
行列は境界条件として立方体の境界で 0 となる楕円型の偏微分方程式に有限差分法を適用して生成されたものを利用します.

$$-\Delta u + a\nabla u + cu = f$$

ここで $a = (a_1, a_2, a_3)$.
行列は関数 init_mat_diag によって生成されます.
対角形式格納法で生成し, 対角ベクトルの下 6 本を圧縮列格納法で格納します. 結果生成された非対称な行列 **A** に関する連立 1 次方程式を解きます.
(並列実行を行うスレッド数は環境変数(OMP_NUM_THREADS)などで指定できます.
例えば, 4 プロセッサーのシステムで 4 つのスレッドで並列実行するときは, OMP_NUM_THREADS を 4 に設定して実行します.)

```
 /* **EXAMPLE** */
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <malloc.h>
#include <omp.h>
#include "cssl.h"

#define  NORD  40
#define  KX  NORD
#define  KY  NORD
#define  KZ  NORD
#define  N  KX * KY * KZ
#define  NBORDER  (N + 1)
#define  NOFFDIAG  6
#define  K  (N + 1)
#define  NDIAG  7
#define  NALL  NDIAG * N

#define  ZWL  2 * NALL
#define  WL  4 * NALL + 6 * N
#define  IW1L  2 * NALL + 2 * (N + 1) + 16 * N
#define  IW2L  47 * N + 47 + 4 * (N + 1) + NALL + 2 * (NALL + N)

void init_mat_diag(double, double, double, double, double*, int*, int, int, int,
                   double, double, double, int, int, int);
double errnrm(dcomplex*, dcomplex*, int);
dcomplex comp_sub(dcomplex, dcomplex);

int MAIN__() {

  int  nofst[NDIAG];
  double  diag[NDIAG][K], diag2[NDIAG][K];
  dcomplex  za[K * NDIAG], zwc[K * NDIAG],
            zw[ZWL], zone;
  int  nrow[K * NDIAG], nfcnz[N + 1],
       nrowsym[K * NDIAG + N], nfcnzsym[N + 1],
```

```
     iwc[K * NDIAG][2];
int  nperm[N],
     nposto[N], ndim[N][3],
     nassign[N],
     mz[N],
     iw1[IW1L], iw2[IW2L];
double  w[WL];
dcomplex  *zpanelfactorl, *zpanelfactoru;
int  *npanelindexl, *npanelindexu;
dcomplex  zdummyfl, zdummyfu;
int  ndummyil,
     ndummyiu;
long  nsizefactorl,
      nsizeindexl,
      nsizeindexu,
      nsizefactoru,
      nfcnzfactorl[N + 1],
      nfcnzfactoru[N + 1],
      nfcnzindexl[N + 1],
      nfcnzindexu[N + 1];
dcomplex  zb[N], zsolex[N];
double  epsz, thepsz, spepsz,
        sclrow[N], sclcol[N];

int  ipivot, istatic, nfcnzpivot[N + 1],
     npivotp[N], npivotq[N],
     irefine, itermax, iter, ipledsm;
double  err, va1, va2, va3, vc, xl, yl, zl, epsr;
int  i, j, nbase, length, numnz, ntopcfg, ncol, nz, icon, iordering,
     isclitermax, isw, nsupnum;

zone.re = 1.0;
zone.im = 0.0;

printf("    LU DECOMPOSITION METHOD\n");
printf("    FOR SPARSE UNSYMMETRIC COMPLEX MATRICES\n");
printf("    IN COMPRESSED COLUMN STORAGE\n\n");

for (i = 0; i < N; i++) {
  zsolex[i] = zone;
}
printf("    EXPECTED SOLUTIONS\n");
printf("    X(1) = (%lf,%lf) X(N) = (%lf,%lf)\n\n",
       zsolex[0].re, zsolex[0].im, zsolex[N - 1].re, zsolex[N - 1].im);

va1 = 1.0;
va2 = 2.0;
va3 = 3.0;
vc = 4.0;
```

```
xl = 1.0;
yl = 1.0;
zl = 1.0;
init_mat_diag(va1, va2, va3, vc, (double *)diag, nofst,
              KX, KY, KZ, xl, yl, zl, NDIAG, N, K);

for (i = 0; i < NDIAG; i++) {
  for (j = 0; j < K; j++) {
    diag2[i][j] = 0;
  }
}

for (i = 0; i < NDIAG; i++) {

  if (nofst[i] < 0) {
    nbase = -nofst[i];
    length = N - nbase;
    for (j = 0; j < length; j++) {
      diag2[i][j] = diag[i][nbase + j];
    }
  } else {
    nbase = nofst[i];
    length = N - nbase;
    for (j = 0; j < length; j++) {
      diag2[i][nbase + j] = diag[i][j];
    }
  }

}

numnz = 1;

for (j = 0; j < N; j++) {
  ntopcfg = 1;

  for (i = NDIAG - 1; i >= 0; i--) {

    if (ntopcfg == 1) {
      nfcnz[j] = numnz;
      ntopcfg = 0;
    }

    if (j + 1 < NBORDER && i + 1 > NOFFDIAG) {
      continue;
    } else {

      if (diag2[i][j] != 0.0) {

        ncol = (j + 1) - nofst[i];
```

```
            za[numnz - 1].re = diag2[i][j];
            za[numnz - 1].im = 0.0;
            nrow[numnz - 1] = ncol;

            numnz++;

          }
        }
      }
    }

    nfcnz[N] = numnz;
    nz = numnz - 1;

    c_dm_vmvsccc(za, nz, nrow, nfcnz, N, zsolex,
                 zb, zwc, (int *)iwc, &icon);

  /* INITIAL CALL WITH IORDER=1 */

    iordering = 0;
    ipledsm = 1;
    isclitermax = 10;
    isw = 1;
    nsizefactorl = 1;
    nsizefactoru = 1;
    nsizeindexl = 1;
    nsizeindexu = 1;
    epsz = 1.0e-16;
    thepsz = 1.0e-2;
    spepsz = 0.0;
    ipivot = 40;
    istatic = 0;
    irefine = 1;
    epsr = 0.0;
    itermax = 10;

    c_dm_vsclu(za, nz, nrow, nfcnz, N,
               ipledsm, mz, isclitermax, &iordering,
               nperm, isw,
               nrowsym, nfcnzsym,
               nassign,
               &nsupnum,
               nfcnzfactorl, &zdummyfl,
               &nsizefactorl,
               nfcnzindexl,
               &ndummyil, &nsizeindexl,
               (int *)ndim,
               nfcnzfactoru, &zdummyfu,
               &nsizefactoru,
```

```
          nfcnzindexu,
          &ndummyiu, &nsizeindexu,
          nposto,
          sclrow, sclcol,
          &epsz, &thepsz,
          ipivot, istatic, &spepsz, nfcnzpivot,
          npivotp, npivotq,
          zw, w, iw1, iw2, &icon);

printf("ICON=%d NSIZEFACTORL=%d NSIZEFACTORU=%d NSIZEINDEXL=%d",
       icon, nsizefactorl, nsizefactoru, nsizeindexl);
printf(" NSIZEINDEXU=%d NSUPNUM=%d\n", nsizeindexu, nsupnum);

zpanelfactorl = (dcomplex *)malloc(nsizefactorl * sizeof(dcomplex));
zpanelfactoru = (dcomplex *)malloc(nsizefactoru * sizeof(dcomplex));
npanelindexl = (int *)malloc(nsizeindexl * sizeof(int));
npanelindexu = (int *)malloc(nsizeindexu * sizeof(int));

isw = 2;

c_dm_vsclu(za, nz, nrow, nfcnz, N,
          ipledsm, mz, isclitermax, &iordering,
          nperm, isw,
          nrowsym, nfcnzsym,
          nassign,
          &nsupnum,
          nfcnzfactorl, zpanelfactorl,
          &nsizefactorl,
          nfcnzindexl,
          npanelindexl, &nsizeindexl,
          (int *)ndim,
          nfcnzfactoru, zpanelfactoru,
          &nsizefactoru,
          nfcnzindexu,
          npanelindexu, &nsizeindexu,
          nposto,
          sclrow, sclcol,
          &epsz, &thepsz,
          ipivot, istatic, &spepsz, nfcnzpivot,
          npivotp, npivotq,
          zw, w, iw1, iw2, &icon);

c_dm_vsclux(N,
            iordering,
            nperm,
            zb,
            nassign,
            nsupnum,
            nfcnzfactorl, zpanelfactorl,
```

```
                nsizefactorl,
                nfcnzindexl,
                npanelindexl, nsizeindexl,
                (int *)ndim,
                nfcnzfactoru, zpanelfactoru,
                nsizefactoru,
                nfcnzindexu,
                npanelindexu, nsizeindexu,
                nposto,
                ipledsm, mz,
                sclrow, sclcol,
                nfcnzpivot,
                npivotp, npivotq,
                irefine, epsr, itermax, &iter,
                za, nz, nrow, nfcnz,
                iw2, &icon);

  err = errnrm(zsolex, zb, N);

  printf("    COMPUTED VALUES\n");
  printf("    X(1) = (%lf,%lf) X(N) = (%lf,%lf)\n\n", zb[0], zb[N - 1]);
  printf("    ICON = %d\n\n", icon);
  printf("    N = %d\n\n", N);
  printf("    ERROR = %lf\n", err);
  printf("    ITER=%d\n\n\n", iter);

  if (err < 1.0e-8 && icon == 0) {
    printf("********** OK **********\n");
  } else {
    printf("********** NG **********\n");
  }

  free(zpanelfactorl);
  free(zpanelfactoru);
  free(npanelindexl);
  free(npanelindexu);

  return(0);
}

/* ======================================
      INITIALIZE COEFFICIENT MATRIX
  ====================================== */
void init_mat_diag(double va1, double va2, double va3, double vc,
                   double *d_l, int *offset,
                   int nx, int ny, int nz, double xl, double yl, double zl,
                   int ndiag, int len, int ndivp) {

  if (ndiag < 1) {
```

```
    printf("FUNCTION INIT_MAT_DIAG:\n");
    printf(" NDIAG SHOULD BE GREATER THAN OR EQUAL TO 1\n");
    return;
  }

#pragma omp parallel default(shared)
{
  int i, j, l, ndiag_loc, nxy, js, k0, j0, i0;
  double hx, hy, hz, hx2, hy2, hz2;

/* NDIAG CANNOT BE GREATER THAN 7 */
  ndiag_loc = ndiag;
  if (ndiag > 7)
    ndiag_loc = 7;

/* INITIAL SETTING */
  hx = xl / (nx + 1);
  hy = yl / (ny + 1);
  hz = zl / (nz + 1);

#pragma omp for
  for (i = 0; i < ndivp; i++) {
    for (j = 0; j < ndiag; j++) {
      d_l[(j * ndivp) + i] = 0.0;
    }
  }

  nxy = nx * ny;

/* OFFSET SETTING */
#pragma omp single
  {
    l = 0;
    if (ndiag_loc >= 7) {
      offset[l] = -nxy;
      l++;
    }
    if (ndiag_loc >= 5) {
      offset[l] = -nx;
      l++;
    }
    if (ndiag_loc >= 3) {
      offset[l] = -1;
      l++;
    }
    offset[l] = 0;
    l++;
    if (ndiag_loc >= 2) {
      offset[l] = 1;
```

```
      l++;
    }
    if (ndiag_loc >= 4) {
      offset[l] = nx;
      l++;
    }
    if (ndiag_loc >= 6) {
      offset[l] = nxy;
    }
  }

/* MAIN LOOP */
#pragma omp for
  for (j = 0; j < len; j++) {
    js = j + 1;

/* DECOMPOSE JS-1 = (K0-1)*NX*NY+(J0-1)*NX+I0-1 */
    k0 = (js -1) / nxy + 1;
    if (k0 > nz) {
      printf("ERROR; K0.GH.NZ \n");
      goto label_100;
    }
    j0 = (js - 1 - nxy * (k0 - 1)) / nx + 1;
    i0 = js - nxy * (k0 - 1) - nx * (j0 - 1);
    l = 0;

    if (ndiag_loc >= 7) {
      if (k0 > 1) d_l[(l * ndivp) + j] = -(1.0 / hz + 0.5 * va3) / hz;
      l++;
    }
    if (ndiag_loc >= 5) {
      if (j0 > 1) d_l[(l * ndivp) + j] = -(1.0 / hy + 0.5 * va2) / hy;
      l++;
    }
    if (ndiag_loc >= 3) {
      if (i0 > 1) d_l[(l * ndivp) + j] = -(1.0 / hx + 0.5 * va1) / hx;
      l++;
    }
    hx2 = hx * hx;
    hy2 = hy * hy;
    hz2 = hz * hz;
    d_l[(l * ndivp) + j] = 2.0 / hx2 + vc;
    if (ndiag_loc >= 5) {
      d_l[(l * ndivp) + j] += 2.0 / hy2;
      if (ndiag_loc >= 7) {
        d_l[(l * ndivp) + j] += 2.0 / hz2;
      }
    }
    l++;
```

```
    if (ndiag_loc >= 2) {
      if (i0 < nx) d_l[(l * ndivp) + j] = -(1.0 / hx - 0.5 * va1) / hx;
      l++;
    }
    if (ndiag_loc >= 4) {
      if (j0 < ny) d_l[(l * ndivp) + j] = -(1.0 / hy - 0.5 * va2) / hy;
      l++;
    }
    if (ndiag_loc >= 6) {
      if (k0 < nz) d_l[(l * ndivp) + j] = -(1.0 / hz - 0.5 * va3) / hz;
    }
label_100: ;
  }

}

  return;
}

/* ======================================
  * SOLUTE ERROR
  * | Z1 - Z2 |
   ====================================== */
double errnrm(dcomplex *z1, dcomplex *z2, int len) {
  double  rtc, s;
  dcomplex  ss;
  int i;

  s = 0.0;
  for (i = 0; i < len; i++) {
    ss = comp_sub(z1[i], z2[i]);
    s += ss.re * ss.re + ss.im * ss.im;
  }

  rtc = sqrt(s);
  return(rtc);
}

dcomplex comp_sub(dcomplex so1, dcomplex so2) {

  dcomplex obj;

  obj.re = so1.re - so2.re;
  obj.im = so1.im - so2.im;
  return obj;
}
```

c_dm_vscs

複素スパース行列の連立 1 次方程式(LU 分解法)
ierr = c_dm_vscs(za, nz, nrow, nfcnz, n, ipledsm, mz, isclitermax, &iordering, nperm, isw, nrowsym, nfcnzsym, zb, nassign, &nsupnum, nfcnzfactorl, zpanelfactorl, &nsizefactorl, nfcnzindexl, npanelindexl, &nsizeindexl, ndim, nfcnzfactoru, zpanelfactoru, &nsizefactoru, nfcnzindexu, npanelindexu, &nsizeindexu, nposto, sclrow, sclcol, &epsz, &thepsz, ipivot, istatic, &spepsz, nfcnzpivot, npivotp, npivotq, irefine, epsr, itermax, &iter, zw, w, iw1, iw2, &icon);

1. 機能

$n \times n$ の複素スパース行列 **A** に対角要素に大きな要素を並べる並び換えを行い. 行および列の均衡化を行うスケーリングを行います.スーパーノードの対角ブロック内で指定した Pivot をとり, LU 分解し解きます.

要素の並び換えや スケーリングおよび Pivot では, 複素数の大きさを表す絶対値は, 実部の絶対値と虚部の絶対値を加えたもので近似します.

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

複素スパース行列 **A** は以下のように変換できます.

$$\mathbf{A_1} = \mathbf{D_r A P_c D_c}$$

ここで $\mathbf{P_c}$ は列の入れ換えを行う直交行列, $\mathbf{D_r}$ は行のスケーリングを行う対角行列, $\mathbf{D_c}$ は列のスケーリングを行う対角行列です.

$$\mathbf{A_2} = \mathbf{Q P A_1 P^T Q^T}$$

$\mathbf{A_2}$ は, スーパーノードの対角ブロックで指定された Pivot を行い, 行および列の入れ換えを行い LU 分解されます.

ただし, **P** は, $\mathbf{SYM} = \mathbf{A_1}+\mathbf{A_1^T}$ の非ゼロパターンに対して求めた ordering による行列要素の並び換えを示す置換行列で, **Q** は **SYM** に対する post order による行列要素の並び換えを示す置換行列を示します. **P**, **Q** は直交行列です.

L は下三角行列, **U** は単位上三角行列です.

Pivot 処理で, 閾値 thepsz より絶対値が大きな Pivot を見つけることが出来なかったとき, 対角ブロック内の絶対値が最大の要素を Pivot の候補とします. この値が小さ過ぎるときに Static Pivot を与えて近似的に LU 分解を続けることができます.

そして LU 分解の結果を利用して, 解を求めます.

また, 解の反復改良で精度を改良することを指定できます.

2. 引数

呼出し形式:

```
ierr = c_dm_vscs(za, nz, nrow, nfcnz, n, ipledsm, mz, isclitermax,
            &iordering, nperm, isw, nrowsym, nfcnzsym, zb, nassign, &nsupnum,
            nfcnzfactorl, zpanelfactorl, &nsizefactorl, nfcnzindexl,
            npanelindexl, &nsizeindexl, (int *)ndim, nfcnzfactoru,
            zpanelfactoru, &nsizefactoru, nfcnzindexu, npanelindexu,
            &nsizeindexu, nposto, sclrow, sclcol, &epsz, &thepsz, ipivot,
            istatic, &spepsz, nfcnzpivot, npivotp, npivotq, irefine, epsr,
            itermax, &iter, zw, w, iw1, iw2, &icon);
```

引数の説明:

za	dcomplex za[nz]	入力	複素スパースな係数行列 **A** を圧縮列格納法で格納します. 圧縮列格納法については, 実スパース行列と実ベクトル(圧縮列格納法), c_dm_vmvscc の図 c_dm_vmvscc-1 を参照してください. 実数型の配列 a を複素数型の配列に変えたものを利用します.
nz	int	入力	複素スパースな係数行列 **A** の非零要素の総数.
nrow	int nrow[nz]	入力	圧縮列格納法で使用される行指標で za に格納される要素が何番目の行ベクトルに属するかを示します.
nfcnz	int nfcnz[n+1]	入力	行列 **A** の各列の非零要素を列方向に圧縮して順次配列 za に格納するとき,対応する列の最初の非零要素が格納される位置を表します. nfcnz[n] = nz + 1.
n	int	入力	行列 **A** の次数 *n*.
ipledsm	int	入力	対角要素に大きな要素を並べる列の入れ換えを行うかを指定します. 1 のとき, 列の入れ換えを求めて入れ換えます. 1 以外のとき, 列の入れ換えを行いません.
mz	int mz[n]	出力	ipledsm に 1 が指定されたとき, 列の入れ換えを表す. mz[i] = j は行列 $\mathbf{a}_{ij}$ の属する *j* 番目の列を *i* 番目の列に移動する. $\mathbf{a}_{ij}$ は対角要素に並べる大きな要素を表す.
isclitermax	int	入力	係数行列のスケーリングを行う対角行列 $\mathbf{D}_r$ と $\mathbf{D}_c$ を反復して求める反復回数. isclitermax ≤ 0 のときスケーリングは行わすに, $\mathbf{D}_r$ および $\mathbf{D}_c$ は単位行列が設定されます. isclitermax ≥ 10 のときは 10 回を上限値とします.
iordering	int	入力	ordering を表す置換行列 **P** で $\mathbf{PA_1P^T}$ と変換した行列を LU 分解するかを指定します. 置換行列は直交行列です. 10 のとき, isw = 1 で呼び出すと ordering を求める $\mathbf{A_1}$ に関する情報を求めて nrowsym, nfcnzsym に設定します. 11 のとき, isw = 1 で 10 を指定して呼び出して得た行列を対称化した情報 nrowsym, nfcnzsym を使って決めた ordering を nperm に指定して同じく isw = 1 で呼び出し, 計算を続けることを示します. $\mathbf{PA_1P^T}$ を LU 分解する処理を続けます.

			10,11 以外のとき, ordering は指定せずに行列 $\mathbf{A}_1$ をそのまま LU 分解します.
		出力	isw = 1 と iordering = 10 を指定して呼び出した後, iordering に 11 が設定されます. そのため, ordering を nperm に指定して再び呼び出すときに特に 11 を指定する必要はありません. (“使用上の注意” a)参照)
nperm	int nperm[n]	入力	iordering = 11 のとき使用する置換行列をベクトルで指定します. (“使用上の注意” a)参照)
isw	int	入力	呼び出しに関する制御情報を示します. 1) 1 のとき 対称化および symbolic decomposition を行い, 必要な領域が割り当てられているか調べ計算を行います. iordering = 10 で呼び出し, ordering を求めるための情報が nrowsym, nfcnzsym に出力されます. これらを使って **SYM** に対する ordering を求めた後, nperm に指定して iordering = 11 を指定してもう 1 回 isw = 1 で呼び出します. iordering が 10,11 以外のときは ordering は行ないません. 2) 2 のとき isw = 1 で呼び出したとき, zpanelfactorl, zpanelfactoru, npanelindexl または npanelindexu の大きさが不足して icon = 31000 で終了した処理を継続します. このとき, nsizefactorl, nsizefactoru, nsizeindexl または nsizeindezu に返却された必要な大きさで zpanelfactorl, zpanelfactoru, npanelindexl または npanelindexu を確保し直して指定しなおして再度呼び出します. これらの引数と実行を制御する引数 isw 以外の引数に格納されている値は変更してはいけません. 3) 3 のとき 先立つ呼び出しで LU 分解された同じ係数行列に対する連立 1 次方程式の右辺ベクトル **b** を変えて再度解くことを指定します. このとき先立つ呼び出しで LU 分解した結果を使います. 実行を制御する引数 isw と右辺ベクトル **b** を格納する引数 zb 以外の引数に格納されている値を変更していけません.
nrowsym	int nrowsym[nz+n]	出力	iordering = 10 で呼び出したとき, 対称化した **SYM** = $\mathbf{A}_1 + \mathbf{A}_1^T$ の非ゼロパターンの下三角行列部分の行指標を列圧縮したものが返却されます. ipledsm = 1 のときは, $\mathbf{A}_1 = \mathbf{D}_r\mathbf{A}\mathbf{P}_c\mathbf{D}_c$ です.
nfcnzsym	int nfcnzsym[n+1]	出力	iordering = 10 で呼び出したとき, 行列 SYM の下三角部分の各列の非零要素の行指標を列方向に圧縮して順次配列 nrowsym に格納するとき, 対応する列の最初の非零要素が格納される位置を表します. nfcnzsym[n] = nsymz + 1 nsymz は, 総要素数を表し

			ます.
zb	dcomplex zb[n]	入力	**Ax** = **b** の右辺定数ベクトル.
		出力	解ベクトル.
nassign	int nassign[n]	出力	各 supernode に対する L, U は圧縮して 2 次元の panel に格納します. この panel を zpanelfactorl および zpanelfactoru の 1 次元部分配列として順番に割り付けたとき何番目になるかを示します. これらの指標ベクトルも同じように npanelindexl, npanelindexu に割り付けます. j = nassign[i-1]のとき, i 番目の supernode を j 番目に割り付けたことを示します. 分解結果の格納方法については図 c_dm_vscs-1 を参照してください.
nsupnum	int	出力	supernode の総数.
		入力	isw ≠ 1 のとき, 初回呼び出しの値を再利用します. (≤ n)
nfcnzfactorl	long nfcnzfactorl[n+1]	出力	複素スパース行列の LU 分解の結果の行列 **L** および **U** はスーパーノードに対応しておのおの求めます. 各 supernode に対応する **L** の列ベクトルは, 共通の行指標ベクトルを持つように圧縮してブロック対角部分に **U** の対応部分を含めて 2 次元の panel に格納します. この panel を zpanelfactorl の 1 次元部分配列として順番に割り付けたとき, *i* 番目の panel の先頭要素 panel[0][0]が 1 次元配列の zpanelfactorl の何番目の要素になるかを示します. 結果の格納方法については図 c_dm_vscs-1 を参照してください.
		入力	isw ≠ 1 のとき, 初回呼び出しの値を再利用します.
zpanelfactorl	dcomplex zpanelfactorl [nsizefactorl]	出力	各 supernode の分解結果に対応する行列 **L** の複数の列ベクトルと行列 **U** のブロック対角部分について, 共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します. panel を順番に割り付けます. i 番目の supernode に対応する panel が何番目の位置に割り当てられるかは j = nassign[i-1]から分かります. その先頭位置は nfcnzfactorl[j-1]に格納されています. panel ごとに分解結果が格納されます. i 番目に割付けられた panel の大きさは ndim[i-1][0] × ndim[i-1][1]の 2 次元配列と見なせます. *i* 番目の panel の panel[t-1][s-1], s ≥ t, s = 1,…, ndim[i-1][0], t=1,…, ndim[i-1][1]に下三角行列 **L** が格納されます. panel の panel[t-1][s-1], s < t, t = 1,…, ndim[i][1]には単位上三角行列 **U** の対角要素を除いたブロック対角部分が格納されます. 結果の格納方法については図 c_dm_vscs-1 を参照してください. (“使用上の注意” c)参照)
nsizefactorl	long	入力	zpanelfactorl の大きさを示す.
		出力	zpanelfactorl の大きさとして必要な大きさが返却されます. (“使用上の注意” c)参照)

nfcnzindexl	long nfcnzindexl[n+1]	出力	各 supernode の分解結果に対応する行列 **L** の複数の列ベクトルと行列 **U** のブロック対角部分について, 共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します. この行指標ベクトルを npanelindexl に順番に割り付けたとき i 番目の行指標ベクトルの先頭要素が 1 次元配列である npanelindexl の何番目の要素になるかを示します. 結果の格納方法については図 c_dm_vscs-1 を参照してください.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します.
npanelindexl	int npanelindexl [nsizeindexl]	出力	各 supernode の分解結果に対応する行列 **L** の複数の列ベクトルと行列 **U** のブロック対角部分について, 共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します. この行指標ベクトルを npanelindexl に順番に割り付けます. *i* 番目の supernode に対応する panel の行の指標ベクトルが何番目に割り当てられるかは j = nassign[i-1]から分かります. その先頭位置は nfcnzindexl[j-1]に格納されています. この行指標は行列 **SYM** に対する post order で並び換えたときの行の番号です. 結果の格納方法については図 c_dm_vscs-1 を参照してください. ("使用上の注意" c)参照)
nsizeindexl	long	入力	npanelindexl の大きさを示す.
		出力	npanelindexl の大きさとして必要な大きさが返却されます. ("使用上の注意" c)参照)
ndim	int ndim[n][3]	出力	ndim[i-1][0]および ndim[i-1][1]は行列 **L** に関して *i* 番目に割り付けられた panel の 1 次元目と 2 次元目の大きさを示します. ndim[i-1][2]は行列 **U** に関して割り付けられた panel を転置したときの 1 次元目の大きさに対角ブロックの大きさを加えた大きさを示します. 結果の格納方法については図 c_dm_vscs-1 を参照してください.
		入力	isw≠1 のとき, 初回呼び出しの値を再利用します.
nfcnzfactoru	long nfcnzfactoru[n+1]	出力	複素スパース行列の LU 分解の結果の行列 **U** に関しては, 各 supernode に対応する **U** の行ベクトルは, ブロック対角部分を除いて共通の列指標ベクトルを持つように圧縮し, 転置したものを 2 次元の panel に格納します. この panel を zpanelfactoru の 1 次元部分配列として順番に割り付けたとき, *i* 番目の panel の先頭要素 panel[0][0]が 1 次元配列の zpanelfactoru の何番目の要素になるかを示します. 結果の格納方法については図 c_dm_vscs-1 を参照してください.
		入力	isw ≠ 1 のとき, 初回呼び出しの値を再利用します.
zpanelfactoru	dcomplex zpanelfactoru [nsizefactoru]	出力	各 supernode の分解結果に対応する行列 **U** のブロック対角部分を除いた複数の行ベクトルについて, 共通の列指標ベクトルを持つように圧縮し, 転置して 2 次元

			の panel に格納します. panel を順番に割り付けます. *i* 番目の supernode に対応する panel が何番目の位置に割り当てられるかは j = nassign[i-1]から分かります.その先頭位置は nfcnzfactoru[j-1]に格納されています. panel ごとに分解結果が格納されます. *i* 番目に割付けられた panel の大きさは{ndim[i-1][2] – ndim[i-1][1]} × ndim[i-1][1]の 2 次元配列と見なせます. *i* 番目の panel の panel[t-1][s-1], s = 1,…, ndim[i-1][2] – ndim[i-1][1], t = 1,…, ndim[i-1][1]に単位上三角行列 **U** のブロック対角部分を除いた部分が行ベクトルを圧縮し, 転置して格納します. 結果の格納方法については図 c_dm_vscs-1 を参照してください. (“使用上の注意” c)参照)
nsizefactoru	long	入力	zpanelfactoru の大きさを示す.
		出力	zpanelfactoru の大きさとして必要な大きさが返却されます. (“使用上の注意” c)参照)
nfcnzindexu	long nfcnzindexu[n+1]	出力	各 supernode の分解結果に対応する行列 **U** は対角ブロック部分を除いて複数の行ベクトルについて, 共通の列指標ベクトルを持つように圧縮し, 転置して 2 次元の panel に格納します. 対角ブロック部分も含んだこの列指標ベクトルを npanelindexu に順番に割り付けたとき *i* 番目の列指標ベクトルの先頭要素が 1 次元配列である npanelindexu の何番目の要素になるかを示します. 結果の格納方法については図 c_dm_vscs-1 を参照してください.
		入力	isw ≠ 1 のとき, 初回呼び出しの値を再利用します.
npanelindexu	int npanelindexu [nsizeindexu]	出力	各 supernode の分解結果に対応する行列 ***U*** の対角ブロックを除いた部分を転置して圧縮して 2 次元の panel に格納します. この列指標ベクトルを対角ブロック部分を含めて npanelindexu に順番に割り付けます. i 番目の supernode に対応する panel の行の指標ベクトルが何番目に割り当てられるかは j = nassign[i-1]から分かります. その先頭位置は nfcnzindexu[j-1]に格納されています. この行指標は行列 **SYM** に対する post order で並び換えたときの列の番号です. 結果の格納方法については図 c_dm_vscs-1 を参照してください. (“使用上の注意” c)参照)
nsizeindexu	long	入力	npanelindexu の大きさを示す.
		出力	npanelindexu の大きさとして必要な大きさが返却されます. (“使用上の注意” c)参照)
nposto	int nposto[n]	出力	post order で i 番目の node が行列 **A** の何番目の列番号に対応するかを表す 1 次元ベクトル.
		入力	isw ≠ 1 のとき, 初回呼び出しの値を再利用します. (“使用上の注意” d)参照)
sclrow	double sclrow[n]	出力	スケーリング対角行列 $\mathbf{D}_r$. 対角要素が 1 次元配列に格納されます.

		入力	isw ≠ 1 のとき, 初回呼び出しの値を再利用します.
sclcol	double sclcol[n]	出力	スケーリング対角行列 $\mathbf{D}_c$. 対角要素が 1 次元配列に格納されます.
		入力	isw ≠ 1 のとき, 初回呼び出しの値を再利用します.
epsz	double	入力	分解過程で対角要素の大きさの絶対値の相対判定値
		出力	epsz ≤ 0.0 のときは標準値が設定されます. ("使用上の注意" b)参照)
thepsz	double	入力	ピボットの判定での閾値(threshold). これより大きな値はピボットとして採用します. 見つかればその時点で, ピボット検索は打ち切ります. 例えば 10^{-2} 程度.
		出力	thepsz ≤ 0.0 のときは 10^{-2} が設定されます. epsz ≥ thepsz > 0.0 のときは, epsz が設定されます.
ipivot	int	入力	スーパーノード内でピボットを行うか, 行う場合どのようなピボットを行うかを指定します. 例えば, 完全ピボットとして 40 を指定. ipivot < 10 : または ipivot ≥ 50: ピボットなし. 10 ≤ ipivot < 20 : 部分ピボット 20 ≤ ipivot < 30 : 対角ピボット 21 : スーパーノード内で対角ピボットがとれないとき, Rook ピボットに変更します. 22 : スーパーノード内で対角ピボットがとれないときは, Rook ピボットに, Rook ピボットが取れないときは完全ピボットに変更します. 30 ≤ ipivot < 40 :Rook ピボット 32 : スーパーノード内で Rook ピボットがとれないとき, 完全ピボットをとります. 40 ≤ ipivot < 50 : 完全ピボット
istatic	int	入力	Pivot を指定したとき, Static Pivot を行うかを示します. 1) = 1 のとき スーパーノード内で指定したピボットが spepsz より大きくないとき, ピボットとして大きさ spepsz の複素数で近似します. ピボットの値が 0.0 なら spepsz に近似します. このとき, 以下を設定しなければなりません. a) epsz は epsz の標準値以下にしなければなりません. b) isclitermax は 10 を設定しスケーリングを行う必要があります. c)thepsz ≥ spepsz でなければなりません. d)解の反復改良を行うため irefine = 1 を指定しなければなりません. 2) ≠ 1 のとき Static Pivot は行いません.
spepsz	double	入力	istatic = 1 のとき, Statc Pivot として使う値. 10^{-10} ≥ spepsz ≥ epsz でなければなりません.
		出力	spepsz < epsz のときは, 10^{-10} が設定されます.
nfcnzpivot	int nfcnzpivot [nsupnum+1]	出力	スーパーノード内の相対的な位置でのピボットの行, 列の入れ換えの履歴を格納する位置を示す. i 番目の

			supernode に関する情報が何番目の位置に割り当てられるかは j = nassign[i-1]から分かります.その先頭位置は nfcnzpivot[j-1]に格納されています. *i* 番目のスーパーノードの入れ換え情報は, npivotp, npivotq の is = nfcnzpivot[j-1],…, ie = nfcnzpivot[j-1] + ndim[j-1][1] - 1 番目の要素に格納されます.
npivotp	int npivotp[n]	出力	スーパーノード内の行の入れ換えに関する情報を格納する.
npivotq	int npivotq[n]	出力	スーパーノード内の列の入れ換えに関する情報を格納する.
irefine	int	入力	LU 分解結果を利用して解を求めるときに, 解の反復改良を行うかどうかを示します. 残差ベクトルを計算するときに 4 倍精度で計算します. = 1 : 解の反復改良を行う. 反復改良して得られる残差ベクトル r_k の絶対値の差分がひとつ前の差分の 1/4 より大きくなるまで, 反復改良を行います. ≠ 1: 解の反復改良を行わない. istatic = 1 のとき, irefine = 1 でなければなりません.
epsr	double	入力	解の残差ベクトル **b - Ax** の絶対値が, **b** の絶対値に対して十分小さいかを判定する判定値. epsr ≤ 0.0 のとき 10^{-6} が設定されます.
itermax	int	入力	(≥ 1) 反復改良を行うときの最大反復回数.
iter	int	出力	反復改良を行った回数.
zw	dcomplex zw[2*nz]	作業域	isw = 1, 2 で続けて呼び出すとき, 呼び出しの間で必要なデータを受け渡すために使われます. 呼び出しの間で値を変更してはいけません.
w	double w[4*nz+6*n]	作業域	isw = 1, 2 で続けて呼び出すとき, 呼び出しの間で必要なデータを受け渡すために使われます. 呼び出しの間で値を変更してはいけません.
iw1	int iw1[2*nz+2*(n+1)+16*n]	作業域	isw = 1, 2 で続けて呼び出すとき, 呼び出しの間で必要なデータを受け渡すために使われます. 呼び出しの間で値を変更してはいけません.
iw2	int iw2[47*n+47+nz+4*(n+1)+2*(nz+n)]	作業域	isw = 1, 2, 3 で続けて呼び出すとき, 呼び出しの間で必要なデータを受け渡すために使われます. 呼び出しの間で値を変更してはいけません.
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になった.	処理を打ち切る.
20100	ipledsm = 1 を指定して, 対角要素に絶対値が大きな要素を並べるため maximum matching を求める処理で, 長さ n の maximum matching がみつからなかった. 行列が特異である可能性がある.	

コード	意　味	処理内容
20200	行および列の均衡化を行う対角行列の対角要素を求める過程で,元の行列 A の行もしくは列にゼロベクトルがあった. 行列が特異である可能性がある.	処理を打ち切る.
20400	LU 分解された行列の対角要素にゼロがあった.	
20500	求めた解ベクトルに対する残差ベクトルのノルムの大きさが方程式 $Ax = b$ の右辺ベクトルのノルムの epsr 倍より大きい. 係数行列は特異に近い可能性がある.	
30000	次のいずれかであった: • $n < 1$ • $nz < 0$ • $nfcnz[n] \neq nz + 1$ • $nsizefactorl < 1$ • $nsizefactoru < 1$ • $nsizeindexl < 1$ • $nsizeindexu < 1$ • $isw < 1$ • $isw > 3$ • $irefine = 1$ のとき $itermax < 1$	
30100	nperm に指定した置換行列が正しくない.	
30200	nrow[j-1]に格納された *i* 列目の行指標 k が $k < 1$ または $k > n$	
30300	*i* 列目の行指標の数 $nfcnz[i] - nfcnz[i-1] > n$	
30500	istatic $= 1$ のとき満たすべき条件をみたしていない. epsz は標準値 $16\boldsymbol{u}$ より大きかった. または $isclitermax < 10$ であった. または $irefine \neq 1$ であった. または spepsz > thepsz であった. または $spepsz > 10^{-8}$ であった.	
31000	zpanelfactorl の大きさ nsizefactorl または npanelindexl の大きさ nsizeindexl または zpanelfactoru の大きさ nsizefactoru または npanelindexu の大きさ nsizeindexu が小さすぎる.	nsizefactorl または nsizeindexl または nsizefactoru または nsizeindexu で指定された 大きさで zpanelfactorl または npanelindexl または zpanelfactoru または npanelindexu を割り付けて $isw = 2$ として再度呼び出す.

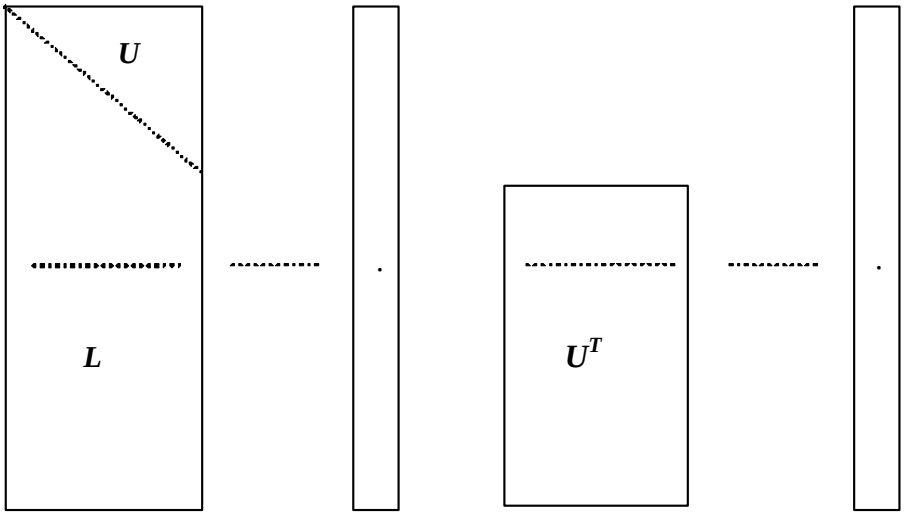


図 c_dm_vscs-1 分解結果の格納概念図

`j = nassign[i-1]` → *i* 番目の supernode は *j* 番目に格納されます.

`p = nfcnzfactorl[j-1]` → *j* 番目の `panel` は `zpanelfactorl` の *p* 番目の要素から `ndim[j-1][0] × ndim[j-1][1]` の長さを占めます.

`q = nfcnzindexl[j-1]` → *j* 番目の `panel` の行指標を表すベクトルは `npanelindexl` の *q* 番目の要素から `ndim[j-1][0]` の長さを占めます.

`panel` は大きさ `ndim[j-1][0] × ndim[j-1][1]` の配列と見なせます.

`panel[t-1][s-1]`, `s ≥ t`, `s = 1,…, ndim[j-1][0]`, `t = 1,…, ndim[j-1][1]` に分解結果の下三角行列 **L** が格納されます.

`panel[t-1][s-1]`, `s < t`, `s = 1,…, ndim[j-1][1]`, `t = 1,…, ndim[j-1][1]` に単位上三角行列 **U** の対角ブロック部分が対角要素を除き格納されます.

`u = nfcnzfactoru[j-1]` → *j* 番目の `panel` は `zpanelfactoru` の *u* 番目の要素から `(ndim[j-1][2] – ndim[j-1][1]) × ndim[j-1][1]` の長さを占めます.

`v = nfcnzindexu[j-1]` → `j` 番目の `panel` の列指標を表すベクトルは `npanelindexu` の *v* 番目の要素から `ndim[j-1][2]` の長さを占めます.

`panel` は大きさ `(ndim[j-1][2] – ndim[j-1][1]) × ndim[j-1][1]` の配列と見なせます.

`panel[y-1][x-1]`, `x = 1,…, ndim[j-1][2] – ndim[j-1][1]`, `y = 1,…, ndim[j-1][1]`

に分解結果の単位上三角行列 **U** の転置行列 $\mathbf{U}^{\mathrm{T}}$ の対角ブロック部分を除いた部分が格納されます.

指標の値は行列 **A** の列番号をもつ node を post order で並び換えた $\mathbf{QAQ}^{\mathrm{T}}$ での列番号を表します.

3. 使用上の注意

a)

直交行列である置換行列 **P** の要素 $\mathbf{p}_{ij} = \mathbf{1}$ のとき, nperm[i-1] = j と表現します.
逆は以下のようにして求めることができます.

```
for (i = 1; i <= n; i++) {
 j = nperm[i-1];
 nperminv[j-1] = i;
}
```

Fill-Reducing Ordering は METIS などを使って求めることができます.
詳細は“付録 参考文献一覧表”の[43], [44]を参照願います.

b)

分解過程で対角要素の大きさの絶対値の相対零判定値にある値を設定したとすると, この値は次の意味を持っています. すなわち, LU 分解の過程で, 対角要素の大きさの絶対値がその値より小さくなった場合に, その値を相対的に零と見なし, icon = 20000 として処理を打ち切ります. epsz の標準値は, 丸め誤差の単位を *u* としたとき, epsz = 16*u* です.
Pivot では, 複素数の大きさを表す絶対値は, 実部の絶対値と虚部の絶対値を加えたもので近似します.
なお, 対角要素の大きさの絶対値が小さくなっても, 処理を続行させたい場合には, epsz に極小の値を与えれば良いのですが, 結果の精度は保証されません.
Static Pivot を指定した場合, 対角要素が spepsz より小さいとき,大きさ spepsz の複素数で近似して LU 分解を行います.
このとき, 解の反復改良を指定しなければなりません.

c)

分解結果を格納する配列 zpanelfactorl, npanelindexl, zpanelfactoru, npanelindexu の必要な大きさは, 事前には分かりません. 十分大きな配列を割り当てるか, 本関数を呼び出して symbolic decomposition を行った解析の結果を使って割り当てを行うことができます.
例えば, 大きさ 1 の 1 次元配列などを割付けます. そしてその大きさ 1 などの小さな値を nsizefactorl, nsizeindexl, nsizefactoru, nsizeindexu に指定して, isw = 1 で呼び出します.
symbolic decomposition を行い, icon = 31000 で終了し, nsizefactorl, nsizeindex, nsizefactoru, nsizeindexu に必要な大きさが返却されます. 必要な大きさの配列を割付け直して引数に指定して, isw = 2 で呼び出すことで, symbolic decomposition 以降の処理を続けることができます. 使用例を参照願います.

d)

列番号に対応するノードを考えます. これを post order で順番を並び換えたときの番号の対応関係が nposto に格納されています. post order の *i* 番目のノードが並び換える前の何番目のノードに対応するかを表します. j = nposto[i-1]は *j* 番目であることを表します.
a)同様にこれは直交行列である置換行列 **Q** を表し,行列 **A** を $\mathbf{QAQ}^{\mathbf{T}}$ と並び換えることに相当します.
逆変換 $\mathbf{Q}^{\mathbf{T}}$ は以下のようにして求めることができます.

```
for (i = 1; i <= n; i++) {
 j = nposto[i-1];
 npostoinv[j-1] = i;
}
```

e)

連立1次方程式**Ax** = **b**は, c_dm_vscluで複素スパース行列**A**をLU分解して, 分解結果を利用して引き続いてc_dm_vscluxを呼び出して解くこともできます.

4. 使用例

連立 1 次方程式 **Ax** = **f** を解きます.
行列は境界条件として立方体の境界で 0 となる楕円型の偏微分方程式に有限差分法を適用して生成されたものを利用します.

$$-\Delta u + a\nabla u + cu = f$$

ここで $a = (a_1, a_2, a_3)$.
行列は関数 init_mat_diag によって生成されます.
対角形式格納法で生成し, 対角ベクトルの下 6 本を圧縮列格納法で格納します. 結果生成された非対称な行列 **A** に関する連立 1 次方程式を解きます.
(並列実行を行うスレッド数は環境変数(OMP_NUM_THREADS)などで指定できます.
例えば, 4 プロセッサーのシステムで 4 つのスレッドで並列実行するときは, OMP_NUM_THREADS を 4 に設定して実行します.)

```
 /* **EXAMPLE** */
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <malloc.h>
#include <omp.h>
#include "cssl.h"

#define  NORD  40
#define  KX  NORD
#define  KY  NORD
#define  KZ  NORD
#define  N  KX * KY * KZ
#define  NBORDER  (N + 1)
#define  NOFFDIAG  6
#define  K  (N + 1)
#define  NDIAG  7
#define  NALL  NDIAG * N

#define  ZWL  2 * NALL
#define  WL  4 * NALL + 6 * N
#define  IW1L  2 * NALL + 2 * (N + 1) + 16 * N
#define  IW2L  47 * N + 47 + 4 * (N + 1) + NALL + 2 * (NALL + N)

void init_mat_diag(double, double, double, double, double*, int*, int, int, int,
                   double, double, double, int, int, int);
double errnrm(dcomplex*, dcomplex*, int);
dcomplex comp_sub(dcomplex, dcomplex);

int MAIN__() {

  int  nofst[NDIAG];
  double  diag[NDIAG][K], diag2[NDIAG][K];
  dcomplex  za[K * NDIAG], zwc[K * NDIAG],
            zw[ZWL], zone;
  int  nrow[K * NDIAG], nfcnz[N + 1],
       nrowsym[K * NDIAG + N], nfcnzsym[N + 1],
```

```
     iwc[K * NDIAG][2];
int  nperm[N],
     nposto[N], ndim[N][3],
     nassign[N],
     mz[N],
     iw1[IW1L], iw2[IW2L];
double  w[WL];
dcomplex  *zpanelfactorl, *zpanelfactoru;
int  *npanelindexl, *npanelindexu;
dcomplex  zdummyfl, zdummyfu;
int  ndummyil,
     ndummyiu;
long  nsizefactorl,
      nsizeindexl,
      nsizeindexu,
      nsizefactoru,
      nfcnzfactorl[N + 1],
      nfcnzfactoru[N + 1],
      nfcnzindexl[N + 1],
      nfcnzindexu[N + 1];
dcomplex  zb[N], zsolex[N];
double  epsz, thepsz, spepsz,
        sclrow[N], sclcol[N];

int  ipivot, istatic, nfcnzpivot[N + 1],
     npivotp[N], npivotq[N],
     irefine, itermax, iter, ipledsm;
double  err, va1, va2, va3, vc, xl, yl, zl, epsr;
int  i, j, nbase, length, numnz, ntopcfg, ncol, nz, icon, iordering,
     isclitermax, isw, nsupnum;

zone.re = 1.0;
zone.im = 0.0;

printf("    LU DECOMPOSITION METHOD\n");
printf("    FOR SPARSE UNSYMMETRIC COMPLEX MATRICES\n");
printf("    IN COMPRESSED COLUMN STORAGE\n\n");

for (i = 0; i < N; i++) {
  zsolex[i] = zone;
}
printf("    EXPECTED SOLUTIONS\n");
printf("    X(1) = (%lf,%lf) X(N) = (%lf,%lf)\n\n",
       zsolex[0].re, zsolex[0].im, zsolex[N - 1].re, zsolex[N - 1].im);

va1 = 1.0;
va2 = 2.0;
va3 = 3.0;
vc = 4.0;
```

```
xl = 1.0;
yl = 1.0;
zl = 1.0;
init_mat_diag(va1, va2, va3, vc, (double *)diag, nofst,
              KX, KY, KZ, xl, yl, zl, NDIAG, N, K);
for (i = 0; i < NDIAG; i++) {
  for (j = 0; j < K; j++) {
    diag2[i][j] = 0;
  }
}

for (i = 0; i < NDIAG; i++) {

  if (nofst[i] < 0) {
    nbase = -nofst[i];
    length = N - nbase;
    for (j = 0; j < length; j++) {
      diag2[i][j] = diag[i][nbase + j];
    }
  } else {
    nbase = nofst[i];
    length = N - nbase;
    for (j = 0; j < length; j++) {
      diag2[i][nbase + j] = diag[i][j];
    }
  }

}

numnz = 1;

for (j = 0; j < N; j++) {
  ntopcfg = 1;
  for (i = NDIAG - 1; i >= 0; i--) {

    if (ntopcfg == 1) {
      nfcnz[j] = numnz;
      ntopcfg = 0;
    }

    if (j + 1 < NBORDER && i + 1 > NOFFDIAG) {
      continue;
    } else {

      if (diag2[i][j] != 0.0) {

        ncol = (j + 1) - nofst[i];
        za[numnz - 1].re = diag2[i][j];
        za[numnz - 1].im = 0.0;
```

```
            nrow[numnz - 1] = ncol;

            numnz++;

          }
        }
      }
    }
    nfcnz[N] = numnz;
    nz = numnz - 1;

    c_dm_vmvsccc(za, nz, nrow, nfcnz, N, zsolex,
                 zb, zwc, (int *)iwc, &icon);

  /* INITIAL CALL WITH IORDER=1 */

    iordering = 0;
    ipledsm = 1;
    isclitermax = 10;
    isw = 1;
    epsz = 1.0e-16;
    nsizefactorl = 1;
    nsizefactoru = 1;
    nsizeindexl = 1;
    nsizeindexu = 1;
    thepsz = 1.0e-2;
    spepsz = 0.0;
    ipivot = 40;
    istatic = 0;
    irefine = 1;
    epsr = 0.0;
    itermax = 10;

    c_dm_vscs(za, nz, nrow, nfcnz, N,
              ipledsm, mz, isclitermax, &iordering,
              nperm, isw,
              nrowsym, nfcnzsym,
              zb,
              nassign,
              &nsupnum,
              nfcnzfactorl, &zdummyfl,
              &nsizefactorl,
              nfcnzindexl,
              &ndummyil, &nsizeindexl,
              (int *)ndim,
              nfcnzfactoru, &zdummyfu,
              &nsizefactoru,
              nfcnzindexu,
              &ndummyiu, &nsizeindexu,
```

```
          nposto,
          sclrow, sclcol,
          &epsz, &thepsz,
          ipivot, istatic, &spepsz, nfcnzpivot,
          npivotp, npivotq,
          irefine, epsr, itermax, &iter,
          zw, w, iw1, iw2, &icon);

printf("ICON=%d NSIZEFACTORL=%d NSIZEFACTORU=%d NSIZEINDEXL=%d",
       icon, nsizefactorl, nsizefactoru, nsizeindexl);
printf(" NSIZEINDEXU=%d NSUPNUM=%d\n", nsizeindexu, nsupnum);

zpanelfactorl = (dcomplex *)malloc(nsizefactorl * sizeof(dcomplex));
zpanelfactoru = (dcomplex *)malloc(nsizefactoru * sizeof(dcomplex));
npanelindexl = (int *)malloc(nsizeindexl * sizeof(int));
npanelindexu = (int *)malloc(nsizeindexu * sizeof(int));

isw = 2;

c_dm_vscs(za, nz, nrow, nfcnz, N,
          ipledsm, mz, isclitermax, &iordering,
          nperm, isw,
          nrowsym, nfcnzsym,
          zb,
          nassign,
          &nsupnum,
          nfcnzfactorl, zpanelfactorl,
          &nsizefactorl,
          nfcnzindexl,
          npanelindexl, &nsizeindexl,
          (int *)ndim,
          nfcnzfactoru, zpanelfactoru,
          &nsizefactoru,
          nfcnzindexu,
          npanelindexu, &nsizeindexu,
          nposto,
          sclrow, sclcol,
          &epsz, &thepsz,
          ipivot, istatic, &spepsz, nfcnzpivot,
          npivotp, npivotq,
          irefine, epsr, itermax, &iter,
          zw, w, iw1, iw2, &icon);

err = errnrm(zsolex, zb, N);

printf("    COMPUTED VALUES\n");
printf("    X(1) = (%lf,%lf) X(N) = (%lf,%lf)\n\n", zb[0], zb[N - 1]);
printf("    ICON = %d\n\n", icon);
printf("    N = %d\n\n", N);
```

```
  printf("    ERROR = %lf\n", err);
  printf("    ITER=%d\n\n\n", iter);
  if (err < 1.0e-8 && icon == 0) {
    printf("********** OK **********\n");
  } else {
    printf("********** NG **********\n");
  }

  free(zpanelfactorl);
  free(zpanelfactoru);
  free(npanelindexl);
  free(npanelindexu);

  return(0);
}

/* ======================================
       INITIALIZE COEFFICIENT MATRIX
   ====================================== */
void init_mat_diag(double va1, double va2, double va3, double vc,
                   double *d_l, int *offset,
                   int nx, int ny, int nz, double xl, double yl, double zl,
                   int ndiag, int len, int ndivp) {
  if (ndiag < 1) {
    printf("FUNCTION INIT_MAT_DIAG:\n");
    printf(" NDIAG SHOULD BE GREATER THAN OR EQUAL TO 1\n");
    return;
  }

#pragma omp parallel default(shared)
{
  int i, j, l, ndiag_loc, nxy, js, k0, j0, i0;
  double hx, hy, hz, hx2, hy2, hz2;

  ndiag_loc = ndiag;
  if (ndiag > 7)
    ndiag_loc = 7;

/* INITIAL SETTING */
  hx = xl / (nx + 1);
  hy = yl / (ny + 1);
  hz = zl / (nz + 1);

#pragma omp for
  for (i = 0; i < ndivp; i++) {
    for (j = 0; j < ndiag; j++) {
      d_l[(j * ndivp) + i] = 0.0;
    }
  }
```

```
  nxy = nx * ny;

/* OFFSET SETTING */
#pragma omp single
  {
    l = 0;
    if (ndiag_loc >= 7) {
      offset[l] = -nxy;
      l++;
    }
    if (ndiag_loc >= 5) {
      offset[l] = -nx;
      l++;
    }
    if (ndiag_loc >= 3) {
      offset[l] = -1;
      l++;
    }
    offset[l] = 0;
    l++;
    if (ndiag_loc >= 2) {
      offset[l] = 1;
      l++;
    }
    if (ndiag_loc >= 4) {
      offset[l] = nx;
      l++;
    }
    if (ndiag_loc >= 6) {
      offset[l] = nxy;
    }
  }

/* MAIN LOOP */
#pragma omp for
  for (j = 0; j < len; j++) {
    js = j + 1;

/* DECOMPOSE JS-1 = (K0-1)*NX*NY+(J0-1)*NX+I0-1 */
    k0 = (js -1) / nxy + 1;
    if (k0 > nz) {
      printf("ERROR; K0.GH.NZ \n");
      goto label_100;
    }
    j0 = (js - 1 - nxy * (k0 - 1)) / nx + 1;
    i0 = js - nxy * (k0 - 1) - nx * (j0 - 1);
    l = 0;
```

```
    if (ndiag_loc >= 7) {
      if (k0 > 1) d_l[(l * ndivp) + j] = -(1.0 / hz + 0.5 * va3) / hz;
      l++;
    }
    if (ndiag_loc >= 5) {
      if (j0 > 1) d_l[(l * ndivp) + j] = -(1.0 / hy + 0.5 * va2) / hy;
      l++;
    }
    if (ndiag_loc >= 3) {
      if (i0 > 1) d_l[(l * ndivp) + j] = -(1.0 / hx + 0.5 * va1) / hx;
      l++;
    }
    hx2 = hx * hx;
    hy2 = hy * hy;
    hz2 = hz * hz;
    d_l[(l * ndivp) + j] = 2.0 / hx2 + vc;
    if (ndiag_loc >= 5) {
      d_l[(l * ndivp) + j] += 2.0 / hy2;
      if (ndiag_loc >= 7) {
        d_l[(l * ndivp) + j] += 2.0 / hz2;
      }
    }
    l++;
    if (ndiag_loc >= 2) {
      if (i0 < nx) d_l[(l * ndivp) + j] = -(1.0 / hx - 0.5 * va1) / hx;
      l++;
    }
    if (ndiag_loc >= 4) {
      if (j0 < ny) d_l[(l * ndivp) + j] = -(1.0 / hy - 0.5 * va2) / hy;
      l++;
    }
    if (ndiag_loc >= 6) {
      if (k0 < nz) d_l[(l * ndivp) + j] = -(1.0 / hz - 0.5 * va3) / hz;
    }
label_100: ;
  }

}

  return;
}

/* ======================================
  * SOLUTE ERROR
  * | Z1 - Z2 |
   ====================================== */
double errnrm(dcomplex *z1, dcomplex *z2, int len) {
  double  rtc, s;
  dcomplex  ss;
```

```
  int i;

  s = 0.0;
  for (i = 0; i < len; i++) {
    ss = comp_sub(z1[i], z2[i]);
    s += ss.re * ss.re + ss.im * ss.im;
  }

  rtc = sqrt(s);
  return(rtc);
}

dcomplex comp_sub(dcomplex so1, dcomplex so2) {

  dcomplex obj;

  obj.re = so1.re - so2.re;
  obj.im = so1.im - so2.im;
  return obj;
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VSCS の項目及び[2], [13], [17], [19], [22], [23], [48], [57], [63], [68], [69]を参照してください.

c_dm_vsevph

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)
ierr = c_dm_vsevph(a, k, n, nf, nl, ivec, &etol, &ctol, nev, e, maxne, m, ev, &icon);

1. 機能

n 次の実対称行列 **A** の指定されたいくつかの固有値を求めます. 指定に応じて, 固有ベクトルを求めます. 実対称行列 **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ます. 固有ベクトルは逆反復法で求めます.

$$\mathbf{Ax} = \lambda\mathbf{x}$$

A は $n \times n$ の実対称行列.

2. 引数

呼出し形式:

```
ierr = c_dm_vsevph((double*)a, k, n, nf, nl, ivec, &etol, &ctol, nev, e,
            maxne, (int*)m, (double*)ev, &icon);
```

引数の説明:

a	double a[n][k]	入力	a の上三角部分{a[i−1][j−1], i ≤ j}に固有値・固有ベクトルを求める実対称行列 **A** の上三角部分$\{a_{ij} \mid i \le j\}$を格納します. 演算後の内容は保証されません.
k	int	入力	配列 a の整合寸法 (k ≥ n).
n	int	入力	実対称行列 **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ます. ivec ≠ 1 のとき固有値のみ求めます.
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.
		出力	etol < 3×10^{-16} のときは, etol = 3×10^{-16} が標準値として設定されます. ("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(≥ etol). 近似的多重固有値(cluster)に属する固有値の対応する固有ベクトルの 1 次独立性を保証するために使用されます. 5.0×10^{-12} が一般的に適当です. ただし非常に大きなクラスタに対しては, 大きな値が必要です. $10^{-6} \ge$ ctol ≥ etol.
		出力	ctol > 10^{-6} のときは, ctol = 10^{-6} と設定されます. ctol < etol のときは, ctol = $10 \times$ etol が標準値として設定されます. ("使用上の注意"a)参照)

nev	int nev[5]	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. nev[3]は, 求めた固有値の内最初の固有値の番号. nev[4]は, 求めた固有値の内最後の固有値の番号.
e	double e[maxne]	出力	固有値が格納されます. 求められた固有値は e[i–1], i = 1, ... , nev[2]に格納されます.
maxne	int	入力	計算できる固有値の最大個数. 配列 e の大きさ. 多重度 m の固有値がいくつかあると考えられるとき, n を上限として $\mathtt{nl}-\mathtt{nf}+1+2\times m$ を設定する必要があります. nev[2] > maxne のとき, 固有ベクトルの計算はできません. ("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します. ("使用上の注意" b)参照)
ev	double ev[maxne][k]	出力	ivec = 1 のとき, 固有値に対応して固有ベクトルが格納されます. 求められた固有ベクトルは ev[i–1][j–1], i = 1,...,nev[2], j = 1, ... ,n に格納されます.
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.
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.

3. 使用上の注意

a) etol 及び ctol

本ルーチンは, 求める固有値を交わりのない順序付けられた集合に分けて独立に計算することにより並列化しています.

ε = etol としたとき, 連続する固有値λ_j, $j = s, s+1, \ldots, s+k$, $(k \geq 0)$に対して,

$$\frac{|\lambda_i - \lambda_{i-1}|}{1+\max(|\lambda_{i-1}|,|\lambda_i|)} \leq \varepsilon, \tag{1}$$

$i = s, s+1, \ldots, s+k$ 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, $j = s-1, s, \ldots, s+k$ は数値的に多重であると見なされます.

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ます.

(1)が ε =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ます.

ε =etol で異なる固有値と見なされた連続する固有値 λ_m, $m = t-1, t, \ldots, t+k$, $(k \geq 0)$ に対して, ε =ctol としたとき, $i = t, t+1, \ldots, t+k$ 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, $m = t-1, t, \ldots, t+k$ を近似的に多重(cluster)と見なします. 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ます.

b) maxne

maxne に, 計算できる固有値の最大個数を指定できます. ctol を大きくすると, cluster の大きさが大きくなり, 計算する固有値の総数が maxne を超えるかもしれません. この場合, ctol を小さくするか, maxne を大きくする必要があります.

計算する固有値の総数が maxne を超えた場合, icon＝20000 を返却します. 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ません. 固有値は求められていますが, 多重度分だけ繰り返して格納はされていません.

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ます.

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 は求める固有値の総数を nt (nt = nl – nf + 1)として nt で十分です. 多重な固有値がいくつかあり, 多重度が m と考えられるときは, 少なくとも maxne は $nt + 2 \times m$ とする必要があります.

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値が nev[2]に返却されます. この値で領域を確保して再度呼び出すことで計算を続けることができます.

4. 使用例

実対称行列の固有値を計算します.

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

#define N              500
#define K                N
#define NF               1
#define NL             100
#define MAXNE      NL-NF+1

MAIN__()
{
  double a[N][K], ab[N][K];
  double e[MAXNE], ev[MAXNE][K];
  double vv[N][K];
  double etol, ctol, pi;
  int    nev[5], m[2][MAXNE];
  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_dm_vmggm ((double*)a, k, (double*)vv, k, (double*)ab, k, n, n, n, &icon);
  ierr = c_dm_vmggm ((double*)vv, k, (double*)ab, k, (double*)a, k, n, n, n, &icon);

  /* Calculate the eigendecomposition of A */
  ierr = c_dm_vsevph ((double*)a, k, n, nf, nl, ivec, &etol, &ctol, nev, e, maxne,
                      (int*)m, (double*)ev, &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 スレッド並列機能使用手引書”の DM_VSEVPH の項目及び[30], [61]を参照してください.

c_dm_vsldl

正値対称行列の LDL^T 分解 (ブロック化された変形コレスキー分解法)
`ierr = c_dm_vsldl(a, k, n, epsz, &icon);`

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 分解します.

$$\mathbf{A} = \mathbf{LDL}^T$$

ただし, **L** は単位下三角行列, **D** は対角行列です. ($n \geq 1$)

2. 引数

呼出し形式:

```
ierr = c_dm_vsldl((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 ≤ j に格納します. 格納方法の詳細については, 図 c_dm_vsldl-1.を参照してください.
		出力	LDL^T 分解された行列 **L**, $\mathbf{D}^{-1}$ および $\mathbf{L}^T$ が, a の上三角部分 a[i−1][j−1], i ≤ j に格納されます.
k	int	入力	配列 a の整合寸法 (≥ 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). epsz = 0 のときは標準値が採用されます. ("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. 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 • k < n • epsz < 0	

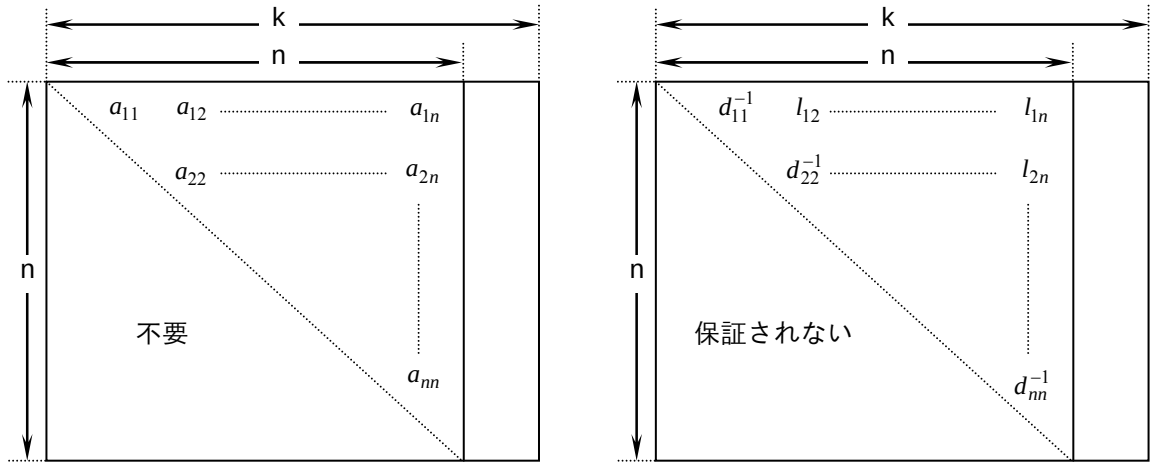


図 c_dm_vsldl-1. コレスキー分解法のデータの格納方法

LDL^{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, ... , n, j=i, ... , n に格納します. LDL^{T}分解された結果は, 行列 $\mathbf{D}^{-1}$ を対角部分に, **L** の対角要素を除いた部分を上三角部分にそれぞれ格納されます. 下三角部分の値は保証されません.

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^{-s} を設定したとすると, この値は次の意味を持っています. 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みなし, icon = 20000 として処理を打ち切ります. epsz の標準値は丸め誤差の単位をμとしたとき, epsz = 16μです.

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いいのですが, 結果は保証されません.

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ります. このとき, icon = 10000 としますが, 処理は続行します. ただし, ピボッティ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す.

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, その逆数をとって下さい.

4. 使用例

1000 × 1000 の係数行列の連立 1 次方程式を解きます.

```
#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 NMAX      (1000)
#define LDA       (NMAX+1)
```

```
MAIN__()
{
  int    n, i, j, icon, ierr;
  double a[NMAX][LDA], b[NMAX];
  double epsz, s, det;

  n    = NMAX;
  epsz = 0.0;

#pragma omp parallel for shared(a,n) private(i,j)
  for(i=0; i<n; i++)
    for(j=0; j<n; j++) a[i][j] = min(i,j)+1;

#pragma omp parallel for shared(b,n) private(i)
  for(i=0; i<n; i++) b[i] = (i+1)*(i+2)/2+(i+1)*(n-i-1);

  ierr = c_dm_vsldl((double*)a, LDA, n, epsz, &icon);

  if (icon != 0) {
    printf("ERROR: c_dm_vsldl failed with icon = %d\n", icon);
    exit(1);
  }

  ierr = c_dm_vldlx(b, (double*)a, LDA, n, &icon);

  if (icon != 0) {
    printf("ERROR: c_dm_vldlx failed with icon = %d\n", icon);
    exit(1);
  }

  s = 1.0;
#pragma omp parallel for shared(a,n) private(i) reduction(*:s)
  for(i=0; i<n; i++) s *= a[i][i];

  printf("solution vector:\n");
  for(i=0; i<10; i++) printf("    b[%d] = %e\n", i, b[i]);

  det = 1.0/s;
  printf("\ndeterminant of the matrix = %e\n", det);
  return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VSLDL の項目及び[30], [54], [70]を参照してください.

c_dm_vsrlu

実スパース行列の LU 分解
ierr = c_dm_vsrlu(a, nz, nrow, nfcnz, n, ipledsm, mz, isclitermax, &iordering, nperm, isw, nrowsym, nfcnzsym, nassign, &nsupnum, nfcnzfactorl, panelfactorl, &nsizefactorl, nfcnzindexl, npanelindexl, &nsizeindexl, ndim, nfcnzfactoru, panelfactoru, &nsizefactoru, nfcnzindexu, npanelindexu, &nsizeindexu, nposto, sclrow, sclcol, &epsz, &thepsz, ipivot, istatic, &spepsz, nfcnzpivot, npivotp, npivotq, w, iw1, iw2, &icon);

1. 機能

$n \times n$ の実スパース行列 **A** に対角要素に大きな要素を並べる並び換えを行い. 行および列の均衡化を行うスケーリングを行います. スーパーノードの対角ブロック内で指定した Pivot をとり, LU 分解します.

実スパース行列 **A** は以下のように変換できます.

$$\mathbf{A_1} = \mathbf{D_r A P_c D_c}$$

ここで $\mathbf{P_c}$ は列の入れ換えを行う直交行列, $\mathbf{D_r}$ は行のスケーリングを行う対角行列, $\mathbf{D_c}$ は列のスケーリングを行う対角行列です.

$$\mathbf{A_2} = \mathbf{Q P A_1 P^T Q^T}$$

$\mathbf{A_2}$ は, スーパーノードの対角ブロックで指定された Pivot を行い, 行および列の入れ換えを行い LU 分解されます.

ただし, **P** は, $\mathbf{SYM} = \mathbf{A_1} + \mathbf{A_1^T}$ の非ゼロパターンに対して求めた ordering による行列要素の並び換えを示す置換行列で, **Q** は **SYM** に対する post order による行列要素の並び換えを示す置換行列を示します. **P**, **Q** は直交行列です.

L は下三角行列, **U** は単位上三角行列です.

Pivot 処理で, 閾値 thepsz より絶対値が大きな Pivot を見つけることが出来なかったとき, 対角ブロック内の絶対値が最大の要素を Pivot の候補とします. この値が小さ過ぎるときに Static Pivot を与えて近似的に LU 分解を続けることができます.

2. 引数

呼出し形式:

```
ierr = c_dm_vsrlu(a, nz, nrow, nfcnz, n, ipledsm, mz, isclitermax,
            &iordering, nperm, isw, nrowsym, nfcnzsym, nassign, &nsupnum,
```

```
nfcnzfactorl, panelfactorl, &nsizefactorl, nfcnzindexl,
npanelindexl, &nsizeindexl, (int *)ndim, nfcnzfactoru,
panelfactoru, &nsizefactoru, nfcnzindexu, npanelindexu,
&nsizeindexu, nposto, sclrow, sclcol, &epsz, &thepsz, ipivot,
istatic, &spepsz, nfcnzpivot, npivotp, npivotq, w, iw1, iw2,
&icon);
```

引数の説明:

a	double a[nz]	入力	実スパースな係数行列 **A** を圧縮列格納法で格納します. 圧縮列格納法については, 実スパース行列と実ベクトル(圧縮列格納法), c_dm_vmvscc の図 c_dm_vmvscc-1 を参照してください. ("使用上の注意" e)参照)
nz	int	入力	実スパースな係数行列 **A** の非零要素の総数.
nrow	int nrow[nz]	入力	圧縮列格納法で使用される行指標で a に格納される要素が何番目の行ベクトルに属するかを示します.
nfcnz	int nfcnz[n+1]	入力	行列 **A** の各列の非零要素を列方向に圧縮して順次配列 a に格納するとき,対応する列の最初の非零要素が格納される位置を表します. nfcnz[n] = nz + 1.
n	int	入力	行列 **A** の次数 n.
ipledsm	int	入力	対角要素に大きな要素を並べる列の入れ換えを行うかを指定します. 1 のとき, 列の入れ換えを求めて入れ換えます. 1 以外のとき, 列の入れ換えを行いません.
mz	int mz[n]	出力	ipledsm に 1 が指定されたとき, 列の入れ換えを表す. mz[i-1] = j は行列 $\mathbf{a}_{ij}$ の属する *j* 番目の列を *i* 番目の列に移動する. $\mathbf{a}_{ij}$ は対角要素に並べる大きな要素を表す.
isclitermax	int	入力	係数行列のスケーリングを行う対角行列 $\mathbf{D}_r$ と $\mathbf{D}_c$ を反復して求める反復回数. isclitermax ≤ 0 のときスケーリングは行わずに, $\mathbf{D}_r$ および $\mathbf{D}_c$ は単位行列が設定されます. isclitermax ≥ 10 のときは 10 回を上限値とします.
iordering	int	入力	ordering を表す置換行列 **P** で $\mathbf{PA_1P^T}$ と変換した行列を LU 分解するかを指定します. 置換行列は直交行列です. 10 のとき, isw = 1 で呼び出すと ordering を求める $\mathbf{A_1}$ に関する情報を求めて nrowsym, nfcnzsym に設定します. 11 のとき, isw = 1 で 10 を指定して呼び出して得た行列を対称化した情報 nrowsym, nfcnzsym を使って決めた ordering を nperm に指定して同じく isw = 1 で呼び出し, 計算を続けることを示します. $\mathbf{PA_1P^T}$ を LU 分解する処理を続けます. 10,11 以外のとき, ordering は指定せずに行列 $\mathbf{A_1}$ をそのまま LU 分解します.
		出力	isw = 1 と iordering = 10 を指定して呼び出した後, iordering に 11 が設定されます. そのため, ordering を nperm に指定して再び呼び出すときに特に 11 を指

			定する必要はありません. (“使用上の注意” a)参照)
nperm	int nperm[n]	入力	iordering = 11 のとき使用する置換行列をベクトルで指定します. (“使用上の注意” a)参照)
isw	int	入力	呼び出しに関する制御情報を示します. 3) 1 のとき 対称化および symbolic decomposition を行い, 必要な領域が割り当てられているか調べ計算を行います. iordering = 10 で呼び出し, ordering を求めるための情報が nrowsym, nfcnzsym に出力されます. これらを使って **SYM** に対する ordering を求めた後, nperm に指定して iordering = 11 を指定してもう 1 回 isw = 1 で呼び出します. iordering が 10,11 以外のときは ordering は行ないません. 4) 2 のとき isw = 1 で呼び出したとき, panelfactorl, panelfactoru, npanelindexl または npanelindexu の大きさが不足して icon = 31000 で終了した処理を継続します. このとき, nsizefactorl, nsizefactoru, nsizeindexl または nsizeindezu に返却された必要な大きさで panelfactorl, panelfactoru, npanelindexl または npanelindexu を確保し直して指定しなおして再度呼び出します. これらの引数と実行を制御する引数 isw 以外の引数に格納されている値は変更してはいけません.
nrowsym	int nrowsym[nz+n]	出力	iordering = 10 で呼び出したとき, 対称化した **SYM** = $\mathbf{A_1} + \mathbf{A_1}^T$ の非ゼロパターンの下三角行列部分の行指標を列圧縮したものが返却されます. ipledsm = 1 のときは, $\mathbf{A_1} = \mathbf{D_r A P_c D_c}$ です.
nfcnzsym	int nfcnzsym[n+1]	出力	iordering = 10 で呼び出したとき, 行列 SYM の下三角部分の各列の非零要素の行指標を列方向に圧縮して順次配列 nrowsym に格納するとき,対応する列の最初の非零要素が格納される位置を表します. nfcnzsym[n] = nsymz + 1 nsymz は, 総要素数を表します.
nassign	int nassign[n]	出力	各 supernode に対する L, U は圧縮して 2 次元の panel に格納します. この panel を panelfactorl および panelfactoru の 1 次元部分配列として順番に割り付けたとき何番目になるかを示します. これらの指標ベクトルも同じように npanelindexl, npanelindexu に割り付けます. j = nassign[i-1]のとき, *i* 番目の supernode を *j* 番目に割り付けたことを示します. 分解結果の格納方法については図 c_dm_vsrlu-1 を参照してください.
		入力	isw ≠ 1 のとき, 初回呼び出しの値を再利用します.
nsupnum	int	出力	supernode の総数.

		入力	isw ≠ 1 のとき, 初回呼び出しの値を再利用します. (≤n)
nfcnzfactorl	long nfcnzfactorl[n+1]	出力	実スパース行列の LU 分解の結果の行列 **L** および **U** はスーパーノードに対応しておのおの求めます. 各 supernode に対応する **L** の列ベクトルは, 共通の行指標ベクトルを持つように圧縮してブロック対角部分に **U** の対応部分を含めて 2 次元の panel に格納します. この panel を panelfactorl の 1 次元部分配列として順番に割り付けたとき, *i* 番目の panel の先頭要素 panel[0][0]が 1 次元配列の panelfactorl の何番目の要素になるかを示します. 結果の格納方法については図 c_dm_vsrlu-1 を参照してください.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します.
panelfactorl	double panelfactorl [nsizefactorl]	出力	各 supernode の分解結果に対応する行列 **L** の複数の列ベクトルと行列 **U** のブロック対角部分について, 共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します. panel を順番に割り付けます. *i* 番目の supernode に対応する panel が何番目の位置に割り当てられるかは j = nassign[i-1]から分かります. その先頭位置は nfcnzfactorl[j-1]に格納されています. panel ごとに分解結果が格納されます. *i* 番目に割付けられた panel の大きさは ndim[i-1][0] × ndim[i-1][1]の 2 次元配列と見なせます. i 番目の panel の panel[t-1][s-1], s ≥ t, s = 1,…, ndim[i-1][0], t = 1,…, ndim[i-1][1]に下三角行列 **L** が格納されます. panel の panel[t-1][s-1], s < t, t = 1,…, ndim[i-1][1]には単位上三角行列 **U** の対角要素を除いたブロック対角部分が格納されます. 結果の格納方法については図 c_dm_vsrlu-1 を参照してください. ("使用上の注意" c)参照)
nsizefactorl	long	入力	panelfactorl の大きさを示す.
		出力	panelfactorl の大きさとして必要な大きさが返却されます. ("使用上の注意" c)参照)
nfcnzindexl	long nfcnzindexl[n+1]	出力	各 supernode の分解結果に対応する行列 **L** の複数の列ベクトルと行列 **U** のブロック対角部分について, 共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します. この行指標ベクトルを npanelindexl に順番に割り付けたとき i 番目の行指標ベクトルの先頭要素が 1 次元配列である npanelindexl の何番目の要素になるかを示します. 結果の格納方法については図 c_dm_vsrlu-1 を参照してください.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します.
npanelindexl	int npanelindexl [nsizeindexl]	出力	各 supernode の分解結果に対応する行列 **L** の複数の列ベクトルと行列 **U** のブロック対角部分について, 共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します. この行指標ベクトルを npanelindexl に順番に割り付けます. i 番目の supernode に対応する panel の行の指標ベクトルが何番目に割り当てられる

			かは j = nassign[i-1]から分かります. その先頭位置は nfcnzindexl[j-1]に格納されています. この行指標は行列 **SYM** に対する post order で並び換えたときの行の番号です. 結果の格納方法については図 c_dm_vsrlu-1 を参照してください. ("使用上の注意" c)参照)
nsizeindexl	long	入力	npanelindexl の大きさを示す.
		出力	npanelindexl の大きさとして必要な大きさが返却されます. ("使用上の注意" c)参照)
ndim	int ndim[n][3]	出力	ndim[i-1][0]および ndim[i-1][1]は行列 **L** に関して *i* 番目に割り付けられた panel の 1 次元目と 2 次元目の大きさを示します. ndim[i-1][2]は行列 **U** に関して割り付けられた panel を転置したときの 1 次元目の大きさに対角ブロックの大きさを加えた大きさを示します. 結果の格納方法については図 c_dm_vsrlu-1 を参照してください.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します.
nfcnzfactoru	long nfcnzfactoru[n+1]	出力	実スパース行列の LU 分解の結果の行列 **U** に関しては, 各 supernode に対応する **U** の行ベクトルは, ブロック対角部分を除いて共通の列指標ベクトルを持つように圧縮し, 転置したものを 2 次元の panel に格納します. この panel を panelfactoru の 1 次元部分配列として順番に割り付けたとき, *i* 番目の panel の先頭要素 panel[0][0]が 1 次元配列の panelfactoru の何番目の要素になるかを示します. 結果の格納方法については図 c_dm_vsrlu-1 を参照してください.
		入力	isw ≠ 1 のとき, 初回呼び出しの値を再利用します.
panelfactoru	double panelfactoru [nsizefactoru]	出力	各 supernode の分解結果に対応する行列 **U** のブロック対角部分を除いた複数の行ベクトルについて, 共通の列指標ベクトルを持つように圧縮し, 転置して 2 次元の panel に格納します. panel を順番に割り付けます. *i* 番目の supernode に対応する panel が何番目の位置に割り当てられるかは j = nassign[i-1]から分かります.その先頭位置は nfcnzfactoru[j-1]に格納されています. panel ごとに分解結果が格納されます. *i* 番目に割付けられた panel の大きさは{ndim[i-1][2] – ndim[i-1][1] } × ndim[i-1][1]の 2 次元配列と見なせます. *i* 番目の panel の panel[t-1][s-1], s = 1,…, ndim[i-1][2] – ndim[i-1][1], t = 1,…, ndim[i-1][1]に単位上三角行列 **U** のブロック対角部分を除いた部分が行ベクトルを圧縮し, 転置して格納します. 結果の格納方法については図 c_dm_vsrlu-1 を参照してください. ("使用上の注意" c)参照)
nsizefactoru	long	入力	panelfactoru の大きさを示す.
		出力	panelfactoru の大きさとして必要な大きさが返却されます. ("使用上の注意" c)参照)

nfcnzindexu	long nfcnzindexu[n+1]	出力	各 supernode の分解結果に対応する行列 **U** は対角ブロック部分を除いて複数の行ベクトルについて，共通の列指標ベクトルを持つように圧縮し, 転置して 2 次元の panel に格納します. 対角ブロック部分も含んだこの列指標ベクトルを npanelindexu に順番に割り付けたとき *i* 番目の列指標ベクトルの先頭要素が 1 次元配列である npanelindexu の何番目の要素になるかを示します. 結果の格納方法については図 c_dm_vsrlu-1 を参照してください.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します.
npanelindexu	int npanelindexu [nsizeindexu]	出力	各 supernode の分解結果に対応する行列 **U** の対角ブロックを除いた部分を転置して圧縮して 2 次元の panel に格納します. この列指標ベクトルを対角ブロック部分を含めて npanelindexu に順番に割り付けます. *i* 番目の supernode に対応する panel の行の指標ベクトルが何番目に割り当てられるかは j = nassign[i-1]から分かります. その先頭位置は nfcnzindexu[j-1]に格納されています. この行指標は行列 **SYM** に対する post order で並び換えたときの列の番号です. 結果の格納方法については図 c_dm_vsrlu-1 を参照してください. (“使用上の注意” c)参照)
nsizeindexu	long	入力	npanelindexu の大きさを示す.
		出力	npanelindexu の大きさとして必要な大きさが返却されます. (“使用上の注意” c)参照)
nposto	int nposto[n]	出力	post order で *i* 番目の node が行列 **A** の何番目の列番号に対応するかを表す 1 次元ベクトル.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します. (“使用上の注意” d)参照)
sclrow	double sclrow[n]	出力	スケーリング対角行列 $\mathbf{D}_r$. 対角要素が 1 次元配列に格納されます.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します.
sclcol	double sclcol[n]	出力	スケーリング対角行列 $\mathbf{D}_c$. 対角要素が 1 次元配列に格納されます.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します.
epsz	double	入力	分解過程で対角要素の大きさの絶対値の相対判定値
		出力	epsz ≤ 0.0 のときは標準値が設定されます. (“使用上の注意” b)参照)
thepsz	double	入力	ピボットの判定での閾値(threshold). これより大きな値はピボットとして採用します. 見つかればその時点で, ピボット検索は打ち切ります. 例えば 10^{-2} 程度.
		出力	thepsz ≤ 0.0 のときは 10^{-2} が設定されます. epsz ≥ thepsz > 0.0 のときは, epsz が設定されます.
ipivot	int	入力	スーパーノード内でピボットを行うか, 行う場合どのようなピボットを行うかを指定します. 例えば, 完全ピボットとして 40 を指定. ipivot < 10 : または ipivot ≥ 50: ピボットなし.

			10 ≤ ipivot < 20 : 部分ピボット 20 ≤ ipivot < 30 : 対角ピボット 21 : スーパーノード内で対角ピボットがとれないとき, Rook ピボットに変更します. 22 : スーパーノード内で対角ピボットがとれないときは, Rook ピボットに, Rook ピボットが取れないときは完全ピボットに変更します. 30 ≤ ipivot < 40 :Rook ピボット 32 : スーパーノード内で Rook ピボットがとれないとき, 完全ピボットをとります. 40 ≤ ipivot < 50 : 完全ピボット
istatic	int	入力	Pivot を指定したとき, Static Pivot を行うかを示します. 1) = 1 のとき スーパーノード内で指定したピボットが spepsz より大きくないとき, ピボットとして copysign(spepsz, Pivot)で近似します. ピボットの値が 0.0 なら spepsz に近似します. このとき, 以下を設定しなければなりません. a) epsz は epsz の標準値以下にしなければなりません. b) isclitermax は 10 を設定しスケーリングを行う必要があります. c)thepsz ≥ spepsz でなければなりません. 2) ≠ 1 のとき Static Pivot は行いません.
spepsz	double	入力	istatic = 1 のとき, Statc Pivot として使う値. thepsz ≥ spepsz ≥ epsz でなければなりません.
		出力	spepsz < epsz のときは, 10^{-10} が設定されます.
nfcnzpivot	int nfcnzpivot [nsupnum+1]	出力	スーパーノード内の相対的な位置でのピボットの行, 列の入れ換えの履歴を格納する位置を示す. *i* 番目の supernode に関する情報が何番目の位置に割り当てられるかは j = nassign[i-1]から分かります.その先頭位置は nfcnzpivot[j-1]に格納されています. *i* 番目のスーパーノードの入れ換え情報は, npivotp, npivotq の is = nfcnzpivot[j-1],…, ie = nfcnzpivot[j-1] + ndim[j-1][2] - 1 番目の要素に格納されます.
npivotp	int npivotp[n]	出力	スーパーノード内の行の入れ換えに関する情報を格納する.
npivotq	int npivotq[n]	出力	スーパーノード内の列の入れ換えに関する情報を格納する.
w	double w[4*nz+6*n]	作業域	isw = 1, 2 で続けて呼び出すとき,呼び出しの間で必要なデータを受け渡すために使われます. 呼び出しの間で値を変更してはいけません.
iw1	int iw1[2*nz+2*(n+1)+16*n]	作業域	isw = 1, 2 で続けて呼び出すとき, 呼び出しの間で必要なデータを受け渡すために使われます. 呼び出しの間で値を変更してはいけません.
iw2	int iw2[47*n+47+nz+4*(n+1)+2*(nz+n)]	作業域	isw = 1, 2 で続けて呼び出すとき,呼び出しの間で必要なデータを受け渡すために使われます. 呼び出しの間で値を変更してはいけません.

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	istatic = 1 のとき小さすぎるピボットを spepsz で置き換える Static Pivot を行った.	処理は続行する.
20000	ピボットが相対的に零になった.	処理を打ち切る.
20100	ipledsm = 1 を指定して,対角要素に絶対値が大きな要素を並べるため maximum matching を求める処理で, 長さ n の maximum matching がみつからなかった. 行列が特異である可能性がある.	
20200	行および列の均衡化を行う対角行列の対角要素を求める過程で,元の行列 A の行もしくは列にゼロベクトルがあった. 行列が特異である可能性がある.	
30000	次のいずれかであった: • n < 1 • nz < 0 • nfcnz[n] ≠ nz + 1 • nsizefactorl < 1 • nsizefactoru < 1 • nsizeindexl < 1 • nsizeindexu < 1 • isw < 1 • isw > 2	
30100	nperm に指定した置換行列が正しくない.	
30200	nrow[j-1]に格納された *i* 列目の行指標 k が k<1 または k>n	
30300	*i* 列目の行指標の数 nfcnz[i] − nfcnz[i-1] > n	
30500	istatic = 1 のとき満たすべき条件をみたしていない. epsz は標準値 16***u*** より大きかった. または isclitermax < 10 であった. または spepsz > thepsz であった.	

コード	意　味	処 理 内 容
31000	panelfactorl の大きさ nsizefactorl または npanelindexl の大きさ nsizeindexl または panelfactoru の大きさ nsizefactoru または npanelindexu の大きさ nsizeindexu が小さすぎます.	nsizefactorl または nsizeindexl または nsizefactoru または nsizeindexu で指定された 大きさで panelfactorl または npanelindexl または panelfactoru または npanelindexu を割り付けて isw＝2 として再度呼び出す.

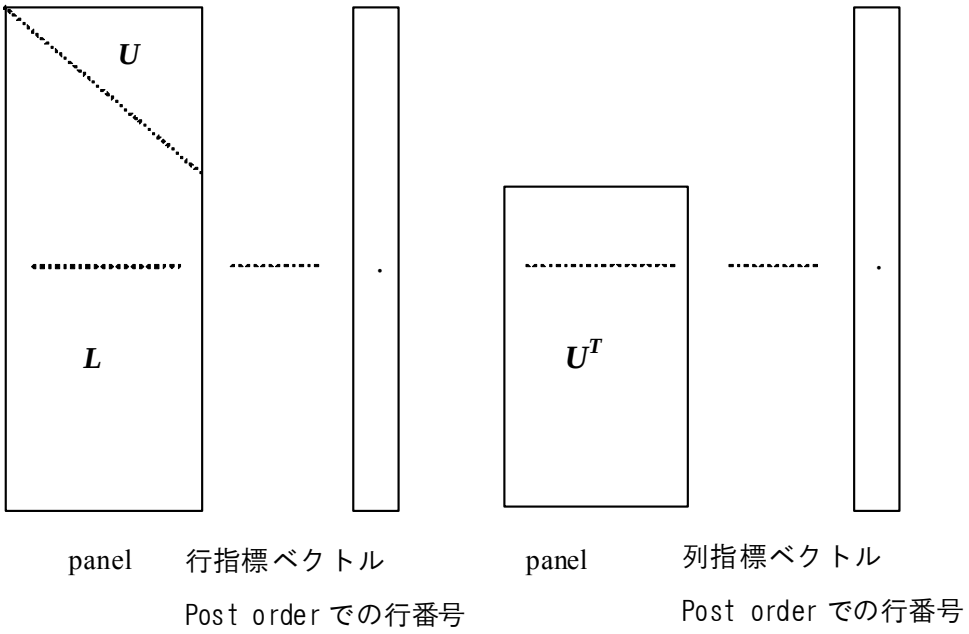


図 c_dm_vsrlu-1 分解結果の格納概念図

j＝nassign[i-1]　→ *i* 番目の supernode は *j* 番目に格納されます.

p = nfcnzfactorl[j-1] → *j* 番目の panel は panelfactorl の *p* 番目の要素から ndim[j-1][0] × ndim[j-1][1] の長さを占めます.

q＝nfcnzindexl[j-1] → *j* 番目の panel の行指標を表すベクトルは npanelindexl の *q* 番目の要素から ndim[j-1][0] の長さを占めます.

panel は大きさ ndim[j-1][0]×ndim[j-1][1] の配列と見なせます.

panel[t-1][s-1] , s ≥ t, s = 1,…, ndim[j-1][0], t = 1,…, ndim[j-1][1] に分解結果の下三角行列 **L** が格納されます.

panel[t-1][s-1], s < t, s = 1,…, ndim[j-1][1], t = 1,…, ndim[j-1][1] に単位上三角行列 **U** の対角ブロック部分が対角要素を除き格納されます.

u = nfcnzfactoru[j-1] → *j* 番目の panel は panelfactoru の *u* 番目の要素から(ndim[j-1][2] – ndim[j-1][1])×ndim[j-1][1] の長さを占めます.

v = nfcnzindexu[j-1] → *j* 番目の panel の列指標を表すベクトルは npanelindexu の *v* 番目の要素から ndim[j-1][2] の長さを占めます.

panel は大きさ(ndim[j-1][2] – ndim[j-1][1]) × ndim[j-1][1]の配列と見なせます.

panel[y-1][x-1], x = 1,…, ndim[j-1][2] - ndim[j-1][1], y = 1,…, ndim[j-1][1]

に分解結果の単位上三角行列 **U** の転置行列 $\mathbf{U}^{\mathbf{T}}$ の対角ブロック部分を除いた部分が格納されます.

指標の値は行列 **A** の列番号をもつ node を post order で並び換えた $\mathbf{QAQ}^{\mathbf{T}}$ での列番号を表します.

3. 使用上の注意

a)

直交行列である置換行列 **P** の要素 $\mathbf{p_{ij}} = \mathbf{1}$ のとき, nperm[i-1] = j と表現します.
逆は以下のようにして求めることができます.

```
for (i = 1; i <= n; i++) {
 j = nperm[i-1];
 nperminv[j-1] = i;
}
```

Fill-Reducing Ordering は METIS などを使って求めることができます.
詳細は“付録 参考文献一覧表”の[43], [44]を参照願います.

b)

分解過程で対角要素の大きさの絶対値の相対零判定値にある値を設定したとすると, この値は次の意味を持っています. すなわち, LU 分解の過程で, 対角要素の大きさの絶対値がその値より小さくなった場合に, その値を相対的に零と見なし, icon = 20000 として処理を打ち切ります. epsz の標準値は, 丸め誤差の単位を *u* としたとき, epsz = 16*u* です.
なお, 対角要素の大きさの絶対値が小さくなっても, 処理を続行させたい場合には, epsz に極小の値を与えれば良いのですが, 結果の精度は保証されません.
Static Pivot を指定した場合, 対角要素が spepsz より小さいとき, spepsz で近似して LU 分解を行います.

c)

分解結果を格納する配列 panelfactorl, npanelindexl, panelfactoru, npanelindexu の必要な大きさは,事前には分かりません. 十分大きな配列を割り当てるか, 本関数を呼び出して symbolic decomposition を行った解析の結果を使って割り当てを行うことができます.
例えば, 大きさ 1 の 1 次元配列などを割付けます. そしてその大きさ 1 などの小さな値を nsizefactorl, nsizeindexl, nsizefactoru, nsizeindexu に指定して, isw = 1 で呼び出します.
symbolic decomposition を行い, icon = 31000 で終了し, nsizefactorl, nsizeindexl, nsizefactoru, nsizeindexu に必要な大きさが返却されます. 必要な大きさの配列を割付け直して引数に指定して, isw = 2 で呼び出すことで, symbolic decomposition 以降の処理を続けることができます. 使用例を参照願います.
LU 分解の結果を利用して c_dm_vsrlux を利用して連立 1 次方程式を解く上での注意は“使用上の注意” e)参照.

d)

列番号に対応するノードを考えます. これを post order で順番を並び換えたときの番号の対応関係が nposto に格納されています. post order の *i* 番目のノードが並び換える前の何番目のノードに対応するかを表します. *j* = nposto[i-1]は *j* 番目であることを表します.
a)同様にこれは直交行列である置換行列 **Q** を表し,行列 **A** を $\mathbf{QAQ}^{\mathbf{T}}$ と並び換えることに相当します.

逆変換 $\mathbf{Q}^T$ は以下のようにして求めることができます.

```
for (i = 1; i <= n; i++) {
 j = nposto[i-1];
 npostoinv[j-1] = i;
}
```

e)

本関数で得られた LU 分解の結果を使い, c_dm_vsrlux を引き続いて呼びだすことで連立 1 次方程式 $\mathbf{Ax} = \mathbf{b}$ を解くことができます.
このとき, c_dm_vsrlu で利用した以下の引数を指定します. 使用例を参照願います.
a, nz, nrow, nfcnz, n,
ipledsm, mz, iordering, nperm,
nassign, nsupnum,
nfcnzfactorl, panelfactorl,
nsizefactorl, nfcnzindexl, npanelindexl,
nsizeindexl, ndim,
nfcnzfactoru, panelfactoru, nsizefactoru,
nfcnzindexu, npanelindexu, nsizeindexu, nposto,
sclrow, sclcol,
nfcnzpivot,
npivotp, npivotq, iw2

4. 使用例

連立 1 次方程式 $\mathbf{Ax} = \mathbf{f}$ を解きます.
行列は境界条件として立方体の境界で 0 となる楕円型の偏微分方程式に有限差分法を適用して生成されたものを利用します.

$$-\Delta u + a\nabla u + cu = f$$

ここで $a = (a_1, a_2, a_3)$.
行列は関数 init_mat_diag によって生成されます.
対角形式格納法で生成し, 対角ベクトルの下 6 本を圧縮列格納法で格納します. 結果生成された非対称な行列 **A** に関する連立 1 次方程式を解きます.
(並列実行を行うスレッド数は環境変数(OMP_NUM_THREADS)などで指定できます.
例えば, 4 プロセッサーのシステムで 4 つのスレッドで並列実行するときは, OMP_NUM_THREADS を 4 に設定して実行します.)

```
/* **EXAMPLE** */
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <malloc.h>
#include <omp.h>
#include "cssl.h"

#define NORD    40
#define KX      NORD
#define KY      NORD
#define KZ      NORD
#define N       (KX * KY * KZ)
#define NBORDER (N + 1)
#define NOFFDIAG    6
```

```
#define K       (N + 1)
#define NDIAG   7
#define NALL    (NDIAG*N)
#define WL      (4 * NALL + 6 * N)
#define IW1L    (2 * NALL + 2 * (N + 1) + 16 * N)
#define IW2L    (47 * N + 47 + 4 * (N + 1) + NALL + 2 * (NALL + N))

void init_mat_diag(double, double, double, double, double*, int*, int, int, int,
                   double, double, double, int, int, int);
double errnrm(double*, double*, int);

int MAIN__() {

  int     nofst[NDIAG];
  double  diag[NDIAG][K], diag2[NDIAG][K];
  double  a[K * NDIAG], wc[K * NDIAG];
  int     nrow[K * NDIAG], nfcnz[N + 1], nrowsym[K * NDIAG + N], nfcnzsym[N + 1],
          iwc[K * NDIAG][2];
  int     nperm[N], nposto[N], ndim[N][3], nassign[N], mz[N], iw1[IW1L],
          iw2[IW2L];
  double  w[WL];
  double  *panelfactorl, *panelfactoru;
  int     *npanelindexl, *npanelindexu;
  double  dummyfl, dummyfu;
  int     ndummyil, ndummyiu;
  long    nsizefactorl, nsizeindexl, nsizeindexu, nsizefactoru,
          nfcnzfactorl[N + 1], nfcnzfactoru[N + 1], nfcnzindexl[N + 1],
          nfcnzindexu[N + 1];
  double  b[N], solex[N];
  double  thepsz, epsz, spepsz, sclrow[N], sclcol[N];
  int     ipivot, istatic, nfcnzpivot[N + 1], npivotp[N], npivotq[N], irefine,
          itermax, iter, ipledsm;
  int i, j, nbase, length, numnz, ntopcfg, ncol, nz, icon, iordering,
      isclitermax, isw, nsupnum;
  double va1, va2, va3, vc, xl, yl, zl, err, epsr;

  printf("     LU DECOMPOSITION METHOD\n");
  printf("     FOR SPARSE UNSYMMETRIC REAL MATRICES\n");
  printf("     IN COMPRESSED COLUMN STORAGE\n \n");

  for (i = 0; i < N; i++) {
    solex[i] = 1.0;
  }

  printf("     EXPECTED SOLUTIONS\n");
  printf("     X(1) = %18.15lf  X(N) = %18.15lf\n \n", solex[0], solex[N-1]);

  va1 = 1.0;
  va2 = 2.0;
```

```
va3 = 3.0;
vc = 4.0;
xl = 1.0;
yl = 1.0;
zl = 1.0;

init_mat_diag(va1, va2, va3, vc, (double *)diag, nofst, KX, KY, KZ,
              xl, yl, zl, NDIAG, N, K);

for (i = 0; i < NDIAG; i++) {
  for (j = 0; j < K; j++) {
    diag2[i][j] = 0;
  }
}

for (i = 0; i < NDIAG; i++) {
  if (nofst[i] < 0) {
    nbase = -nofst[i];
    length = N - nbase;
    for (j = 0; j < length; j++) {
      diag2[i][j] = diag[i][nbase + j];
    }
  } else {
    nbase = nofst[i];
    length = N - nbase;
    for (j = 0; j < length; j++) {
      diag2[i][nbase + j] = diag[i][j];
    }
  }
}

numnz = 1;

for (j = 0; j < N; j++) {
  ntopcfg = 1;

  for (i = NDIAG - 1; i >= 0; i--) {
    if (ntopcfg == 1) {
      nfcnz[j] = numnz;
      ntopcfg = 0;
    }

    if (j + 1 < NBORDER && i + 1 > NOFFDIAG) {
      continue;
    } else {
      if (diag2[i][j] != 0.0) {
        ncol = (j + 1) - nofst[i];
        a[numnz - 1] = diag2[i][j];
        nrow[numnz - 1] = ncol;
```

```
            numnz++;
          }
        }
      }

    }

    nfcnz[N] = numnz;
    nz = numnz - 1;

    c_dm_vmvscc(a, nz, nrow, nfcnz, N, solex, b, wc, (int *)iwc, &icon);

  /* INITIAL CALL WITH IORDER=1 */

    iordering = 0;
    ipledsm = 1;
    isclitermax = 10;
    isw = 1;
    nsizefactorl = 1;
    nsizefactoru = 1;
    nsizeindexl = 1;
    nsizeindexu = 1;
    epsz = 1.0e-16;
    thepsz = 1.0e-2;
    spepsz = 0.0;
    ipivot = 40;
    istatic = 0;
    irefine = 1;
    epsr = 0.0;
    itermax = 10;

    c_dm_vsrlu(a, nz, nrow, nfcnz, N, ipledsm, mz, isclitermax, &iordering,
               nperm, isw, nrowsym, nfcnzsym, nassign, &nsupnum, nfcnzfactorl,
               &dummyfl, &nsizefactorl, nfcnzindexl, &ndummyil, &nsizeindexl,
               (int *)ndim, nfcnzfactoru, &dummyfu, &nsizefactoru, nfcnzindexu,
               &ndummyiu, &nsizeindexu, nposto, sclrow, sclcol, &epsz, &thepsz,
               ipivot, istatic, &spepsz, nfcnzpivot, npivotp, npivotq, w, iw1,
               iw2, &icon);

    printf(" ICON= %d  NSIZEFACTORL= %d  NSIZEFACTORU= %d NSIZEINDEXL= %d",
           icon, nsizefactorl, nsizefactoru, nsizeindexl);
    printf(" NSIZEINDEXU= %d NSUPNUM= %d\n", nsizeindexu, nsupnum);

    panelfactorl = (double *)malloc(nsizefactorl * sizeof(double));
    panelfactoru = (double *)malloc(nsizefactoru * sizeof(double));
    npanelindexl = (int *)malloc(nsizeindexl * sizeof(int));
    npanelindexu = (int *)malloc(nsizeindexu * sizeof(int));

    isw = 2;
```

```
  c_dm_vsrlu(a, nz, nrow, nfcnz, N, ipledsm, mz,isclitermax, &iordering, nperm,
             isw, nrowsym, nfcnzsym, nassign, &nsupnum,nfcnzfactorl,
             panelfactorl, &nsizefactorl, nfcnzindexl, npanelindexl,
             &nsizeindexl, (int *)ndim, nfcnzfactoru, panelfactoru,
             &nsizefactoru, nfcnzindexu, npanelindexu, &nsizeindexu, nposto,
             sclrow, sclcol, &epsz, &thepsz, ipivot, istatic, &spepsz,
             nfcnzpivot, npivotp, npivotq, w, iw1, iw2, &icon);

  c_dm_vsrlux(N, iordering, nperm, b, nassign, nsupnum, nfcnzfactorl,
              panelfactorl, nsizefactorl, nfcnzindexl, npanelindexl,
              nsizeindexl, (int *)ndim, nfcnzfactoru, panelfactoru,
              nsizefactoru, nfcnzindexu, npanelindexu, nsizeindexu, nposto,
              ipledsm, mz, sclrow, sclcol, nfcnzpivot, npivotp, npivotq,
              irefine, epsr, itermax, &iter, a, nz, nrow, nfcnz, iw2, &icon);

  err = errnrm(solex, b, N);

  printf("     COMPUTED VALUES\n");
  printf("     X(1) = %18.15lf  X(N) = %18.15lf\n \n", b[0], b[N-1]);
  printf("     ICON =  %d\n \n", icon);
  printf("     N = %6d\n \n", N);
  printf("     ERROR = %18.15lf\n", err);
  printf("     ITER= %d\n \n \n", iter);

  if (err < 1.0e-8 && icon == 0) {
    printf(" ********** OK **********\n");
  } else {
    printf(" ********** NG **********\n");
  }

  free(panelfactorl);
  free(panelfactoru);
  free(npanelindexl);
  free(npanelindexu);

  return(0);
}

/* ======================================
       INITIALIZE COEFFICIENT MATRIX
   ====================================== */
void init_mat_diag(double va1, double va2, double va3, double vc, double *d_l,
                   int *offset, int nx, int ny, int nz, double xl, double yl,
                   double zl, int ndiag, int len, int ndivp) {

  if (ndiag < 1) {
    printf("FUNCTION INIT_MAT_DIAG:\n");
    printf(" NDIAG SHOULD BE GREATER THAN OR EQUAL TO 1\n");
```

```
    return;
  }

#pragma omp parallel default(shared)
{
  int i, j, l, ndiag_loc, nxy, js, k0, j0, i0;
  double hx, hy, hz, hx2, hy2, hz2;

  ndiag_loc = ndiag;
  if (ndiag > 7) ndiag_loc = 7;

/* INITIAL SETTING */
  hx = xl / (nx + 1);
  hy = yl / (ny + 1);
  hz = zl / (nz + 1);

#pragma omp for
  for (i = 0; i < ndivp; i++) {
    for (j = 0; j < ndiag; j++) {
      d_l[(j * ndivp) + i] = 0.0;
    }
  }

  nxy = nx * ny;

/* OFFSET SETTING */
#pragma omp single
  {
    l = 0;
    if (ndiag_loc >= 7) {
      offset[l] = -nxy;
      l++;
    }
    if (ndiag_loc >= 5) {
      offset[l] = -nx;
      l++;
    }
    if (ndiag_loc >= 3) {
      offset[l] = -1;
      l++;
    }
    offset[l] = 0;
    l++;
    if (ndiag_loc >= 2) {
      offset[l] = 1;
      l++;
    }
    if (ndiag_loc >= 4) {
      offset[l] = nx;
```

```
      l++;
    }
    if (ndiag_loc >= 6) {
      offset[l] = nxy;
    }
  }

/* MAIN LOOP */
#pragma omp for
  for (j = 0; j < len; j++) {
    js = j + 1;

    k0 = (js - 1) / nxy + 1;
    if (k0 > nz) {
      printf("ERROR; K0.GH.NZ \n");
      goto label_100;
    }
    j0 = (js - 1 - nxy * (k0 - 1)) / nx + 1;
    i0 = js - nxy * (k0 - 1) - nx * (j0 - 1);
    l = 0;

    if (ndiag_loc >= 7) {
      if (k0 > 1) d_l[(l * ndivp) + j] = -(1.0 / hz + 0.5 * va3) / hz;
      l++;
    }
    if (ndiag_loc >= 5) {
      if (j0 > 1) d_l[(l * ndivp) + j] = -(1.0 / hy + 0.5 * va2) / hy;
      l++;
    }
    if (ndiag_loc >= 3) {
      if (i0 > 1) d_l[(l * ndivp) + j] = -(1.0 / hx + 0.5 * va1) / hx;
      l++;
    }
    hx2 = hx * hx;
    hy2 = hy * hy;
    hz2 = hz * hz;
    d_l[(l * ndivp) + j] = 2.0 / hx2 + vc;
    if (ndiag_loc >= 5) {
      d_l[(l * ndivp) + j] += 2.0 / hy2;
      if (ndiag_loc >= 7) {
        d_l[(l * ndivp) + j] += 2.0 / hz2;
      }
    }
    l++;
    if (ndiag_loc >= 2) {
      if (i0 < nx) d_l[(l * ndivp) + j] = -(1.0 / hx - 0.5 * va1) / hx;
      l++;
    }
    if (ndiag_loc >= 4) {
```

```
      if (j0 < ny) d_l[(l * ndivp) + j] = -(1.0 / hy - 0.5 * va2) / hy;
      l++;
    }
    if (ndiag_loc >= 6) {
      if (k0 < nz) d_l[(l * ndivp) + j] = -(1.0 / hz - 0.5 * va3) / hz;
    }
label_100: ;
  }

}

  return;
}

/* ======================================
  * SOLUTE ERROR
  * | X1 - X2 |
   ====================================== */
double errnrm(double *x1, double *x2, int len) {

  double rtc, s, ss;
  int i;

  s = 0.0;
  for (i = 0; i < len; i++) {
    ss = x1[i] - x2[i];
    s = s + ss * ss;
  }

  rtc = sqrt(s);
  return(rtc);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VSRLU の項目及び[2], [13], [17], [19], [22], [23], [48], [57], [63], [68], [69]を参照してください.

c_dm_vsrlux

LU 分解された実スパース行列の連立 1 次方程式
ierr = c_dm_vsrlux(n, iordering, nperm, b, nassign, nsupnum, nfcnzfactorl, panelfactorl, nsizefactorl, nfcnzindexl, npanelindexl, nsizeindexl, ndim, nfcnzfactoru, panelfactoru, nsizefactoru, nfcnzindexu, npanelindexu, nsizeindexu, nposto, ipledsm, mz, sclrow, sclcol, nfcnzpivot, npivotp, npivotq, irefine, epsr, itermax, &iter, a, nz, nrow, nfcnz, iw2, &icon);

1. 機能

$n \times n$ の実スパース行列 **A** に対角要素に大きな要素を並べる並び換えを行い, 行および列の均衡化を行うスケーリングを行い, そしてスーパーノード内で Pivot をとり, 以下のように LU 分解されています. LU 分解の結果を利用して以下の連立 1 次方程式を解きます.

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

行列 **A** は以下のように分解されています.

$$\mathbf{P_{rs}QPD_rAP_cD_cP^TQ^TP_{cs}} = \mathbf{LU}$$

ここで, 実スパース行列 **A** は以下のように変換されています.

$$\mathbf{A_1} = \mathbf{D_rAP_cD_c}$$

ここで $\mathbf{P_c}$ は列の入れ換えを行う直交行列, $\mathbf{D_r}$ は行のスケーリングを行う対角行列, $\mathbf{D_c}$ は列のスケーリングを行う対角行列です.

$$\mathbf{A_2} = \mathbf{QPA_1P^TQ^T}$$

$\mathbf{A_2}$ は, スーパーノード内に閉じて指定された Pivot を行い, 行および列の入れ換えを行い LU 分解されています.
$\mathbf{P_{rs}}$, $\mathbf{P_{cs}}$ はそれぞれ行の入れ換えおよび列の入れ換えを示す直交行列です. 実際の入れ換えはスーパーノードに属する限られた行列の部分で行われます.
ただし, **P** は, $\mathbf{SYM} = \mathbf{A_1} + \mathbf{A_1^T}$ の非ゼロパターンに対して求めた `ordering` による行列要素の並び換えを示す置換行列で, **Q** は **SYM** に対する post order による行列要素の並び換えを示す置換行列を示します. **P**, **Q** は直交行列です.

L は下三角行列, **U** は単位上三角行列です.
解の反復改良で精度を改良することを指定できます.

2. 引数

呼出し形式:

```
ierr = c_dm_vsrlux(n, iordering, nperm, b, nassign, nsupnum, nfcnzfactorl,
            panelfactorl, nsizefactorl, nfcnzindexl, npanelindexl,
            nsizeindexl, (int *)ndim, nfcnzfactoru, panelfactoru,
            nsizefactoru, nfcnzindexu, npanelindexu, nsizeindexu, nposto,
            ipledsm, mz, sclrow, sclcol, nfcnzpivot, npivotp, npivotq,
            irefine, &epsr, itermax, &iter, a, nz, nrow, nfcnz, iw2, &icon);
```

引数の説明:

n	int	入力	行列 **A** の次数 n.
iordering	int	入力	11 のとき, nperm に ordering を指定して LU 分解されたことを示します. $\mathbf{PA_1P^T}$ を LU 分解しました. 11 以外のとき, ordering は指定されていません. ("使用上の注意" a)参照)
nperm	int nperm[n]	入力	iordering = 11 のとき使用する置換行列をベクトルで指定します. ("使用上の注意" b)参照)
b	double b[n]	入力	**Ax** = **b** の右辺定数ベクトル.
		出力	解ベクトル.
nassign	int nassign[n]	入力	各 supernode に対する **L**, **U** は圧縮して 2 次元の panel に格納します.この panel を panelfactorl および panelfactoru の 1 次元部分配列として順番に割り付けたとき何番目になるかを示します. これらの指標ベクトルも同じように npanelindexl, npanelindexu に割り付けます. j = nassign[i-1]のとき, *i* 番目の supernode を *j* 番目に割り付けたことを示します. 分解結果の格納方法については図 c_dm_vsrlux-1 を参照してください.
nsupnum	int	入力	supernode の総数. (≤ n)
nfcnzfactorl	long nfcnzfactorl[n+1]	入力	実スパース行列の LU 分解の結果の行列 **L** および **U** はスーパーノードに対応しておのおの求めます. 各 supernode に対応する **L** の列ベクトルは, 共通の行指標ベクトルを持つように圧縮してブロック対角部分に **U** の対応部分を含めて 2 次元の panel に格納します. この panel を panelfactorl の 1 次元部分配列として順番に割り付けたとき, *i* 番目の panel の先頭要素 panel[0][0]が 1 次元配列の panelfactorl の何番目の要素になるかを示します. 結果の格納方法については図 c_dm_vsrlux-1 を参照してください.
panelfactorl	double panelfactorl [nsizefactorl]	入力	各 supernode の分解結果に対応する行列 **L** の複数の列ベクトルと行列 **U** のブロック対角部分について, 共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します. panel を順番に割り付けます. *i* 番目の supernode に対応する panel が何番目の位置に割り当てられるかは j = nassign[i-1]から分かります. その先頭位置は nfcnzfactorl[j-1]に格納されています. panel ごとに分解結果が格納されます.

			i 番目に割付けられた panel の大きさは ndim[i-1][0] × ndim[i-1][1]の 2 次元配列と見なせます. *i* 番目の panel の panel[t-1][s-1], s ≥ t, s = 1,…, ndim[i-1][0], t = 1,…, ndim[i-1][1]に下三角行列 **L** が格納されます. panel の panel[t-1][s-1], s < t, t = 1,…, ndim[i-1][1]には単位上三角行列 **U** の対角要素を除いたブロック対角部分が格納されます. 結果の格納方法については図 c_dm_vsrlux-1 を参照してください.
nsizefactorl	long	入力	panelfactorl の大きさを示す.
nfcnzindexl	long nfcnzindexl[n+1]	入力	各 supernode の分解結果に対応する行列 **L** の複数の列ベクトルと行列 **U** のブロック対角部分について, 共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します. この行指標ベクトルを npanelindexl に順番に割り付けたとき *i* 番目の行指標ベクトルの先頭要素が 1 次元配列である npanelindexl の何番目の要素になるかを示します.
npanelindexl	int npanelindexl [nsizeindexl]	入力	各 supernode の分解結果に対応する行列 **L** の複数の列ベクトルと行列 **U** のブロック対角部分について, 共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します. この行指標ベクトルを npanelindexl に順番に割り付けます. i 番目の supernode に対応する panel の行の指標ベクトルが何番目に割り当てられるかは j = nassign[i-1]から分かります. その先頭位置は nfcnzindexl[j-1]に格納されています. この行指標は行列 **SYM** に対する post order で並び換えたときの行の番号です. 結果の格納方法については図 c_dm_vsrlux-1 を参照してください.
nsizeindexl	long	入力	npanelindexl の大きさを示す.
ndim	int ndim[n][3]	入力	ndim[i-1][0]および ndim[i-1][1]は行列 **L** に関して *i* 番目に割り付けられた panel の 1 次元目と 2 次元目の大きさを示します. ndim[i-1][2]は行列 **U** に関して割り付けられた panel を転置したときの 1 次元目の大きさに対角ブロックの大きさを加えた大きさを示します. 結果の格納方法については図 c_dm_vsrlux-1 を参照してください.
nfcnzfactoru	long nfcnzfactoru[n+1]	入力	実スパース行列の LU 分解の結果の行列 **U** に関しては, 各 supernode に対応する **U** の行ベクトルは, ブロック対角部分を除いて共通の列指標ベクトルを持つように圧縮し, 転置して 2 次元の panel に格納します. この panel を panelfactoru の 1 次元部分配列として順番に割り付けたとき, i 番目の panel の先頭要素 panel[0][0]が 1 次元配列の panelfactoru の何番目の要素になるかを示します. 結果の格納方法については図 c_dm_vsrlux-1 を参照してください.

panelfactoru	double panelfactoru [nsizefactoru]	入力	各 supernode の分解結果に対応する行列 **U** のブロック対角部分を除いた複数の行ベクトルについて, 共通の列指標ベクトルを持つように圧縮し, 転置して 2 次元の panel に格納します. panel を順番に割り付けます. *i* 番目の supernode に対応する panel が何番目の位置に割り当てられるかは j = nassign[i-1]から分かります. その先頭位置は nfcnzfactoru[j-1]に格納されています. panel ごとに分解結果が格納されます. *i* 番目に割付けられた panel の大きさは{ndim[i-1][2] – ndim[i-1][1]} × ndim[i-1][1]の 2 次元配列と見なせます. *i* 番目の panel の panel[t-1][s-1], s = 1,…, ndim[i-1][2] – ndim[i-1][1], t = 1,…, ndim[i-1][1]に単位上三角行列 **U** のブロック対角部分を除いた部分が行ベクトルを圧縮し, 転置したものが格納されます. 結果の格納方法については図 c_dm_vsrlux-1 を参照してください.
nsizefactoru	long	入力	panelfactoru の大きさを示す.
nfcnzindexu	long nfcnzindexu[n+1]	入力	各 supernode の分解結果に対応する行列 **U** は対角ブロック部分を除いて複数の行ベクトルについて, 共通の列指標ベクトルを持つように圧縮し, 転置した形で 2 次元の panel に格納します. 対角ブロック部分も含んだこの列指標ベクトルを npanelindexu に順番に割り付けたとき *i* 番目の列指標ベクトルの先頭要素が 1 次元配列である npanelindexu の何番目の要素になるかを示します. 結果の格納方法については図 c_dm_vsrlux-1 を参照してください.
npanelindexu	int npanelindexu [nsizeindexu]	入力	各 supernode の分解結果に対応する行列 **U** の対角ブロックを除いた部分を転置して圧縮して 2 次元の panel に格納します. 対応する列指標ベクトルには対角ブロック部分を含めたものを npanelindexu に順番に割り付けます. *i* 番目の supernode に対応する panel の行の指標ベクトルが何番目に割り当てられるかは j = nassign[i-1]から分かります. その先頭位置は nfcnzindexu[j-1]に格納されています. この行指標は行列 **SYM** に対する post order で並び換えたときの列の番号です. 結果の格納方法については図 c_dm_vsrlux-1 を参照してください.
nsizeindexu	long	入力	npanelindexu の大きさを示す.
nposto	int nposto[n]	入力	post order で *i* 番目の node が行列 **A** の何番目の列番号に対応するかを表す 1 次元ベクトル. (“使用上の注意” c)参照)
ipledsm	int	入力	LU 分解を行うときに対角要素に大きな要素を並べる列の入れ換えを行ったかを指定します. 1 のとき, 列の入れ換えを行いました. 1 以外のとき, 列の入れ換えを行っていません.
mz	int mz[n]	入力	ipledsm に 1 が指定されたとき, 列の入れ換えを表す. mz[i-1] = *j* は行列 $\mathbf{a}_{ij}$ の属する *j* 番目の列を *i* 番目の列

			に移動する. $\mathbf{a}_{ij}$ は対角要素に並べる大きな要素を表す.
sclrow	double sclrow[n]	入力	スケーリング対角行列 $\mathbf{D}_r$. 対角要素が 1 次元配列に格納されます.
sclcol	double sclcol[n]	入力	スケーリング対角行列 $\mathbf{D}_c$. 対角要素が 1 次元配列に格納されます.
nfcnzpivot	int nfcnzpivot [nsupnum+1]	入力	スーパーノード内の相対的な位置でのピボットの行, 列の入れ換えの履歴を格納する位置を示す. *i* 番目の supernode に関する情報が何番目の位置に割り当てられるかは j = nassign[i-1]から分かります.その先頭位置は nfcnzpivot[j-1]に格納されています. *i* 番目のスーパーノードの入れ換え情報は, npivotp, npivotq の is = nfcnzpivot[j-1],..., ie = nfcnzpivot[j-1] + ndim[j-1][2] - 1 番目の要素に格納されます.
npivotp	int npivotp[n]	入力	スーパーノード内の行の入れ換えに関する情報を格納する.
npivotq	int npivotq[n]	入力	スーパーノード内の列の入れ換えに関する情報を格納する.
irefine	int	入力	LU 分解結果を利用して解を求めるときに, 解の反復改良を行うかどうかを示す. 残差ベクトルを計算するときに 4 倍精度で計算します. =1 : 解の反復改良を行う. 反復改良して得られる残差ベクトル $\mathbf{r}_k$ の絶対値の差分がひとつ前の差分の 1/4 より大きくなるまで, 反復改良を行います. ≠1: 解の反復改良を行わない.
epsr	double	入力	解の残差ベクトル **b-Ax** の絶対値が, **b** の絶対値に対して十分小さいかを判定する判定値 epsr ≤ 0.0 のとき 10^{-6} が設定されます.
itermax	int	入力	(≥ 1) 反復改良を行うときの最大反復回数.
iter	int	出力	反復改良を行った回数.
a	double a[nz]	入力	実スパースな係数行列 **A** を圧縮列格納法で格納します. 圧縮列格納法については, 実スパース行列と実ベクトル(圧縮列格納法), c_dm_vmvscc の図 c_dm_vmvscc-1 を参照してください.
nz	int	入力	実スパースな係数行列 **A** の非零要素の総数.
nrow	int nrow[nz]	入力	圧縮列格納法で使用される行指標で a に格納される要素が何番目の行ベクトルに属するかを示します.
nfcnz	int nfcnz[n+1]	入力	行列 **A** の各列の非零要素を列方向に圧縮して順次配列 a に格納するとき,対応する列の最初の非零要素が格納される位置を表します. nfcnz[n] = nz + 1.
iw2	int iw2[47*n+47+nz+4*(n+1)+2*(nz+n)]	作業域	実スパース行列の LU 分解を行う c_dm_vsrlu からのデータの受け渡しに使われます. 呼び出しの間で値を変更してはいけません.
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.
20400	LU 分解された行列の対角要素にゼロがあった	処理を打ち切る.
20500	求めた解ベクトルに対する残差ベクトルのノルムが方程式 **Ax** = **b** の右辺ベクトルのノルムの epsr 倍より大きい. 係数行列は特異に近い可能性がある.	
30000	次のいずれかであった: • n < 1 • nz < 0 • nfcnz[n] ≠ nz + 1 • nsizefactorl < 1 • nsizefactoru < 1 • nsizeindexl < 1 • nsizeindexu < 1 • irefine = 1 のとき itermax < 1	
30100	nperm に指定した置換行列が正しくない.	
30200	nrow[j-1]に格納された *i* 列目の行指標 k が k < 1 または k > n	
30300	*i* 列目の行指標の数 nfcnz[i] – nfcnz[i-1] > n	

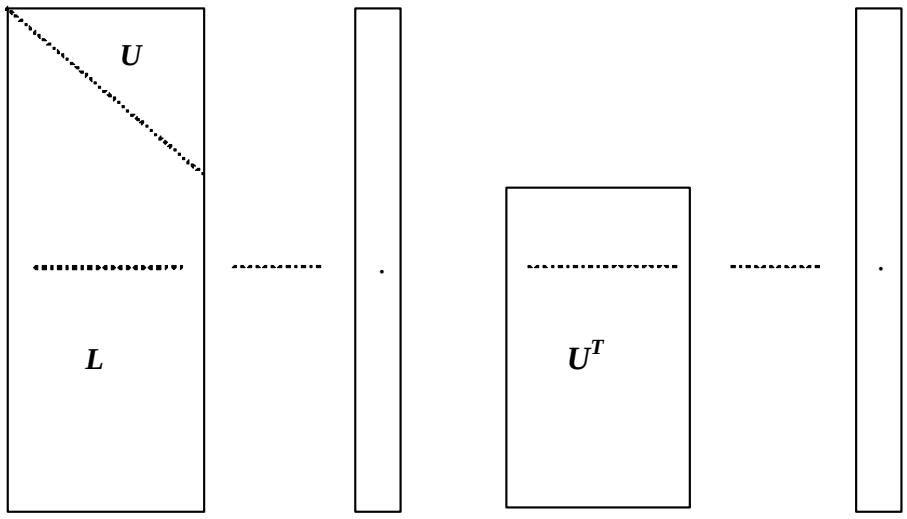


図 c_dm_vsrlux-1 分解結果の格納概念図

j = nassign[i-1]　→ *i* 番目の supernode は *j* 番目に格納されます.

p = nfcnzfactorl[j-1] → *j* 番目の panel は panelfactorl の *p* 番目の要素から ndim[j-1][0] × ndim[j-1][1] の長さを占めます.

q = nfcnzindexl[j-1] → *j* 番目の panel の行指標を表すベクトルは npanelindexl の *q* 番目の要素から ndim[j-1][0] の長さを占めます.

panel は大きさ ndim[j-1][0] × ndim[j-1][1] の配列と見なせます.

panel[t-1][s-1], s ≥ t, s = 1,… , ndim[j-1][0], t = 1,…, ndim[j-1][1] に分解結果の下三角行列 **L** が格納されます.

panel[t-1][s-1], s < t, s = 1,…, ndim[j-1][1], t = 1,…, ndim[j-1][1] に単位上三角行列 **U** の対角ブロック部分が対角要素を除き格納されます.

u = nfcnzfactoru[j-1] → *j* 番目の panel は panelfactoru の *u* 番目の要素から(ndim[j-1][2] – ndim[j-1][1]) × ndim[j-1][1] の長さを占めます.

v = nfcnzindexu[j-1] → *j* 番目の panel の列指標を表すベクトルは npanelindexu の *v* 番目の要素から ndim[j-1][2] の長さを占めます.

panel は大きさ(ndim[j-1][2] – ndim[j-1][1]) × ndim[j-1][1] の配列と見なせます.

panel[y-1][x-1], x = 1,…, ndim[j-1][2] - ndim[j-1][1], y = 1,…, ndim[j-1][1]

に分解結果の単位上三角行列 **U** の転置行列 $\mathbf{U}^{\mathrm{T}}$ の対角ブロック部分を除いた部分が格納されます.

指標の値は行列 **A** の列番号をもつ node を post order で並び換えた $\mathbf{QAQ}^{\mathrm{T}}$ での列番号を表します.

3. 使用上の注意

a)

c_dm_vsrlu で LU 分解を行った結果を利用します.
c_dm_vsrlu の“使用上の注意” e)参照や c_dm_vsrlux の使用例を参照願います.

b)

直交行列である置換行列 **P** の要素 $p_{ij} = 1$ のとき, nperm[i-1] = j と表現します.
逆は以下のようにして求めることができます.

```
for (i = 1; i <= n; i++) {
 j = nperm[i-1];
 npermінv[j-1] = i;
}
```

c)

列番号に対応するノードを考えます. これを post order で順番を並び換えたときの番号の対応関係が nposto に格納されています. post order の *i* 番目のノードが並び換える前の何番目のノードに対応するかを表します. j=nposto[i-1]は *j* 番目であることを表します.
b)同様にこれは直交行列である置換行列 **Q** を表し,行列 **A** を $\mathbf{QAQ}^{\mathrm{T}}$ と並び換えることに相当します.
逆変換 $\mathbf{Q}^{\mathrm{T}}$ は以下のようにして求めることができます.

```
for (i = 1; i <= n; i++) {
 j = nposto[i-1];
 npostoinv[j-1] = i;
}
```

4. 使用例

連立 1 次方程式 **Ax** = **f** を解きます.
行列は境界条件として立方体の境界で 0 となる楕円型の偏微分方程式に有限差分法を適用して生成されたものを利用します.

$$-\Delta u + a\nabla u + cu = f$$

ここで $a = (a_1, a_2, a_3)$.
行列は関数 init_mat_diag によって生成されます.
対角形式格納法で生成し, 対角ベクトルの下 6 本を圧縮列格納法で格納します. 結果生成された非対称な行列 **A** に関する連立 1 次方程式を解きます.
(並列実行を行うスレッド数は環境変数(OMP_NUM_THREADS)などで指定できます.
例えば, 4 プロセッサーのシステムで 4 つのスレッドで並列実行するときは, OMP_NUM_THREADS を 4 に設定して実行します.)

```
/* **EXAMPLE** */
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <malloc.h>
#include <omp.h>
#include "cssl.h"


#define NORD    40
#define KX      NORD
#define KY      NORD
#define KZ      NORD
#define N       (KX * KY * KZ)
#define NBORDER (N + 1)
#define NOFFDIAG    6
#define K       (N + 1)
#define NDIAG   7
#define NALL    (NDIAG*N)
#define WL      (4 * NALL + 6 * N)
#define IW1L    (2 * NALL + 2 * (N + 1) + 16 * N)
#define IW2L    (47 * N + 47 + 4 * (N + 1) + NALL + 2 * (NALL + N))

void init_mat_diag(double, double, double, double, double*, int*, int, int, int,
                   double, double, double, int, int, int);
double errnrm(double*, double*, int);

int MAIN__() {

  int     nofst[NDIAG];
  double  diag[NDIAG][K], diag2[NDIAG][K];
  double  a[K * NDIAG], wc[K * NDIAG];
  int     nrow[K * NDIAG], nfcnz[N + 1], nrowsym[K * NDIAG + N], nfcnzsym[N + 1],
          iwc[K * NDIAG][2];
  int     nperm[N], nposto[N], ndim[N][3], nassign[N], mz[N], iw1[IW1L],
          iw2[IW2L];
  double  w[WL];
  double  *panelfactorl, *panelfactoru;
  int     *npanelindexl, *npanelindexu;
```

```
double  dummyfl, dummyfu;
int     ndummyil, ndummyiu;
long    nsizefactorl, nsizeindexl, nsizeindexu, nsizefactoru,
        nfcnzfactorl[N + 1], nfcnzfactoru[N + 1], nfcnzindexl[N + 1],
        nfcnzindexu[N + 1];
double  b[N], solex[N];
double  thepsz, epsz, spepsz, sclrow[N], sclcol[N];
int     ipivot, istatic, nfcnzpivot[N + 1], npivotp[N], npivotq[N], irefine,
        itermax, iter, ipledsm;
int i, j, nbase, length, numnz, ntopcfg, ncol, nz, icon, iordering,
    isclitermax, isw, nsupnum;
double va1, va2, va3, vc, xl, yl, zl, err, epsr;

printf("     LU DECOMPOSITION METHOD\n");
printf("     FOR SPARSE UNSYMMETRIC REAL MATRICES\n");
printf("     IN COMPRESSED COLUMN STORAGE\n \n");

for (i = 0; i < N; i++) {
  solex[i] = 1.0;
}

printf("     EXPECTED SOLUTIONS\n");
printf("     X(1) = %18.15lf  X(N) = %18.15lf\n \n", solex[0], solex[N-1]);

va1 = 1.0;
va2 = 2.0;
va3 = 3.0;
vc = 4.0;
xl = 1.0;
yl = 1.0;
zl = 1.0;

init_mat_diag(va1, va2, va3, vc, (double *)diag, nofst, KX, KY, KZ,
              xl, yl, zl, NDIAG, N, K);

for (i = 0; i < NDIAG; i++) {
  for (j = 0; j < K; j++) {
    diag2[i][j] = 0;
  }
}

for (i = 0; i < NDIAG; i++) {
  if (nofst[i] < 0) {
    nbase = -nofst[i];
    length = N - nbase;
    for (j = 0; j < length; j++) {
      diag2[i][j] = diag[i][nbase + j];
    }
  } else {
```

```
      nbase = nofst[i];
      length = N - nbase;
      for (j = 0; j < length; j++) {
        diag2[i][nbase + j] = diag[i][j];
      }
    }
  }

  numnz = 1;

  for (j = 0; j < N; j++) {
    ntopcfg = 1;

    for (i = NDIAG - 1; i >= 0; i--) {
      if (ntopcfg == 1) {
        nfcnz[j] = numnz;
        ntopcfg = 0;
      }

      if (j + 1 < NBORDER && i + 1 > NOFFDIAG) {
        continue;
      } else {
        if (diag2[i][j] != 0.0) {
          ncol = (j + 1) - nofst[i];
          a[numnz - 1] = diag2[i][j];
          nrow[numnz - 1] = ncol;
          numnz++;
        }
      }
    }

  }

  nfcnz[N] = numnz;
  nz = numnz - 1;

  c_dm_vmvscc(a, nz, nrow, nfcnz, N, solex, b, wc, (int *)iwc, &icon);

/* INITIAL CALL WITH IORDER=1 */

  iordering = 0;
  ipledsm = 1;
  isclitermax = 10;
  isw = 1;
  nsizefactorl = 1;
  nsizefactoru = 1;
  nsizeindexl = 1;
  nsizeindexu = 1;
  epsz = 1.0e-16;
```

```
thepsz = 1.0e-2;
spepsz = 0.0;
ipivot = 40;
istatic = 0;
irefine = 1;
epsr = 0.0;
itermax = 10;

c_dm_vsrlu(a, nz, nrow, nfcnz, N, ipledsm, mz, isclitermax, &iordering,
           nperm, isw, nrowsym, nfcnzsym, nassign, &nsupnum, nfcnzfactorl,
           &dummyfl, &nsizefactorl, nfcnzindexl, &ndummyil, &nsizeindexl,
           (int *)ndim, nfcnzfactoru, &dummyfu, &nsizefactoru, nfcnzindexu,
           &ndummyiu, &nsizeindexu, nposto, sclrow, sclcol, &epsz, &thepsz,
           ipivot, istatic, &spepsz, nfcnzpivot, npivotp, npivotq, w, iw1,
           iw2, &icon);

printf(" ICON= %d  NSIZEFACTORL= %d  NSIZEFACTORU= %d NSIZEINDEXL= %d",
       icon, nsizefactorl, nsizefactoru, nsizeindexl);
printf(" NSIZEINDEXU= %d NSUPNUM= %d\n", nsizeindexu, nsupnum);

panelfactorl = (double *)malloc(nsizefactorl * sizeof(double));
panelfactoru = (double *)malloc(nsizefactoru * sizeof(double));
npanelindexl = (int *)malloc(nsizeindexl * sizeof(int));
npanelindexu = (int *)malloc(nsizeindexu * sizeof(int));

isw = 2;

c_dm_vsrlu(a, nz, nrow, nfcnz, N, ipledsm, mz,isclitermax, &iordering, nperm,
           isw, nrowsym, nfcnzsym, nassign, &nsupnum,nfcnzfactorl,
           panelfactorl, &nsizefactorl, nfcnzindexl, npanelindexl,
           &nsizeindexl, (int *)ndim, nfcnzfactoru, panelfactoru,
           &nsizefactoru, nfcnzindexu, npanelindexu, &nsizeindexu, nposto,
           sclrow, sclcol, &epsz, &thepsz, ipivot, istatic, &spepsz,
           nfcnzpivot, npivotp, npivotq, w, iw1, iw2, &icon);

c_dm_vsrlux(N, iordering, nperm, b, nassign, nsupnum, nfcnzfactorl,
            panelfactorl, nsizefactorl, nfcnzindexl, npanelindexl,
            nsizeindexl, (int *)ndim, nfcnzfactoru, panelfactoru,
            nsizefactoru, nfcnzindexu, npanelindexu, nsizeindexu, nposto,
            ipledsm, mz, sclrow, sclcol, nfcnzpivot, npivotp, npivotq,
            irefine, epsr, itermax, &iter, a, nz, nrow, nfcnz, iw2, &icon);

err = errnrm(solex, b, N);

printf("     COMPUTED VALUES\n");
printf("     X(1) = %18.15lf  X(N) = %18.15lf\n \n", b[0], b[N-1]);
printf("     ICON =  %d\n \n", icon);
printf("     N = %6d\n \n", N);
printf("     ERROR = %18.15lf\n", err);
```

```
  printf("     ITER= %d\n \n \n", iter);

  if (err < 1.0e-8 && icon == 0) {
    printf(" ********** OK **********\n");
  } else {
    printf(" ********** NG **********\n");
  }

  free(panelfactorl);
  free(panelfactoru);
  free(npanelindexl);
  free(npanelindexu);

  return(0);
}

/* ========================================
       INITIALIZE COEFFICIENT MATRIX
   ======================================== */
void init_mat_diag(double va1, double va2, double va3, double vc, double *d_l,
                   int *offset, int nx, int ny, int nz, double xl, double yl,
                   double zl, int ndiag, int len, int ndivp) {

  if (ndiag < 1) {
    printf("FUNCTION INIT_MAT_DIAG:\n");
    printf(" NDIAG SHOULD BE GREATER THAN OR EQUAL TO 1\n");
    return;
  }

#pragma omp parallel default(shared)
{
  int i, j, l, ndiag_loc, nxy, js, k0, j0, i0;
  double hx, hy, hz, hx2, hy2, hz2;

  ndiag_loc = ndiag;
  if (ndiag > 7) ndiag_loc = 7;

/* INITIAL SETTING */
  hx = xl / (nx + 1);
  hy = yl / (ny + 1);
  hz = zl / (nz + 1);

#pragma omp for
  for (i = 0; i < ndivp; i++) {
    for (j = 0; j < ndiag; j++) {
      d_l[(j * ndivp) + i] = 0.0;
    }
  }
```

```
  nxy = nx * ny;

/* OFFSET SETTING */
#pragma omp single
  {
    l = 0;
    if (ndiag_loc >= 7) {
      offset[l] = -nxy;
      l++;
    }
    if (ndiag_loc >= 5) {
      offset[l] = -nx;
      l++;
    }
    if (ndiag_loc >= 3) {
      offset[l] = -1;
      l++;
    }
    offset[l] = 0;
    l++;
    if (ndiag_loc >= 2) {
      offset[l] = 1;
      l++;
    }
    if (ndiag_loc >= 4) {
      offset[l] = nx;
      l++;
    }
    if (ndiag_loc >= 6) {
      offset[l] = nxy;
    }
  }

/* MAIN LOOP */
#pragma omp for
  for (j = 0; j < len; j++) {
    js = j + 1;

    k0 = (js - 1) / nxy + 1;
    if (k0 > nz) {
      printf("ERROR; K0.GH.NZ \n");
      goto label_100;
    }
    j0 = (js - 1 - nxy * (k0 - 1)) / nx + 1;
    i0 = js - nxy * (k0 - 1) - nx * (j0 - 1);
    l = 0;

    if (ndiag_loc >= 7) {
      if (k0 > 1) d_l[(l * ndivp) + j] = -(1.0 / hz + 0.5 * va3) / hz;
```

```
      l++;
    }
    if (ndiag_loc >= 5) {
      if (j0 > 1) d_l[(l * ndivp) + j] = -(1.0 / hy + 0.5 * va2) / hy;
      l++;
    }
    if (ndiag_loc >= 3) {
      if (i0 > 1) d_l[(l * ndivp) + j] = -(1.0 / hx + 0.5 * va1) / hx;
      l++;
    }
    hx2 = hx * hx;
    hy2 = hy * hy;
    hz2 = hz * hz;
    d_l[(l * ndivp) + j] = 2.0 / hx2 + vc;
    if (ndiag_loc >= 5) {
      d_l[(l * ndivp) + j] += 2.0 / hy2;
      if (ndiag_loc >= 7) {
        d_l[(l * ndivp) + j] += 2.0 / hz2;
      }
    }
    l++;
    if (ndiag_loc >= 2) {
      if (i0 < nx) d_l[(l * ndivp) + j] = -(1.0 / hx - 0.5 * va1) / hx;
      l++;
    }
    if (ndiag_loc >= 4) {
      if (j0 < ny) d_l[(l * ndivp) + j] = -(1.0 / hy - 0.5 * va2) / hy;
      l++;
    }
    if (ndiag_loc >= 6) {
      if (k0 < nz) d_l[(l * ndivp) + j] = -(1.0 / hz - 0.5 * va3) / hz;
    }
label_100: ;
  }

}

  return;
}

/* =======================================
  * SOLUTE ERROR
  * | X1 - X2 |
   ======================================= */
double errnrm(double *x1, double *x2, int len) {

  double rtc, s, ss;
  int i;
```

```
  s = 0.0;
  for (i = 0; i < len; i++) {
    ss = x1[i] - x2[i];
    s = s + ss * ss;
  }

  rtc = sqrt(s);
  return(rtc);
}
```

c_dm_vsrs

実スパース行列の連立 1 次方程式(LU 分解法)
ierr = c_dm_vsrs(a, nz, nrow, nfcnz, n, ipledsm, mz, isclitermax, &iordering, nperm, isw, nrowsym, nfcnzsym, b, nassign, &nsupnum, nfcnzfactorl, panelfactorl, &nsizefactorl, nfcnzindexl, npanelindexl, &nsizeindexl, ndim, nfcnzfactoru, panelfactoru, &nsizefactoru, nfcnzindexu, npanelindexu, &nsizeindexu, nposto, sclrow, sclcol, &epsz, &thepsz, ipivot, istatic, &spepsz, nfcnzpivot, npivotp, npivotq, irefine, epsr, itermax, &iter, w, iw1, iw2, &icon);

1. 機能

$n \times n$ の実スパース行列 **A** に対角要素に大きな要素を並べる並び換えを行い. 行および列の均衡化を行うスケーリングを行います. スーパーノードの対角ブロック内で指定した Pivot をとり, LU 分解します.

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

実スパース行列 **A** は以下のように変換できます.

$$\mathbf{A_1} = \mathbf{D_r A P_c D_c}$$

ここで $\mathbf{P_c}$ は列の入れ換えを行う直交行列, $\mathbf{D_r}$ は行のスケーリングを行う対角行列, $\mathbf{D_c}$ は列のスケーリングを行う対角行列です.

$$\mathbf{A_2} = \mathbf{Q P A_1 P^T Q^T}$$

$\mathbf{A_2}$ は, スーパーノードの対角ブロックで指定された Pivot を行い, 行および列の入れ換えを行い LU 分解されます.
ただし, **P** は, $\mathbf{SYM} = \mathbf{A_1 + A_1^T}$ の非ゼロパターンに対して求めた ordering による行列要素の並び換えを示す置換行列で, **Q** は **SYM** に対する post order による行列要素の並び換えを示す置換行列を示します. **P**, **Q** は直交行列です.

L は下三角行列, **U** は単位上三角行列です.

Pivot 処理で, 閾値 thepsz より絶対値が大きな Pivot を見つけることが出来なかったとき, 対角ブロック内の絶対値が最大の要素を Pivot の候補とします. この値が小さ過ぎるときに Static Pivot を与えて近似的に LU 分解を続けることができます.

2. 引数

呼出し形式:

```
ierr = c_dm_vsrs(a, nz, nrow, nfcnz, n, ipledsm, mz, isclitermax,
            &iordering, nperm, isw, nrowsym, nfcnzsym, b, nassign, &nsupnum,
            nfcnzfactorl, panelfactorl, &nsizefactorl, nfcnzindexl,
            npanelindexl, &nsizeindexl, (int *)ndim, nfcnzfactoru,
            panelfactoru, &nsizefactoru, nfcnzindexu, npanelindexu,
            &nsizeindexu, nposto, sclrow, sclcol, &epsz, &thepsz, ipivot,
            istatic, &spepsz, nfcnzpivot, npivotp, npivotq, irefine, epsr,
            itermax, iter, w, iw1, iw2, &icon);
```

引数の説明:

a	double a[nz]	入力	実スパースな係数行列 **A** を圧縮列格納法で格納します. 圧縮列格納法については, 実スパース行列と実ベクトル(圧縮列格納法), c_dm_vmvscc の図 c_dm_vmvscc-1 を参照してください.
nz	int	入力	実スパースな係数行列 **A** の非零要素の総数.
nrow	int nrow[nz]	入力	圧縮列格納法で使用される行指標で a に格納される要素が何番目の行ベクトルに属するかを示します.
nfcnz	int nfcnz[n+1]	入力	行列 **A** の各列の非零要素を列方向に圧縮して順次配列 a に格納するとき,対応する列の最初の非零要素が格納される位置を表します. nfcnz[n] = nz + 1.
n	int	入力	行列 **A** の次数 *n*.
ipledsm	int	入力	対角要素に大きな要素を並べる列の入れ換えを行うかを指定します. 1 のとき, 列の入れ換えを求めて入れ換えます. 1 以外のとき, 列の入れ換えを行いません.
mz	int mz[n]	出力	ipledsm に 1 が指定されたとき, 列の入れ換えを表す. mz[i] = j は行列 $\mathbf{a}_{ij}$ の属する *j* 番目の列を *i* 番目の列に移動する. $\mathbf{a}_{ij}$ は対角要素に並べる大きな要素を表す.
isclitermax	int	入力	係数行列のスケーリングを行う対角行列 $\mathbf{D}_r$ と $\mathbf{D}_c$ を反復して求める反復回数. isclitermax ≤ 0 のときスケーリングは行わすに, $\mathbf{D}_r$ および $\mathbf{D}_c$ は単位行列が設定されます. isclitermax ≥ 10 のときは 10 回を上限値とします.
iordering	int	入力	ordering を表す置換行列 **P** で $\mathbf{PA_1P^T}$ と変換した行列を LU 分解するかを指定します. 置換行列は直交行列です. 10 のとき, isw = 1 で呼び出すと ordering を求める $\mathbf{A_1}$ に関する情報を求めて nrowsym, nfcnzsym に設定します. 11 のとき, isw = 1 で 10 を指定して呼び出して得た行列を対称化した情報 nrowsym, nfcnzsym を使って決めた ordering を nperm に指定して同じく isw = 1 で呼び出し, 計算を続けることを示します. $\mathbf{PA_1P^T}$ を LU 分解する処理を続けます. 10,11 以外のとき, ordering は指定せずに行列 $\mathbf{A_1}$ をそのまま LU 分解します.
		出力	isw = 1 と iordering = 10 を指定して呼び出した後, iordering に 11 が設定されます. そのため, ordering を nperm に指定して再び呼び出すときに特に 11 を指定す

			る必要はありません. (“使用上の注意” a)参照)
nperm	int nperm[n]	入力	iordering = 11 のとき使用する置換行列をベクトルで指定します. (“使用上の注意” a)参照)
isw	int	入力	呼び出しに関する制御情報を示します. 4) 1 のとき 対称化および symbolic decomposition を行い, 必要な領域が割り当てられているか調べ計算を行います. iordering = 10 で呼び出し, ordering を求めるための情報が nrowsym, nfcnzsym に出力されます. これらを使って **SYM** に対する ordering を求めた後, nperm に指定して iordering = 11 を指定してもう 1 回 isw = 1 で呼び出します. iordering が 10,11 以外のときは ordering は行ないません. 5) 2 のとき isw = 1 で呼び出したとき, panelfactorl, panelfactoru, npanelindexl または npanelindexu の大きさが不足して icon = 31000 で終了した処理を継続します. このとき, nsizefactorl, nsizefactoru, nsizeindexl または nsizeindezu に返却された必要な大きさで panelfactorl, panelfactoru, npanelindexl または npanelindexu を確保し直して指定しなおして再度呼び出します. これらの引数と実行を制御する引数 isw 以外の引数に格納されている値は変更してはいけません. 6) 3 のとき 先立つ呼び出しで LU 分解された同じ係数行列に対する連立 1 次方程式の右辺ベクトル **b** を変えて再度解くことを指定します. このとき先立つ呼び出しで LU 分解した結果を使います. 実行を制御する引数 isw と右辺ベクトル **b** を格納する引数 B 以外の引数に格納されている値を変更していけません.
nrowsym	int nrowsym[nz+n]	出力	iordering = 10 で呼び出したとき, 対称化した **SYM** = $\mathbf{A_l} + \mathbf{A_l}^T$ の非ゼロパターンの下三角行列部分の行指標を列圧縮したものが返却されます. ipledsm = 1 のときは, $\mathbf{A_l} = \mathbf{D_r A P_c D_c}$ です.
nfcnzsym	int nfcnzsym[n+1]	出力	iordering = 10 で呼び出したとき, 行列 SYM の下三角部分の各列の非零要素の行指標を列方向に圧縮して順次配列 nrowsym に格納するとき, 対応する列の最初の非零要素が格納される位置を表します. nfcnzsym[n] = nsymz + 1 nsymz は, 総要素数を表します.
b	double b[n]	入力	**Ax** = **b** の右辺定数ベクトル.
		出力	解ベクトル.
nassign	int nassign[n]	出力	各 supernode に対する L, U は圧縮して 2 次元の panel に格納します.

			この panel を panelfactorl および panelfactoru の 1 次元部分配列として順番に割り付けたとき何番目になるかを示します. これらの指標ベクトルも同じように npanelindexl, npanelindexu に割り付けます. j = nassign[i-1]のとき, i 番目の supernode を j 番目に割り付けたことを示します. 分解結果の格納方法については図 c_dm_vsrs-1 を参照してください.
nsupnum	int	出力	supernode の総数.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します. (≤ n)
nfcnzfactorl	long nfcnzfactorl[n+1]	出力	実スパース行列の LU 分解の結果の行列 **L** および **U** はスーパーノードに対応しておのおの求めます. 各 supernode に対応する **L** の列ベクトルは, 共通の行指標ベクトルを持つように圧縮してブロック対角部分に **U** の対応部分を含めて 2 次元の panel に格納します. この panel を panelfactorl の 1 次元部分配列として順番に割り付けたとき, *i* 番目の panel の先頭要素 panel[0][0]が 1 次元配列の panelfactorl の何番目の要素になるかを示します. 結果の格納方法については図 c_dm_vsrs-1 を参照してください.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します.
panelfactorl	double panelfactorl [nsizefactorl]	出力	各 supernode の分解結果に対応する行列 **L** の複数の列ベクトルと行列 **U** のブロック対角部分について, 共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します. panel を順番に割り付けます. i 番目の supernode に対応する panel が何番目の位置に割り当てられるかは j = nassign[i-1]から分かります. その先頭位置は nfcnzfactorl[j]に格納されています. panel ごとに分解結果が格納されます. i 番目に割付けられた panel の大きさは ndim[i-1][0] × ndim[i-1][1]の 2 次元配列と見なせます. *i* 番目の panel の panel[t-1][s-1], s ≥ t, s = 1,…, ndim[i-1][0], t=1,…, ndim[i-1][1]に下三角行列 **L** が格納されます. panel の panel[t-1][s-1], s < t, t = 1,…, ndim[i][1]には単位上三角行列 **U** の対角要素を除いたブロック対角部分が格納されます. 結果の格納方法については図 c_dm_vsrs-1 を参照してください. (“使用上の注意” c)参照)
nsizefactorl	long	入力	panelfactorl の大きさを示す.
		出力	panelfactorl の大きさとして必要な大きさが返却されます. (“使用上の注意” c)参照)
nfcnzindexl	long nfcnzindexl[n+1]	出力	各 supernode の分解結果に対応する行列 **L** の複数の列ベクトルと行列 **U** のブロック対角部分について, 共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します. この行指標ベクトルを npanelindexl に順番に割り付けたとき i 番目の行指標ベクトルの先頭要素が 1 次元配列である npanelindexl の何番目の要素になるかを示します.

			結果の格納方法については図 c_dm_vsrs-1 を参照してください.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します.
npanelindexl	int npanelindexl [nsizeindexl]	出力	各 supernode の分解結果に対応する行列 **L** の複数の列ベクトルと行列 **U** のブロック対角部分について，共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します. この行指標ベクトルを npanelindexl に順番に割り付けます. *i* 番目の supernode に対応する panel の行の指標ベクトルが何番目に割り当てられるかは j = nassign[i-1]から分かります. その先頭位置は nfcnzindexl[j-1]に格納されています. この行指標は行列 **SYM** に対する post order で並び換えたときの行の番号です. 結果の格納方法については図 c_dm_vsrs-1 を参照してください. (“使用上の注意” c)参照)
nsizeindexl	long	入力	npanelindexl の大きさを示す.
		出力	npanelindexl の大きさとして必要な大きさが返却されます. (“使用上の注意” c)参照)
ndim	int ndim[n][3]	出力	ndim[i-1][0]および ndim[i-1][1]は行列 **L** に関して *i* 番目に割り付けられた panel の 1 次元目と 2 次元目の大きさを示します. ndim[i-1][2]は行列 **U** に関して割り付けられた panel を転置したときの 1 次元目の大きさに対角ブロックの大きさを加えた大きさを示します. 結果の格納方法については図 c_dm_vsrs-1 を参照してください.
		入力	isw≠1 のとき,初回呼び出しの値を再利用します.
nfcnzfactoru	long nfcnzfactoru[n+1]	出力	実スパース行列の LU 分解の結果の行列 **U** に関しては, 各 supernode に対応する **U** の行ベクトルは, ブロック対角部分を除いて共通の列指標ベクトルを持つように圧縮し, 転置したものを 2 次元の panel に格納します. この panel を panelfactoru の 1 次元部分配列として順番に割り付けたとき, *i* 番目の panel の先頭要素 panel[0][0]が 1 次元配列の panelfactoru の何番目の要素になるかを示します. 結果の格納方法については図 c_dm_vsrs-1 を参照してください.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します.
panelfactoru	double panelfactoru [nsizefactoru]	出力	各 supernode の分解結果に対応する行列 **U** のブロック対角部分を除いた複数の行ベクトルについて, 共通の列指標ベクトルを持つように圧縮し, 転置して 2 次元の panel に格納します. panel を順番に割り付けます. *i* 番目の supernode に対応する panel が何番目の位置に割り当てられるかは j = nassign[i-1]から分かります.その先頭位置は nfcnzfactoru[j-1]に格納されています. panel ごとに分解結果が格納されます. *i* 番目に割付けられた panel の大きさは{ndim[i-1][2] – ndim[i-1][1]} × ndim[i-1][1]の 2 次元配列と見なせます. *i* 番目の panel の panel[t-1][s-1],

			s = 1,..., ndim[i-1][2] – ndim[i-1][1], t = 1,..., ndim[i-1][1]に単位上三角行列 **U** のブロック対角部分を除いた部分が行ベクトルを圧縮し, 転置して格納します. 結果の格納方法については図 c_dm_vsrs-1 を参照してください. ("使用上の注意" c)参照)
nsizefactoru	long	入力	panelfactoru の大きさを示す.
		出力	panelfactoru の大きさとして必要な大きさが返却されます. ("使用上の注意" c)参照)
nfcnzindexu	long nfcnzindexu[n+1]	出力	各 supernode の分解結果に対応する行列 **U** は対角ブロック部分を除いて複数の行ベクトルについて, 共通の列指標ベクトルを持つように圧縮し, 転置して 2 次元の panel に格納します. 対角ブロック部分も含んだこの列指標ベクトルを npanelindexu に順番に割り付けたとき *i* 番目の列指標ベクトルの先頭要素が 1 次元配列である npanelindexu の何番目の要素になるかを示します. 結果の格納方法については図 c_dm_vsrs-1 を参照してください.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します.
npanelindexu	int npanelindexu [nsizeindexu]	出力	各 supernode の分解結果に対応する行列 ***U*** の対角ブロックを除いた部分を転置して圧縮して 2 次元の panel に格納します. この列指標ベクトルを対角ブロック部分を含めて npanelindexu に順番に割り付けます. i 番目の supernode に対応する panel の行の指標ベクトルが何番目に割り当てられるかは j = nassign[i-1]から分かります. その先頭位置は nfcnzindexu[j-1]に格納されています. この行指標は行列 **SYM** に対する post order で並び換えたときの列の番号です. 結果の格納方法については図 c_dm_vsrs-1 を参照してください. ("使用上の注意" c)参照)
nsizeindexu	long	入力	npanelindexu の大きさを示す.
		出力	npanelindexu の大きさとして必要な大きさが返却されます. ("使用上の注意" c)参照)
nposto	int nposto[n]	出力	post order で i 番目の node が行列 **A** の何番目の列番号に対応するかを表す 1 次元ベクトル.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します. ("使用上の注意" d)参照)
sclrow	double sclrow[n]	出力	スケーリング対角行列 $\mathbf{D}_r$. 対角要素が 1 次元配列に格納されます.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します.
sclcol	double sclcol[n]	出力	スケーリング対角行列 $\mathbf{D}_c$. 対角要素が 1 次元配列に格納されます.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します.
epsz	double	入力	分解過程で対角要素の大きさの絶対値の相対判定値
		出力	epsz ≤ 0.0 のときは標準値が設定されます. ("使用上の注意" b)参照)
thepsz	double	入力	ピボットの判定での閾値(threshold). これより大きな値はピボットとして採用します. 見つかればその時点で, ピ

			ボット検索は打ち切ります. 例えば 10^{-2} 程度.
		出力	Thepsz ≤ 0.0 のときは 10^{-2} が設定されます. epsz ≥ thepsz > 0.0 のときは, epsz が設定されます.
ipivot	int	入力	スーパーノード内でピボットを行うか, 行う場合どのようなピボットを行うかを指定します. 例えば, 完全ピボットとして 40 を指定. ipivot < 10 : または ipivot ≥ 50: ピボットなし. 10 ≤ ipivot < 20 : 部分ピボット 20 ≤ ipivot < 30 : 対角ピボット 21 : スーパーノード内で対角ピボットがとれないとき, Rook ピボットに変更します. 22 : スーパーノード内で対角ピボットがとれないときは, Rook ピボットに, Rook ピボットが取れないときは完全ピボットに変更します. 30 ≤ ipivot < 40 :Rook ピボット 32 : スーパーノード内で Rook ピボットがとれないとき, 完全ピボットをとります. 40 ≤ ipivot < 50 : 完全ピボット
istatic	int	入力	Pivot を指定したとき, Static Pivot を行うかを示します. 1) = 1 のとき スーパーノード内で指定したピボットが spepsz より大きくないとき, ピボットとして copysign(spepsz, Pivot) で近似します. ピボットの値が 0.0 なら spepsz に近似します. このとき, 以下を設定しなければなりません. a) epsz は epsz の標準値以下にしなければなりません. b) isclitermax は 10 を設定しスケーリングを行う必要があります. c)thepsz ≥ spepsz でなければなりません. d)解の反復改良を行うため irefine = 1 を指定しなければなりません. 2) ≠ 1 のとき Static Pivot は行いません.
spepsz	double	入力	istatic = 1 のとき, Statc Pivot として使う値. 10^{-10} ≥ spepsz ≥ epsz でなければなりません.
		出力	spepsz < epsz のときは, 10^{-10} が設定されます.
nfcnzpivot	int nfcnzpivot [nsupnum+1]	出力	スーパーノード内の相対的な位置でのピボットの行, 列の入れ換えの履歴を格納する位置を示す. i 番目の supernode に関する情報が何番目の位置に割り当てられるかは j = nassign[i-1]から分かります.その先頭位置は nfcnzpivot[j-1]に格納されています. *i* 番目のスーパーノードの入れ換え情報は, npivotp, npivotq の is = nfcnzpivot[j-1],…, ie = nfcnzpivot[j-1] + ndim[j-1][2] - 1 番目の要素に格納されます.
npivotp	int npivotp[n]	出力	スーパーノード内の行の入れ換えに関する情報を格納する.
npivotq	int npivotq[n]	出力	スーパーノード内の列の入れ換えに関する情報を格納

			する.
irefine	int	入力	LU 分解結果を利用して解を求めるときに, 解の反復改良を行うかどうかを示す. 残差ベクトルを計算するときに 4 倍精度で計算します. = 1 : 解の反復改良を行う. 反復改良して得られる残差ベクトル r_k の絶対値の差分がひとつ前の差分の 1/4 より大きくなるまで, 反復改良を行います. ≠ 1: 解の反復改良を行わない. istatic = 1 のとき, irefine = 1 でなければなりません.
epsr	double	入力	解の残差ベクトル **b - Ax** の絶対値が, **b** の絶対値に対して十分小さいかを判定する判定値. epsr ≤ 0.0 のとき 10^{-6} が設定されます.
itermax	int	入力	(≥ 1) 反復改良を行うときの最大反復回数.
iter	int	出力	反復改良を行った回数.
w	double w[4*nz+6*n]	作業域	isw = 1, 2 で続けて呼び出すとき,呼び出しの間で必要なデータを受け渡すために使われます. 呼び出しの間で値を変更してはいけません.
iw1	int iw1[2*nz+2*(n+1)+16*n]	作業域	isw = 1, 2 で続けて呼び出すとき,呼び出しの間で必要なデータを受け渡すために使われます. 呼び出しの間で値を変更してはいけません.
iw2	int iw2[47*n+47+nz+4*(n+1)+2*(nz+n)]	作業域	isw = 1, 2, 3 で続けて呼び出すとき,呼び出しの間で必要なデータを受け渡すために使われます. 呼び出しの間で値を変更してはいけません.
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になった.	処理を打ち切る.
20100	ipledsm = 1 を指定して,対角要素に絶対値が大きな要素を並べるため maximum matching を求める処理で, 長さ n の maximum matching がみつからなかった. 行列が特異である可能性がある.	
20200	行および列の均衡化を行う対角行列の対角要素を求める過程で,元の行列 A の行もしくは列にゼロベクトルがあった. 行列が特異である可能性がある.	
20400	LU 分解された行列の対角要素にゼロがあった.	
20500	求めた解ベクトルに対する残差ベクトルのノルムの大きさが方程式 Ax = b の右辺ベクトルのノルムの epsr 倍より大きい. 係数行列は特異に近い可能性がある.	

コード	意　味	処理内容
30000	次のいずれかであった: • n < 1 • nz < 0 • nfcnz[n] ≠ nz + 1 • nsizefactorl < 1 • nsizefactoru < 1 • nsizeindexl < 1 • nsizeindexu < 1 • isw < 1 • isw > 3 • irefine = 1 のとき itermax < 1	処理を打ち切る.
30100	nperm に指定した置換行列が正しくない.	
30200	nrow[j-1]に格納された *i* 列目の行指標 k が k < 1 または k > n	
30300	*i* 列目の行指標の数 nfcnz[i] – nfcnz[i-1] > n	
30500	istatic = 1 のとき満たすべき条件をみたしていない. epsz は標準値 16***u*** より大きかった. または isclitermax < 10 であった. または irefine≠1 であった. または spepsz > thepsz であった. または spepsz > 10^{-10} であった.	
31000	panelfactorl の大きさ nsizefactorl または npanelindexl の大きさ nsizeindexl または panelfactoru の大きさ nsizefactoru または npanelindexu の大きさ nsizeindexu が小さすぎる.	nsizefactorl または nsizeindexl または nsizefactoru または nsizeindexu で指定された 大きさで panelfactorl または npanelindexl または panelfactoru または npanelindexu を割り付けて isw = 2 として再度呼び出す.

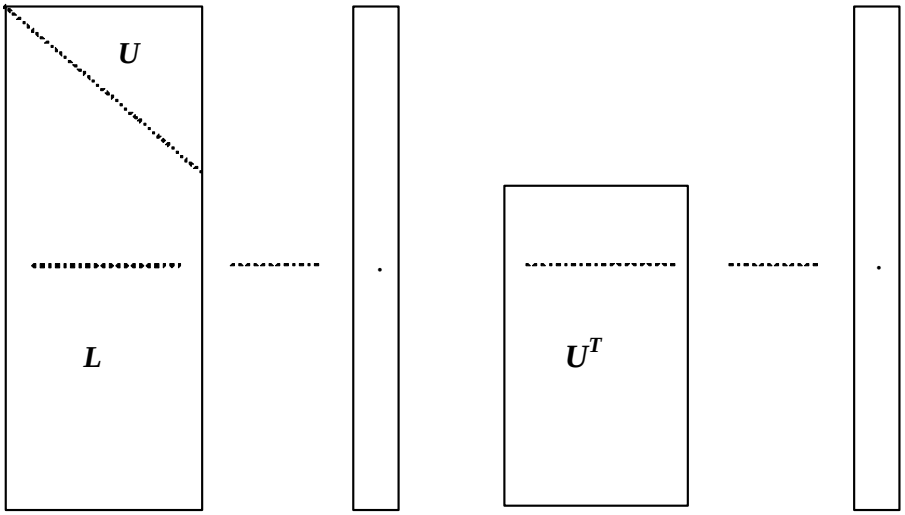


図 c_dm_vsrs-1 分解結果の格納概念図

j = nassign[i-1] → *i* 番目の supernode は *j* 番目に格納されます.

p = nfcnzfactorl[j-1] → *j* 番目の panel は panelfactorl の *p* 番目の要素から ndim[j-1][0] × ndim[j-1][1] の長さを占めます.

q = nfcnzindexl[j-1] → *j* 番目の panel の行指標を表すベクトルは npanelindexl の *q* 番目の要素から ndim[j-1][0] の長さを占めます.

panel は大きさ ndim[j-1][0] × ndim[j-1][1] の配列と見なせます.

panel[t-1][s-1], s ≥ t, s = 1,..., ndim[j-1][0], t = 1,..., ndim[j-1][1] に分解結果の下三角行列 **L** が格納されます.

panel[t-1][s-1], s < t, s = 1,..., ndim[j-1][1], t = 1,..., ndim[j-1][1] に単位上三角行列 **U** の対角ブロック部分が対角要素を除き格納されます.

u = nfcnzfactoru[j-1] → *j* 番目の panel は panelfactoru の *u* 番目の要素から(ndim[j-1][2] – ndim[j-1][1]) × ndim[j-1][1] の長さを占めます.

v = nfcnzindexu[j-1] → j 番目の panel の列指標を表すベクトルは npanelindexu の *v* 番目の要素から ndim[j-1][2]の長さを占めます.

panel は大きさ(ndim[j-1][2] – ndim[j-1][1]) × ndim[j-1][1] の配列と見なせます.

panel[y-1][x-1], x = 1,..., ndim[j-1][2] – ndim[j-1][1], y = 1,..., ndim[j-1][1]

に分解結果の単位上三角行列 **U** の転置行列 $\mathbf{U}^{\mathrm{T}}$ の対角ブロック部分を除いた部分が格納されます.

指標の値は行列 **A** の列番号をもつ node を post order で並び換えた $\mathbf{QAQ}^{\mathrm{T}}$ での列番号を表します.

3. 使用上の注意

a)

直交行列である置換行列 **P** の要素 $\mathbf{p}_{ij} = \mathbf{1}$ のとき, nperm[i-1] = j と表現します.
逆は以下のようにして求めることができます.

```
for (i = 1; i <= n; i++) {
 j = nperm[i-1];
 nperminv[j-1] = i;
}
```

Fill-Reducing Ordering は METIS などを使って求めることができます.
詳細は“付録 参考文献一覧表”の[43], [44]を参照願います.

b)

分解過程で対角要素の大きさの絶対値の相対零判定値にある値を設定したとすると, この値は次の意味を持っています. すなわち, LU 分解の過程で, 対角要素の大きさの絶対値がその値より小さくなった場合に, その値を相対的に零と見なし, icon = 20000 として処理を打ち切ります. epsz の標準値は, 丸め誤差の単位を u としたとき, epsz = 16u です.
なお, 対角要素の大きさの絶対値が小さくなっても, 処理を続行させたい場合には, epsz に極小の値を与えれば良いのですが, 結果の精度は保証されません.
Static Pivot を指定した場合, 対角要素が spepsz より小さいとき, spepsz で近似して LU 分解を行います. このとき, 解の反復改良を指定しなければなりません.

c)

分解結果を格納する配列 panelfactorl, npanelindexl, panelfactoru, npanelindexu の必要な大きさは,事前には分かりません. 十分大きな配列を割り当てるか, 本関数を呼び出して symbolic decomposition を行った解析の結果を使って割り当てを行うことができます.
例えば, 大きさ 1 の 1 次元配列などを割付けます. そしてその大きさ 1 などの小さな値を nsizefactorl, nsizeindexl, nsizefactoru, nsizeindexu に指定して, isw = 1 で呼び出します.
symbolic decomposition を行い, icon = 31000 で終了し, nsizefactorl, nsizeindex, nsizefactoru, nsizeindexu に必要な大きさが返却されます. 必要な大きさの配列を割付け直して引数に指定して, isw = 2 で呼び出すことで, symbolic decomposition 以降の処理を続けることができます. 使用例を参照願います.

d)

列番号に対応するノードを考えます. これを post order で順番を並び換えたときの番号の対応関係が nposto に格納されています. post order の i 番目のノードが並び換える前の何番目のノードに対応するかを表します. j = nposto[i-1]は j 番目であることを表します.
a)同様にこれは直交行列である置換行列 **Q** を表し,行列 **A** を $\mathbf{QAQ}^{\mathbf{T}}$ と並び換えることに相当します.
逆変換 $\mathbf{Q}^{\mathbf{T}}$ は以下のようにして求めることができます.

```
for (i = 1; i <= n; i++) {
 j = nposto[i-1];
 npostoinv[j-1] = i;
}
```

e)

連立1次方程式**Ax** = **b**は, c_dm_vsrluで実スパース行列**A**をLU分解して, 分解結果を利用して引き続いてc_dm_vsrluxを呼び出して解くこともできます.

4. 使用例

連立 1 次方程式 **Ax** = **f** を解きます.
行列は境界条件として立方体の境界で 0 となる楕円型の偏微分方程式に有限差分法を適用して生成された

ものを利用します.
$-\Delta u + a\nabla u + cu = f$

ここで $a = (a_1, a_2, a_3)$.
行列は関数 init_mat_diag によって生成されます.
対角形式格納法で生成し, 対角ベクトルの下 6 本を圧縮列格納法で格納します. 結果生成された非対称な行列 **A** に関する連立 1 次方程式を解きます.
(並列実行を行うスレッド数は環境変数(OMP_NUM_THREADS)などで指定できます.
例えば, 4 プロセッサーのシステムで 4 つのスレッドで並列実行するときは, OMP_NUM_THREADS を 4 に設定して実行します.)

```
/* **EXAMPLE** */
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <malloc.h>
#include <omp.h>
#include "cssl.h"

#define NORD    40
#define KX      NORD
#define KY      NORD
#define KZ      NORD
#define N       (KX * KY * KZ)
#define NBORDER (N + 1)
#define NOFFDIAG    6
#define K       (N + 1)
#define NDIAG   7
#define NALL    (NDIAG * N)
#define WL      (4 * NALL + 6 * N)
#define IW1L    (2 * NALL + 2 * (N + 1) + 16 * N)
#define IW2L    (47 * N + 47 + 4 * (N + 1) + NALL + 2 * (NALL + N))

void init_mat_diag(double, double, double, double, double*, int*, int, int, int,
                   double, double, double, int, int, int);
double errnrm(double*, double*, int);

int MAIN__() {

  int     nofst[NDIAG];
  double  diag[NDIAG][K], diag2[NDIAG][K];
  double  a[K * NDIAG], wc[K * NDIAG];
  int     nrow[K * NDIAG], nfcnz[N + 1], nrowsym[K * NDIAG+N], nfcnzsym[N + 1],
          iwc[K * NDIAG][2];
  int     nperm[N], nposto[N], ndim[N][3], nassign[N], mz[N], iw1[IW1L],
          iw2[IW2L];
  double  w[WL];
  double  *panelfactorl, *panelfactoru;
  int     *npanelindexl, *npanelindexu;
  double  dummyfl, dummyfu;
  int     ndummyil, ndummyiu;
  long    nsizefactorl, nsizeindexl, nsizeindexu, nsizefactoru,
          nfcnzfactorl[N + 1], nfcnzfactoru[N + 1], nfcnzindexl[N + 1],
          nfcnzindexu[N + 1];
  double  b[N], solex[N];
  double  epsz, thepsz, spepsz, sclrow[N], sclcol[N];
```

```
int     ipivot, istatic, nfcnzpivot[N + 1], npivotp[N], npivotq[N], irefine,
        itermax, iter, ipledsm;
int i, j, nbase, length, numnz, ntopcfg, ncol, nz, icon, iordering,
    isclitermax, isw, nsupnum;
double va1, va2, va3, vc, xl, yl, zl, err, epsr;

printf("     LU DECOMPOSITION METHOD\n");
printf("     FOR SPARSE UNSYMMETRIC REAL MATRICES\n");
printf("     IN COMPRESSED COLUMN STORAGE\n \n");

for (i = 0; i < N; i++) {
  solex[i] = 1.0;
}
printf("     EXPECTED SOLUTIONS\n");
printf("     X(1) = %18.15lf  X(N) = %18.15lf\n \n", solex[0], solex[N - 1]);
va1 = 1.0;
va2 = 2.0;
va3 = 3.0;
vc = 4.0;
xl = 1.0;
yl = 1.0;
zl = 1.0;

init_mat_diag(va1, va2, va3, vc, (double *)diag, nofst, KX, KY, KZ,
              xl, yl, zl, NDIAG, N, K);

for (i = 0; i < NDIAG; i++) {
  for (j = 0; j < K; j++) {
    diag2[i][j] = 0;
  }
}

for (i = 0; i < NDIAG; i++) {
  if (nofst[i] < 0) {
    nbase = -nofst[i];
    length = N - nbase;
    for (j = 0; j < length; j++) {
      diag2[i][j] = diag[i][nbase + j];
    }
  } else {
    nbase = nofst[i];
    length = N - nbase;
    for (j = 0; j < length; j++) {
      diag2[i][nbase + j] = diag[i][j];
    }
  }
}

numnz = 1;

for (j = 0; j < N; j++) {
  ntopcfg = 1;
  for (i = NDIAG - 1; i >= 0; i--) {
```

```
      if (ntopcfg == 1) {
        nfcnz[j] = numnz;
        ntopcfg = 0;
      }
      if (j + 1 < NBORDER && i + 1 > NOFFDIAG) {
        continue;
      } else {
        if (diag2[i][j] != 0.0) {
          ncol = (j + 1) - nofst[i];
          a[numnz - 1] = diag2[i][j];
          nrow[numnz - 1] = ncol;
          numnz++;
        }
      }
    }
  }

  nfcnz[N] = numnz;
  nz = numnz - 1;

  c_dm_vmvscc(a, nz, nrow, nfcnz, N, solex, b, wc, (int *)iwc, &icon);

/* INITIAL CALL WITH IORDER=1 */

  iordering = 0;
  ipledsm = 1;
  isclitermax = 10;
  isw = 1;
  epsz = 1.0e-16;
  nsizefactorl = 1;
  nsizefactoru = 1;
  nsizeindexl = 1;
  nsizeindexu = 1;
  thepsz = 1.0e-2;
  spepsz = 0.0;
  ipivot = 40;
  istatic = 0;
  irefine = 1;
  epsr = 0.0;
  itermax = 10;

  c_dm_vsrs(a, nz, nrow, nfcnz, N, ipledsm, mz, isclitermax, &iordering,
            nperm, isw, nrowsym, nfcnzsym, b, nassign, &nsupnum, nfcnzfactorl,
            &dummyfl, &nsizefactorl, nfcnzindexl, &ndummyil, &nsizeindexl,
            (int *)ndim, nfcnzfactoru, &dummyfu, &nsizefactoru, nfcnzindexu,
            &ndummyiu, &nsizeindexu, nposto, sclrow, sclcol, &epsz, &thepsz,
            ipivot, istatic, &spepsz, nfcnzpivot, npivotp, npivotq, irefine,
            epsr, itermax, &iter, w, iw1, iw2, &icon);

  printf(" ICON= %d  NSIZEFACTORL= %d  NSIZEFACTORU= %d NSIZEINDEXL= %d",
         icon, nsizefactorl, nsizefactoru, nsizeindexl);
  printf(" NSIZEINDEXU= %d NSUPNUM= %d\n", nsizeindexu, nsupnum);
```

```
  panelfactorl = (double *)malloc(nsizefactorl * sizeof(double));
  panelfactoru = (double *)malloc(nsizefactoru * sizeof(double));
  npanelindexl = (int *)malloc(nsizeindexl * sizeof(int));
  npanelindexu = (int *)malloc(nsizeindexu * sizeof(int));

  isw = 2;

  c_dm_vsrs(a, nz,nrow, nfcnz, N,ipledsm, mz, isclitermax, &iordering,
            nperm, isw, nrowsym, nfcnzsym, b, nassign, &nsupnum, nfcnzfactorl,
            panelfactorl, &nsizefactorl, nfcnzindexl, npanelindexl,
            &nsizeindexl, (int *)ndim, nfcnzfactoru, panelfactoru,
            &nsizefactoru, nfcnzindexu, npanelindexu, &nsizeindexu, nposto,
            sclrow, sclcol, &epsz, &thepsz, ipivot, istatic, &spepsz,
            nfcnzpivot, npivotp, npivotq, irefine, epsr, itermax, &iter, w,
            iw1, iw2, &icon);

  err = errnrm(solex, b, N);

  printf("     COMPUTED VALUES\n");
  printf("     X(1) = %18.15lf  X(N) = %18.15lf\n \n", b[0], b[N - 1]);
  printf("     ICON =  %d\n \n", icon);
  printf("     N = %6d\n \n", N);
  printf("     ERROR = %18.15lf\n", err);
  printf("     ITER= %d\n \n \n", iter);

  if (err < 1.0e-8 && icon == 0) {
    printf(" ********** OK **********\n");
  } else {
    printf(" ********** NG **********\n");
  }

  free(panelfactorl);
  free(panelfactoru);
  free(npanelindexl);
  free(npanelindexu);

  return(0);
}

/* ======================================
       INITIALIZE COEFFICIENT MATRIX
   ====================================== */
void init_mat_diag(double va1, double va2, double va3, double vc, double *d_l,
                   int *offset, int nx, int ny, int nz, double xl, double yl,
                   double zl, int ndiag, int len, int ndivp) {

  if (ndiag < 1) {
    printf("FUNCTION INIT_MAT_DIAG:\n");
    printf(" NDIAG SHOULD BE GREATER THAN OR EQUAL TO 1\n");
    return;
  }

#pragma omp parallel default(shared)
```

```
{
  int i, j, l, ndiag_loc, nxy, js, k0, j0, i0;
  double hx, hy, hz, hx2, hy2, hz2;

/* NDIAG CANNOT BE GREATER THAN 7 */
  ndiag_loc = ndiag;
  if (ndiag > 7) ndiag_loc = 7;

/* INITIAL SETTING */
  hx = xl / (nx + 1);
  hy = yl / (ny + 1);
  hz = zl / (nz + 1);

#pragma omp for
  for (i = 0; i < ndivp; i++) {
    for (j = 0; j < ndiag; j++) {
      d_l[(j * ndivp) + i] = 0.0;
    }
  }

  nxy = nx * ny;

/* OFFSET SETTING */
#pragma omp single
  {
    l = 0;
    if (ndiag_loc >= 7) {
      offset[l] = -nxy;
      l++;
    }
    if (ndiag_loc >= 5) {
      offset[l] = -nx;
      l++;
    }
    if (ndiag_loc >= 3) {
      offset[l] = -1;
      l++;
    }
    offset[l] = 0;
    l++;
    if (ndiag_loc >= 2) {
      offset[l] = 1;
      l++;
    }
    if (ndiag_loc >= 4) {
      offset[l] = nx;
      l++;
    }
    if (ndiag_loc >= 6) {
      offset[l] = nxy;
    }
  }
```

```
/* MAIN LOOP */
#pragma omp for
  for (j = 0; j < len; j++) {
    js = j + 1;

    k0 = (js -1) / nxy + 1;
    if (k0 > nz) {
      printf("ERROR; K0.GH.NZ \n");
      goto label_100;
    }
    j0 = (js - 1 - nxy * (k0 - 1)) / nx + 1;
    i0 = js - nxy * (k0 - 1) - nx * (j0 - 1);
    l = 0;

    if (ndiag_loc >= 7) {
      if (k0 > 1) d_l[(l * ndivp) + j] = -(1.0 / hz + 0.5 * va3) / hz;
      l++;
    }
    if (ndiag_loc >= 5) {
      if (j0 > 1) d_l[(l * ndivp) + j] = -(1.0 / hy + 0.5 * va2) / hy;
      l++;
    }
    if (ndiag_loc >= 3) {
      if (i0 > 1) d_l[(l * ndivp) + j] = -(1.0 / hx + 0.5 * va1) / hx;
      l++;
    }
    hx2 = hx * hx;
    hy2 = hy * hy;
    hz2 = hz * hz;
    d_l[(l * ndivp) + j] = 2.0 / hx2 + vc;
    if (ndiag_loc >= 5) {
      d_l[(l * ndivp) + j] += 2.0 / hy2;
      if (ndiag_loc >= 7) {
        d_l[(l * ndivp) + j] += 2.0 / hz2;
      }
    }
    l++;
    if (ndiag_loc >= 2) {
      if (i0 < nx) d_l[(l * ndivp) + j] = -(1.0 / hx - 0.5 * va1) / hx;
      l++;
    }
    if (ndiag_loc >= 4) {
      if (j0 < ny) d_l[(l * ndivp) + j] = -(1.0 / hy - 0.5 * va2) / hy;
      l++;
    }
    if (ndiag_loc >= 6) {
      if (k0 < nz) d_l[(l * ndivp) + j] = -(1.0 / hz - 0.5 * va3) / hz;
    }
label_100: ;
  }

}
```

```
  return;
}

/* ======================================
  * SOLUTE ERROR
  * | X1 - X2 |
   ====================================== */
double errnrm(double *x1, double *x2, int len) {
  double rtc, s, ss;
  int i;

  s = 0.0;
  for (i = 0; i < len; i++) {
    ss = x1[i] - x2[i];
    s = s + ss * ss;
  }

  rtc = sqrt(s);
  return(rtc);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VSRS の項目及び[2], [13], [17], [19], [22], [23], [48], [57], [63], [68], [69]を参照してください.

c_dm_vssps

正値対称スパース行列の連立 1 次方程式 (Left-looking な LDL^T 分解法)

`ierr = c_dm_vssps(a, nz, nrow, nfcnz, n,`
`iordering, nperm, isw, &epsz, b,`
`nassign, &nsupnum, nfcnzfactor,`
`panelfactor, &nsizefactor,`
`nfcnzindex, npanelindex,`
`&nsizeindex, ndim, nposto, w, iw1,`
`iw2, iw3, &icon);`

1. 機能

*n*次の正値対称スパース行列**A**を変形コレスキー分解法によりLDL^T分解して解きます.

$$\mathbf{Ax=b}$$

正値対称スパース行列**A**は以下のように分解されます.

$$\mathbf{QPAP^TQ^T=LDL^T}$$

ただし, **P**はorderingによる行列要素の並び換えを示す置換行列, **Q**はpost orderによる行列要素の並び換えを示す置換行列を示します. **P**,**Q**は直交行列です.
L は単位下三角行列, **D** は対角行列です.

2. 引数

呼出し形式:

```
ierr = c_dm_vssps(a, nz, nrow, nfcnz, n, iordering, nperm, isw, &epsz, b, nassign,
          &nsupnum, nfcnzfactor, panelfactor, &nsizefactor, nfcnzindex,
          npanelindex, &nsizeindex, (int *)ndim, nposto, w, iw1, iw2, iw3, &icon)
```

引数の説明:

a	double a[nz]	入力	係数行列 **A** のスパースな正値対称行列の下三角行列部分 $\{a_{ij}\|i\geq j\}$ を圧縮列格納法で a[*i*], *i*=0,…,nz-1 に格納します. 圧縮列格納法については, c_dm_vmvscc の図 c_dm_vmvscc-1 参照.
nz	int	入力	係数行列 **A** のスパースな正値対称行列の下三角行列部分にある非零要素の総数.
nrow	int nrow[nz]	入力	圧縮列格納法で使用される行指標で **A** に格納される要素が何番目の行ベクトルに属するかを示します.
nfcnz	int nfcnz[n+1]	入力	行列 **A** の各列の非零要素を列方向に圧縮して順次配列 a に格納するとき,対応する列の最初の非零要素が格納される位置を表します. nfcnz[n]=nz+1.
n	int	入力	行列 **A** の次数 *n*.
iordering	int	入力	ordering を表す置換行列 **P** で $\mathbf{PAP^T}$ と変換した行列を LDL^T 分解するかを指定します. 置換行列は直交行列です. 1 のとき: $\mathbf{PAP^T}$ と変換したものを LDL^T 分解します.

			1 以外のとき: 行列 **A** をそのまま LDL^T 分解します.
nperm	int nperm[n]	入力	iordering=1 のとき使用する置換行列をベクトルで指定します. (“使用上の注意” a)参照)
isw	int	入力	呼び出しに関する制御情報を示します. 1 初回の呼び出し. 2 1 回目の呼び出しでは panelfactor または npanelindex の大きさが不足していた. icon=31000 で終了した.このとき, nsizefactor, または nsizeindex に返却された必要な大きさで panelfactor または npanelindex を確保し直して再度呼び出しします. さらに a, nz, nrow, nfcnz, n, iordering, nperm, nassign, nsupnum, nfcnzfactor, nfcnzindex, npanelindex, nposto, ndim, w, iw1, iw2, iw3 に格納されている値を変更してはなりません. 3 同じ非零パターンを持つ次数の同じ行列で,行列要素の値が異なる行列に対して symbolic decomposition の解析結果や必要な大きさが同じになる配列 panelfactor, npanelindex を再利用して LDL^T 分解を行って方程式を解くことを指定します.行列の値を配列要素に格納し直して呼び出します. このとき, nrow の値を変えずに配列 a に行列要素の値を格納し直すか,別の配列 c に格納し引数 a として受け渡さなければなりません. さらに nz, nrow, nfcnz, n, iordering, nperm, nassign, nsupnum, nfcnzfactor, nsizefactor, nfcnzindex, npanelindex, nsizeindex, nposto, ndim, w, iw1, iw2, iw3 に格納されている値を変更してはなりません. 4 先立つ呼び出しで LDL^T 分解された同じ係数行列に対する連立 1 次方程式の右辺ベクトル **b** を変えて再度解くことを指定します.このとき先立つ呼び出しで LDL^T 分解した結果を使います. さらに n, iordering, nperm, nassign, nsupnum, nfcnzfactor, nsizefactor, nfcnzindex, npanelindex, nsizeindex, nposto, ndim, iw3 に格納されている値を変更してはなりません.
epsz	double	入力 出力	ピボットの相対判定値 (≥0.0) 0.0 のときは標準値が設定されます. (“使用上の注意” b)参照)
b	double b[n]	入力 出力	**Ax=b** の右辺定数ベクトル. 解ベクトル.
nassign	int nassign[n]	出力	各 supernode は複数の列ベクトルからなり,分解結果に関して共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します.この panel を

			panelfactor の 1 次元部分配列として順番に割り付けたとき何番目になるかを示します. j=nassign[i-1]のとき, *i* 番目の supernode を *j* 番目に割り付けたことを示します.
		入力	isw≠1 のとき,初回呼び出しの値を再利用します. 分解結果の格納方法については図 c_dm_vssps-1 を参照. ("使用上の注意" c)参照)
nsupnum	int	出力	supernode の総数.
		入力	isw≠1 のとき,初回呼び出しの値を再利用します. (≤n)
nfcnzfactor	long long int nfcnzfactor [n+1]	出力	各 supernode は複数の列ベクトルからなり,分解結果に関して共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します.この panel を panelfactor の 1 次元部分配列として順番に割り付けたとき, *i* 番目の panel の先頭要素 panel[0][0]が 1 次元配列の panelfactor の何番目の要素になるかを示します. 分解結果の格納方法については図 c_dm_vssps-1 を参照.
		入力	isw≠1 のとき,初回呼び出しの値を再利用します.
panelfactor	double panelfactor [nsizefactor]	出力	各 supernode は複数の列ベクトルからなり,分解結果に関して共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します. panel を順番に割り付けます. *i* 番目の supernode に対応する panel が何番目の位置に割り当てられるかは j=nassign[i-1]から分かります. その先頭位置は nfcnzfactor[j-1] に格納されています. panel ごとに分解結果が格納されます. *i* 番目に割付けられた panel の大きさは ndim[i-1][1]×ndim[i-1][0]の 2 次元配列と見なせます. *i* 番目の panel の panel[t-1][s-1], s>t, s=1,…, ndim[i-1][0], t=1,…,ndim[i-1][1]に単位下三角行列 **L** の対角要素を除いた対応部分が転置した形で格納されます.対角部分 panel[t-1][t-1]には対角行列 **D** の対応部分が格納されます. panelfactor[nsizefactor-1]なる 1 次元配列. 分解結果の格納方法については図 c_dm_vssps-1 を参照してください. ("使用上の注意" c)参照)
nsizefactor	long long int	入力	panelfactor の大きさを示す.
		出力	panelfactor の大きさとして必要な大きさが返却されます. ("使用上の注意" c)参照)
nfcnzindex	long long int nfcnzindex [n+1]	出力	各 supernode は複数の列ベクトルからなり,分解結果に関して共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します.この行指標ベクトルを npanelindex に順番に割り付けたとき *i* 番目の行指標ベクトルの先頭要素が 1 次元配列である npanelindex の何番目の要素になるかを示します.
		入力	isw≠1 のとき,初回呼び出しの値を再利用します. 分解結果の格納方法については図 c_dm_vssps-1 を参照.

npanelindex	int npanelindex [nsizeindex]	出力	各 supernode は複数の列ベクトルからなり,分解結果に関して共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します.この行指標ベクトルを npanelindex に順番に割り付けます. *i* 番目の supernode に対応する panel の行の指標ベクトルが何番目に割り当てられるかは j=nassign[i-1]から分かります.その先頭位置は nfcnzindex[j-1]に格納されています. panel ごとの行の指標ベクトルが格納されます. この行指標は行列 **A** を post order で並び換えた行列 $\mathbf{QAQ}^{\mathrm{T}}$ での行の番号です. 分解結果の格納方法については図 c_dm_vssps-1 を参照. (“使用上の注意” c)参照)
nsizeindex	long long int	入力 出力	npanelindex の大きさを示す. 必要な大きさが返却されます. (“使用上の注意” c)参照)
ndim	int ndim[n][2]	出力 入力	ndim[i-1][0]および ndim[i-1][1] は *i* 番目に割り付けられた panel の 1 次元目と 2 次元目の大きさを示します. isw≠1 のとき,初回呼び出しの値を再利用します. 分解結果の格納方法については図 c_dm_vssps-1 を参照.
nposto	int nposto[n]	出力 入力	post order で *i* 番目の node が行列 **A** の何番目の列番号に対応するかを表す 1 次元ベクトル. isw≠1 のとき,初回呼び出しの値を再利用します. (“使用上の注意” d)参照)
w	double w[*Iwlen1*]	作業域 出力/入力	isw=1, 2, 3 で続けて呼び出すとき,呼び出しの間で必要なデータを受け渡すために使われます.呼び出しの間で値を変更してはいけません. 1) iordering=1 のとき *Iwlen1*=nz 2) iordering≠1 のとき *Iwlen1*=1
iw1	int iw1[*Iwlen2*]	作業域 出力/入力	isw=1, 2, 3 で続けて呼び出すとき,呼び出しの間で必要なデータを受け渡すために使われます.呼び出しの間で値を変更してはいけません. 1) iordering=1 のとき *Iwlen2*=nz+n+1 2) iordering≠1 のとき *Iwlen2*=1
iw2	int iw2[nz+n+1]	作業域 出力/入力	isw=1, 2, 3 で続けて呼び出すとき,呼び出しの間で必要なデータを受け渡すために使われます.呼び出しの間で値を変更してはいけません.
iw3	int iw3[n*35+35]	作業域 出力/入力	isw =1, 2, 3, 4 で続けて呼び出すとき,呼び出しの間で必要なデータを受け渡すために使われます.呼び出しの間で値を変更してはいけません.
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.
20000	ピボットが相対的にゼロになった. 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 • nz < 0 • nfcnz[n] ≠nz+1 • nsizefactor<1 • nsizeindex<1 • epsz<0.0 • isw<0 • isw>4	
30100	nperm に指定した置換行列が正しくない.	
30200	nrow[j-1]に格納された *i* 列目の行指標 k が k<i または k>n.	
30300	*i* 列目の行指標の数 nfcnz[i]-nfcnz[i-1]>n-i+1	
30400	対角要素が格納されていない列がある.	
31000	panelfactor の大きさ nsizefactor または npanelindex の大きさ nsizeindex が小さすぎる.	nsizefactor または nsizeindex で指定された大きさで panelfactor または npanelindex を割り付けて isw=2 として再度呼び出す.

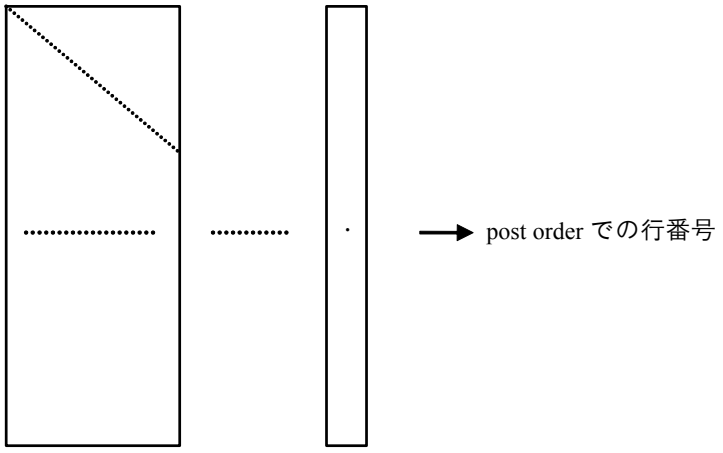


図 c_dm_vssps-1 分解結果の格納概念図

`j = nassign[i-1]` → *i* 番目の supernode は *j* 番目に格納されます.
`p = nfcnzfactor[j-1]` → *j* 番目の `panel` は `panelfactor` の *p* 番目の要素から `ndim[j-1][1]×ndim[j-1][0]`の長さを占めます.
`q = nfcnzindex[j-1]` → *j* 番目の `panel` の行指標を表すベクトルは `npanelindex` の *q* 番目の要素から `ndim[j-1][0]`の長さを占めます.

`panel` は大きさ `ndim[j-1][1]×ndim[j-1][0]`の配列と見なせます.
`panel[t-1][s-1]` , `s` > `t` , `s=1` , … , `ndim[j-1][0]`,
`t=1` , … , `ndim[j-1][1]`
に分解結果の単位下三角行列 **L** の対角要素を除いた部分が転置した形で格納されます.
`panel[t-1][t-1]`に対角行列 **D** の対応部分が格納されます.
行指標の値は行列 **A** の列番号をもつ node を post order で並べ換えた $\mathbf{QAQ}^{\mathrm{T}}$ での列番号を表します.

3. 使用上の注意

a) `nperm` について

直交行列である置換行列 **P** の要素 $\mathbf{p_{ij}=1}$ のとき, `nperm[i-1]=j` と表現します.
逆は以下のようにして求めることができます.

```
for(i=1; i<=n; i++){
  j=nperm[i-1];
  nperminv[j-1]=i;
}
```

b) `epsz`

ピボットの相対零判定値にある値を設定したとすると, この値は次の意味を持っています. すなわち, 変形コレスキー分解法による $\mathrm{LDL}^{\mathrm{T}}$ 分解の過程で, ピボットの絶対値が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見なし, `icon`=20000 として処理を打ち切ります. `epsz` の標準値は, 丸め誤差の単位を *u* としたとき, `epsz`=16*u* です.

なお, ピボットの絶対値が小さくなっても, 処理を続行させたい場合には, `epsz` に極小の値を与えれば良いのですが, 結果の精度は保証されません.

分解の途中でピボットが負となった場合, 係数行列は正値ではありません. このとき, icon=10000 として処理は続行します. ただし, ピボッティングを行っていないため, 計算誤差は大きい可能性があります.

c) nsizefactor と nsizeindex について

分解結果を格納する配列 panelfactor, npanelindex の必要な大きさは,事前には分かりません.十分大きな配列を割り当てるか,本ルーチンを呼び出して symbolic decomposition を行った解析の結果を使って割り当てを行うことができます.

例えば,大きさ 1 の 1 次元配列などを割付けます.そしてその大きさ 1 などの小さな値を nsizefactor, nsizeindex に指定して, isw=1 で呼び出します. symbolic decomposition を行い, icon=31000 で終了し, nsizefactor と nsizeindex に必要な大きさが返却されます.必要な大きさの配列を割付け直して, isw=2 で呼び出すことで, symbolic decomposition 以降の処理を続けることができます.

d) nposto について

列番号に対応するノードを考えます.これを post order で順番を並び換えたときの番号の対応関係が nposto に格納されています. post order の *i* 番目のノードが並び換える前の何番目のノードに対応するかを表します. j=nposto[i-1]は *j* 番目であることを表します.

a)と同様にこれは直交行列である置換行列 **Q** を表し,行列 **A** を $\mathbf{QAQ}^{\mathrm{T}}$ と並び換えることに相当します. 逆変換 $\mathbf{Q}^{\mathrm{T}}$は以下のようにして求めることができます.

```
for(i=1; i<=n; i++){
   j=nposto[i-1];
   npostoinv[j-1]=i;
}
```

4. 使用例

連立 1 次方程式 **Ax** = **f** を解きます. 行列 **A** は境界条件として立方体の境界で 0 となる楕円型の偏微分方程式に有限差分法を適用して生成されます.

$$-\Delta u + a\nabla u + cu = f$$

ここで $a = (a_1, a_2, a_3)$, a_1, a_2, a_3 および c はゼロで,ラプラシアンになり行列 **A** は正値対称です.行列 **A** は関数 init_mat_diag によって生成されます. これを圧縮列格納法に変換します. (並列実行を行うスレッド数は環境変数(OMP_NUM_THREADS)などで指定できます. 例えば, 4 プロセッサーのシステムで 4 つのスレッドで並列実行するときは, OMP_NUM_THREADS を 4 に設定して実行します.)

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

#define NORD   (39)
#define NX     (NORD)
#define NY     (NORD)
#define NZ     (NORD)
#define N      (NX*NY*NZ)
#define K      (N+1)
#define NDIAG  (7)
#define NDIAGH (4)

MAIN__()
{
  int    ierr, icon, iguss, iter, itmax;
  int    nord, n, l, i, j, k;
  int    nx, ny, nz, nnz, nnzc;
  int    length, nbase, ndiag, ntopcfgc;
  int    numnz, numnzc, nsupnum, ntopcfg, ncol;
  int    iordering, isw;
```

```
int    *npanelindex;
int    ndummyi;
int    nofst[NDIAG];
int    nrow[NDIAG*K];
int    nrowc[NDIAG*K];
int    nfcnz[N+1];
int    nfcnzc[N+1];
int    nperm[N];
int    nassign[N];
int    nposto[N];
int    ndim[N][2];
int    iw1[N*NDIAGH+N+1];
int    iw2[N*NDIAGH+N+1];
int    iw3[N*35+35];
int    iwc[NDIAG*K][2];

double err, epsz;
double t0, t1, t2;
double va1, va2, va3, vc;
double xl, yl, zl;
double dummyf;
double *panelfactor;
double diag[NDIAG][K];
double diag2[NDIAG][K];
double a[N*NDIAGH];
double b[N];
double c[NDIAG*K];
double w[N*NDIAGH];
double wc[NDIAG*K];
double x[N];
double solex[N];

long long int nsizefactor;
long long int nsizeindex;
long long int nfcnzfactor[N+1];
long long int nfcnzindex[N+1];

void init_mat_diag(double va1, double va2, double va3, double vc,
            double d_l[], int offset[], int nx, int ny, int nz,
            double xl, double yl,double zl, int ndiag, int len, int ndivp);

double errnrm(double *x1, double *x2, int len);

nord=NORD, nx=NX, ny=NY, nz=NZ, n=N, k=K, ndiag=NDIAG;

printf("     LEFT-LOOKING MODIFIED CHOLESKY METHOD\n");
printf("     FOR SPARSE POSITIVE DEFINITE MATRICES\n");
printf("     IN COMPRESSED COLUMN STORAGE\n");
printf("\n");

for (i=1; i<=n; i++){
  solex[i-1]=1.0;
}
printf("     EXPECTED SOLUTIONS\n");
printf("     X(1) = %.15lf  X(N) = %.15lf\n", solex[0], solex[n-1]);
printf("\n");

va1 = 0.0;
va2 = 0.0;
va3 = 0.0;
vc = 0.0;
xl = 1.0;
yl = 1.0;
zl = 1.0;
init_mat_diag(va1, va2, va3, vc, (double*)diag, (int*)nofst,
                    nx, ny, nz, xl, yl, zl, ndiag, n, k);

for (i=1; i<=ndiag; i++){
  if (nofst[i-1] < 0){
    nbase=-nofst[i-1];
    length=n-nbase;
    for (j=1; j<=length; j++){
      diag2[i-1][j-1]=diag[i-1][nbase+j-1];
    }
  }
  else{
    nbase=nofst[i-1];
```

```
            length=n-nbase;
            for (j=nbase+1; j<=n; j++){
              diag2[i-1][j-1]=diag[i-1][j-nbase-1];
            }
          }
        }

        numnzc=1;
        numnz=1;
        for (j=1; j<=n; j++){
          ntopcfgc = 1;
          ntopcfg = 1;
          for (i=ndiag; i>=1; i--){
            if (diag2[i-1][j-1]!=0.0){
              ncol=j-nofst[i-1];
              c[numnzc-1]=diag2[i-1][j-1];
              nrowc[numnzc-1]=ncol;
              if (ncol>=j){
                a[numnz-1]=diag2[i-1][j-1];
                nrow[numnz-1]=ncol;
              }
              if (ntopcfgc==1){
                nfcnzc[j-1]=numnzc;
                ntopcfgc=0;
              }
              if (ntopcfg==1){
                nfcnz[j-1]=numnz;
                ntopcfg=0;
              }
              if (ncol>=j){
                numnz=numnz+1;
              }
              numnzc=numnzc+1;
            }
          }
        }


        nfcnzc[n]=numnzc;
        nnzc=numnzc-1;
        nfcnz[n]=numnz;
        nnz=numnz-1;

        ierr=c_dm_vmvscc(c, nnzc, nrowc, nfcnzc, n, solex, b, wc, (int*)iwc, &icon);

        for(i=1; i<=n; i++){
           x[i-1]=b[i-1];
        }
        iordering=0;
        isw=1;
        epsz=0;
        nsizefactor=1;
        nsizeindex=1;

        ierr=c_dm_vssps(a, nnz, nrow, nfcnz, n, iordering, nperm, isw, &epsz, x, nassign,
&nsupnum, nfcnzfactor, &dummyf, &nsizefactor, nfcnzindex, &ndummyi, &nsizeindex,
(int*)ndim, nposto, w, iw1, iw2, iw3, &icon);

        printf("\n");
        printf("     ICON = %d  NSIZEFACTOR = %lld NSIZEINDEX = %lld\n", icon,
nsizefactor, nsizeindex);
        printf("\n");

        panelfactor = (double *)malloc(sizeof(double)*nsizefactor);
        npanelindex = (int *)malloc(sizeof(int)*nsizeindex);
        isw=2;

        ierr=c_dm_vssps(a, nnz, nrow, nfcnz, n, iordering, nperm, isw, &epsz, x, nassign,
&nsupnum, nfcnzfactor, panelfactor, &nsizefactor, nfcnzindex, npanelindex, &nsizeindex,
(int*)ndim, nposto, w, iw1, iw2, iw3, &icon);

        err = errnrm(solex,x,n);

        printf("     COMPUTED VALUES\n");
        printf("     X(1) = %.15lf  X(N) = %.15f\n", x[0], x[n-1]);
        printf("\n");
        printf("     ICON = %d\n", icon);
```

```
  printf("\n");
  printf("     N = %d  :: NX = %d  NY = %d  NZ = %d\n",n,nx,ny,nz);
  printf("\n");
  printf("     ERROR = %.15e\n",err);
  printf("\n");
  printf("\n");
  if (err<(1.0e-8) && icon==0){
    printf("     ********** OK **********\n");
  }
  else{
    printf("     ********** NG **********\n");
  }
    free(panelfactor);
    free(npanelindex);
    return 0;
}

void init_mat_diag(double va1, double va2, double va3, double vc,
         double d_l[], int offset[], int nx, int ny, int nz,
         double xl, double yl, double zl, int ndiag, int len, int ndivp)
{
  int i, l, j;
  int length, numnz, js;
  int i0, j0, k0;
  int ndiag_loc;
  int nxy;

  double hx, hy, hz;
  double x1, x2;
  double base;
  double ret, remark;

  if (ndiag<1){
    printf("FUNCTION INIT_MAT_DIAG:\n");
    printf("NDIAG SHOULD BE GREATER THAN OR EQUAL TO 1\n");
    return;
  }
  ndiag_loc = ndiag;
  if (ndiag>7){
    ndiag_loc=7;
  }

  hx = xl / (nx + 1);
  hy = yl / (ny + 1);
  hz = zl / (nz + 1);

  for (i=1; i<=ndivp; i++){
    for (j=1; j<=ndiag; j++){
      d_l[i-1+(j-1)*ndivp]= 0.;
    }
  }

  nxy = nx * ny;
  l = 1;
  if (ndiag_loc >= 7) {
    offset[l-1] = -nxy;
    ++l;
  }
  if (ndiag_loc >= 5) {
    offset[l-1] = -nx;
    ++l;
  }
  if (ndiag_loc >= 3) {
    offset[l-1] = -1;
    ++l;
  }
  offset[l-1] = 0;
  ++l;
  if (ndiag_loc >= 2) {
    offset[l-1] = 1;
    ++l;
  }
  if (ndiag_loc >= 4) {
    offset[l-1] = nx;
    ++l;
  }
  if (ndiag_loc >= 6) {
```

```
    offset[l-1] = nxy;
  }

  for (j = 1; j <= len; ++j) {
    js=j;
    k0 = (js - 1) / nxy + 1;
    if (k0 > nz) {
      printf("ERROR; K0.GH.NZ\n");
      return;
    }
    j0 = (js - 1 - nxy * (k0 - 1)) / nx + 1;
    i0 = js - nxy * (k0 - 1) - nx * (j0 - 1);

    l = 1;
    if (ndiag_loc >= 7) {
      if (k0 > 1) {
        d_l[j-1+(l-1)*ndivp] = -(1.0/hz+va3*0.5)/hz;
      }
      ++l;
    }

    if (ndiag_loc >= 5) {
      if (j0 > 1) {
        d_l[j-1+(l-1)*ndivp] = -(1.0/hy+va2*0.5)/hy;
      }
      ++l;
    }

    if (ndiag_loc >= 3) {
      if (i0 > 1) {
        d_l[j-1+(l-1)*ndivp] = -(1.0/hx+va1*0.5)/hx;
      }
      ++l;
    }

    d_l[j-1+(l-1)*ndivp] = 2.0/(hx*hx)+vc;
    if (ndiag_loc >= 5) {
      d_l[j-1+(l-1)*ndivp] += 2.0/(hy*hy);
      if (ndiag_loc >= 7) {
        d_l[j-1+(l-1)*ndivp] += 2.0/(hz*hz);
      }
    }
    ++l;
    if (ndiag_loc >= 2) {
      if (i0 < nx) {
        d_l[j-1+(l-1)*ndivp] = -(1.0/hx-va1*0.5)/hx;
      }
      ++l;
    }

    if (ndiag_loc >= 4) {
      if (j0 < ny) {
        d_l[j-1+(l-1)*ndivp] = -(1.0/hy-va2*0.5)/hy;
      }
      ++l;
    }

    if (ndiag_loc >= 6) {
      if (k0 < nz) {
        d_l[j-1+(l-1)*ndivp] = -(1.0/hz-va3*0.5)/hz;
      }
    }
  }
  return;
}

double errnrm(double *x1, double *x2, int len)
{
  double ret_val;

  int i;
  double s, ss;

  s = 0.;
  for (i = 1; i <= len; ++i) {
    ss = x1[i-1] - x2[i-1];
    s += ss * ss;
```

```
    }
    ret_val = sqrt(s);
    return ret_val;
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VSSPS の項目及び[19]を参照してください.

c_dm_vssslu

構造的に対称な実スパース行列の LU 分解
ierr = c_dm_vssslu(a, nz, nrow, nfcnz, n, isclitermax, iordering, nperm, isw, nassign, &nsupnum, nfcnzfactorl, panelfactorl, &nsizefactorl, nfcnzindexl, npanelindexl, &nsizeindex, ndim, nfcnzfactoru, panelfactoru, &nsizefactoru, nfcnzindexu, npanelindexu nposto, sclrow, sclcol, &epsz, &thepsz, ipivot, istatic, &spepsz, w, iw, &icon);

1. 機能

$n \times n$ の構造的に対称な実スパース行列 **A** に行および列の均衡化を行うスケーリングを行います. スーパーノードの対角ブロック内で指定した Pivot をとり, LU 分解します.

(構造的に対称な実スパース行列の非零要素はそれと対称な位置に要素を持ち、その値は必ずしも等しくはない行列です.)

構造的に対称な実スパース行列 **A** は以下のように変換できます.

$$\mathbf{A_1} = \mathbf{D_r A D_c}$$

ここで $\mathbf{D_r}$ は行のスケーリングを行う対角行列, $\mathbf{D_c}$ は列のスケーリングを行う対角行列です.

$$\mathbf{A_2} = \mathbf{Q P A_1 P^T Q^T}$$

$\mathbf{A_2}$ は, スーパーノードの対角ブロックで指定された Pivot を行い, 行および列の入れ換えを行い LU 分解されます.

ただし, **P** は, **A** の非ゼロパターンに対して求めた ordering による行列要素の並び換えを示す置換行列で, **Q** は **SYM** に対する post order による行列要素の並び換えを示す置換行列を示します. **P**, **Q** は直交行列です.

LU 分解の結果の行列 **L** と **U** の非零要素のパターンはそれぞれに対して対称になります.
L は下三角行列, **U** は単位上三角行列です.

Pivot 処理で, 閾値 thepsz より絶対値が大きな Pivot を見つけることが出来なかったとき, 対角ブロック内の絶対値が最大の要素を Pivot の候補とします. この値が小さ過ぎるときに Static Pivot を与えて近似的に LU 分解を続けることができます.

2. 引数

呼出し形式:

```
ierr = c_dm_vssslu(a, nz, nrow, nfcnz, n, isclitermax, iordering,
            nperm, isw, nassign, &nsupnum, nfcnzfactorl, panelfactorl,
            &nsizefactorl, nfcnzindexl, npanelindexl, &nsizeindex,
```

```
(int *)ndim, nfcnzfactoru, panelfactoru, &nsizefactoru,
nfcnzindexu, npanelindexu, nposto, sclrow, sclcol,
&epsz, &thepsz, ipivot, istatic, &spepsz, w, iw, &icon);
```

引数の説明:

a	double a[nz]	入力	構造的に対称な実スパースな係数行列 **A** を圧縮列格納法で格納します. 圧縮列格納法については, 実スパース行列と実ベクトル(圧縮列格納法), c_dm_vmvscc の図 c_dm_vmvscc-1 を参照してください. (“使用上の注意” e)参照)
nz	int	入力	構造的に対称な実スパースな係数行列 **A** の格納する非零要素の総数.
nrow	int nrow[nz]	入力	圧縮列格納法で使用される行指標で a に格納される要素が何番目の行ベクトルに属するかを示します.
nfcnz	int nfcnz[n+1]	入力	行列 **A** の各列の格納する非零要素を列方向に圧縮して順次配列 a に格納するとき,対応する列の最初の格納する非零要素が格納される位置を表します. nfcnz[n] = nz + 1.
n	int	入力	行列 **A** の次数 n.
isclitermax	int	入力	係数行列のスケーリングを行う対角行列 $\mathbf{D}_r$ と $\mathbf{D}_c$ を反復して求める反復回数. isclitermax ≤ 0 のときスケーリングは行わすに, $\mathbf{D}_r$ および $\mathbf{D}_c$ は単位行列が設定されます. isclitermax ≥ 10 のときは 10 回を上限値とします.
iordering	int	入力	ordering を表す置換行列 **P** で $\mathbf{PAP}^T$ と変換した行列を LU 分解するかを指定します. 置換行列は直交行列です. 1 のとき, $\mathbf{PAP}^T$ と変換した行列を LU 分解します. 1 以外のとき,行列をそのまま LU 分解します.
nperm	int nperm[n]	入力	iordering = 1 のとき使用する置換行列をベクトルで指定します. (“使用上の注意” a)参照)
isw	int	入力	呼び出しに関する制御情報を示します. 1) 1 のとき 初回の呼び出し.Symbolic decomposition を行い, 必要な領域が割り当てられているか調べ計算を行います. 2) 2 のとき isw = 1 で呼び出したとき, panelfactorl, panelfactoru, npanelindexl または npanelindexu の大きさが不足して icon = 31000 で終了した処理を継続します. このとき, nsizefactorl, nsizefactoru,または nsizeindex に返却された必要な大きさで panelfactorl, panelfactoru, npanelindexl または npanelindexu を確保し直して指定しなおして再度呼び出します. これらの引数と実行を制御する引数 isw 以外の引数に格納されている値は変更してはいけません.
nassign	int nassign[n]	出力	各 supernode に対する L, U は圧縮して 2 次元の panel に格納します.

			この panel を panelfactorl および panelfactoru の 1 次元部分配列として順番に割り付けたとき何番目になるかを示します. これらの指標ベクトルも同じように npanelindexl, npanelindexu に割り付けます. j = nassign[i-1]のとき, *i* 番目の supernode を *j* 番目に割り付けたことを示します. 分解結果の格納方法については図 c_dm_vssslu-1 を参照してください.
		入力	isw ≠ 1 のとき, 初回呼び出しの値を再利用します.
nsupnum	int	出力	supernode の総数.
		入力	isw ≠ 1 のとき, 初回呼び出しの値を再利用します. (≤n)
nfcnzfactorl	long nfcnzfactorl[n+1]	出力	構造的に対称な実スパース行列の LU 分解の結果の行列 **L** および **U** はスーパーノードに対応しておのおの求めます. 各 supernode に対応する **L** の列ベクトルは, 共通の行指標ベクトルを持つように圧縮してブロック対角部分に **U** の対応部分を含めて 2 次元の panel に格納します. この panel を panelfactorl の 1 次元部分配列として順番に割り付けたとき, *i* 番目の panel の先頭要素 panel[0][0]が 1 次元配列の panelfactorl の何番目の要素になるかを示します. 結果の格納方法については図 c_dm_vssslu-1 を参照してください.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します.
panelfactorl	double panelfactorl [nsizefactorl]	出力	各 supernode の分解結果に対応する行列 **L** の複数の列ベクトルと行列 **U** のブロック対角部分について, 共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します. panel を順番に割り付けます. *i* 番目の supernode に対応する panel が何番目の位置に割り当てられるかは j = nassign[i-1]から分かります. その先頭位置は nfcnzfactorl[j-1]に格納されています. panel ごとに分解結果が格納されます. *i* 番目に割付けられた panel の大きさは ndim[i-1][0] × ndim[i-1][1]の 2 次元配列と見なせます. i 番目の panel の panel[t-1][s-1], s ≥ t, s = 1,…, ndim[i-1][0], t = 1,…, ndim[i-1][1]に下三角行列 **L** が格納されます. panel の panel[t-1][s-1], s < t, t = 1,…, ndim[i-1][1]には単位上三角行列 **U** の対角要素を除いたブロック対角部分が格納されます. 結果の格納方法については図 c_dm_vssslu-1 を参照してください. ("使用上の注意" c)参照)
nsizefactorl	long	入力	panelfactorl の大きさを示します.
		出力	panelfactorl の大きさとして必要な大きさが返却されます. ("使用上の注意" c)参照)
nfcnzindexl	long nfcnzindexl[n+1]	出力	各 supernode の分解結果に対応する行列 **L** の複数の列ベクトルと行列 **U** のブロック対角部分について, 共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します. この行指標ベクトルを npanelindexl に順番に割り付けたとき *i* 番目の行指標ベクトルの先頭要素が 1 次元配列である npanelindexl の何番目の要

			素になるかを示します. 結果の格納方法については図 c_dm_vssslu-1 を参照してください.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します.
npanelindexl	int npanelindexl [nsizeindex]	出力	各 supernode の分解結果に対応する行列 **L** の複数の列ベクトルと行列 **U** のブロック対角部分について, 共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します. この行指標ベクトルを npanelindexl に順番に割り付けます. *i* 番目の supernode に対応する panel の行の指標ベクトルが何番目に割り当てられるかは j = nassign[i-1]から分かります. その先頭位置は nfcnzindexl[j-1]に格納されています. この行指標は post order で並び換えたときの行の番号です. 結果の格納方法については図 c_dm_vssslu-1 を参照してください. ("使用上の注意" c)参照)
nsizeindex	long	入力	npanelindexl および npanelindexu の大きさを示す.
		出力	npanelindexl および npanelindexu の大きさとして必要な大きさが返却されます. ("使用上の注意" c)参照)
ndim	int ndim[n][2]	出力	ndim[i-1][0]および ndim[i-1][1]は行列 **L** に関して *i* 番目に割り付けられた panel の 1 次元目と 2 次元目の大きさを示します. ndim[i-1][0]- ndim[i-1][1] および ndim[i-1][1]は行列 **U** に関して割り付けられた panel を転置したときの 1 次元目と 2 次元目の大きさを示します. 結果の格納方法については図 c_dm_vssslu-1 を参照してください.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します.
nfcnzfactoru	long nfcnzfactoru[n+1]	出力	構造的に対称な実スパース行列の LU 分解の結果の行列 **U** に関しては, 各 supernode に対応する **U** の行ベクトルは, ブロック対角部分を除いて共通の列指標ベクトルを持つように圧縮し, 転置したものを 2 次元の panel に格納します. この panel を panelfactoru の 1 次元部分配列として順番に割り付けたとき, *i* 番目の panel の先頭要素 panel[0][0]が 1 次元配列の panelfactoru の何番目の要素になるかを示します. 結果の格納方法については図 c_dm_vssslu-1 を参照してください.
		入力	isw ≠ 1 のとき, 初回呼び出しの値を再利用します.
panelfactoru	double panelfactoru [nsizefactoru]	出力	各 supernode の分解結果に対応する行列 **U** のブロック対角部分を除いた複数の行ベクトルについて, 共通の列指標ベクトルを持つように圧縮し, 転置して 2 次元の panel に格納します. panel を順番に割り付けます. *i* 番目の supernode に対応する panel が何番目の位置に割り当てられるかは j = nassign[i-1]から分かります.その先頭位置は nfcnzfactoru[j-1]に格納されています. panel ごとに分解結果が格納されます.

			i 番目に割付けられた panel の大きさは{ndim[i-1][0] – ndim[i-1][1] } × ndim[i-1][1]の 2 次元配列と見なせます. *i* 番目の panel の panel[t-1][s-1], s = 1,…, ndim[i-1][0] – ndim[i-1][1], t = 1,…, ndim[i-1][1]に単位上三角行列 **U** のブロック対角部分を除いた部分が行ベクトルを圧縮し, 転置して格納します. 結果の格納方法については図 c_dm_vssslu-1 を参照してください. (“使用上の注意” c)参照)
nsizefactoru	long	入力	panelfactoru の大きさを示す.
		出力	panelfactoru の大きさとして必要な大きさが返却されます. (“使用上の注意” c)参照)
nfcnzindexu	long nfcnzindexu[n+1]	出力	各 supernode の分解結果に対応する行列 **U** は対角ブロック部分を除いて複数の行ベクトルについて，共通の列指標ベクトルを持つように圧縮し, 転置して 2 次元の panel に格納します. 対角ブロック部分も含んだこの列指標ベクトルを npanelindexu に順番に割り付けたとき *i* 番目の列指標ベクトルの先頭要素が 1 次元配列である npanelindexu の何番目の要素になるかを示します. 結果の格納方法については図 c_dm_vssslu-1 を参照してください.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します.
npanelindexu	int npanelindexu [nsizeindex]	出力	各 supernode の分解結果に対応する行列 **U** の対角ブロックを除いた部分を転置して圧縮して 2 次元の panel に格納します. この列指標ベクトルを対角ブロック部分を含めて npanelindexu に順番に割り付けます. *i* 番目の supernode に対応する panel の行の指標ベクトルが何番目に割り当てられるかは j = nassign[i-1]から分かります. その先頭位置は nfcnzindexu[j-1]に格納されています. この行指標は post order で並び換えたときの列の番号です. 結果の格納方法については図 c_dm_vssslu-1 を参照してください. (“使用上の注意” c)参照)
nposto	int nposto[n]	出力	post order で *i* 番目の node が行列 **A** の何番目の列番号に対応するかを表す 1 次元ベクトル.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します. (“使用上の注意” d)参照)
sclrow	double sclrow[n]	出力	スケーリング対角行列 $\mathbf{D}_r$. 対角要素が 1 次元配列に格納されます.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します.
sclcol	double sclcol[n]	出力	スケーリング対角行列 $\mathbf{D}_c$. 対角要素が 1 次元配列に格納されます.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します.
epsz	double	入力	分解過程で対角要素の大きさの絶対値の相対判定値
		出力	epsz ≤ 0.0 のときは標準値が設定されます. (“使用上の注意” b)参照)
thepsz	double	入力	ピボットの判定での閾値(threshold). これより大きな値

			はピボットとして採用します. 見つかればその時点で, ピボット検索は打ち切ります. 例えば 10^{-2} 程度.
		出力	thepsz ≤ 0.0 のときは 10^{-2} が設定されます. epsz ≥ thepsz > 0.0 のときは, epsz が設定されます.
ipivot	int	入力	スーパーノード内でピボットを行うか, 行う場合どのようなピボットを行うかを指定します. 例えば, 完全ピボットとして 40 を指定. ipivot < 10 : または ipivot ≥ 50: ピボットなし. 10 ≤ ipivot < 20 : 部分ピボット 20 ≤ ipivot < 30 : 対角ピボット 21 : スーパーノード内で対角ピボットがとれないとき, Rook ピボットに変更します. 22 : スーパーノード内で対角ピボットがとれないときは, Rook ピボットに, Rook ピボットが取れないときは完全ピボットに変更します. 30 ≤ ipivot < 40 :Rook ピボット 32 : スーパーノード内で Rook ピボットがとれないとき, 完全ピボットをとります. 40 ≤ ipivot < 50 : 完全ピボット
istatic	int	入力	Pivot を指定したとき, Static Pivot を行うかを示します. 1) = 1 のとき スーパーノード内で指定したピボットが spepsz より大きくないとき, ピボットとして copysign(spepsz, Pivot)で近似します. ピボットの値が 0.0 なら spepsz に近似します. このとき, 以下を設定しなければなりません. a) epsz は epsz の標準値以下にしなければなりません. b) isclitermax は 10 を設定しスケーリングを行う必要があります. c)thepsz ≥ spepsz でなければなりません. 2) ≠ 1 のとき Static Pivot は行いません.
spepsz	double	入力	istatic = 1 のとき, Statc Pivot として使う値. thepsz ≥ spepsz ≥ epsz でなければなりません.
		出力	spepsz < epsz のときは, 10^{-10} が設定されます.
w	double w[4*nz+6*n]	作業域	isw = 1, 2 で続けて呼び出すとき,呼び出しの間で必要なデータを受け渡すために使われます. 呼び出しの間で値を変更してはいけません.
iw	int iw[36*n+36+2*nz+3*(n+1)]	作業域	isw = 1, 2 で続けて呼び出すとき, 呼び出しの間で必要なデータを受け渡すために使われます. 呼び出しの間で値を変更してはいけません.
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	istatic = 1 のとき小さすぎるピボットを spepsz で置き換える Static Pivot を行った.	処理は続行する.
20000	ピボットが相対的に零になった.	処理を打ち切る.
20200	行および列の均衡化を行う対角行列の対角要素を求める過程で,元の行列 A の行もしくは列にゼロベクトルがあった. 行列が特異である可能性がある.	
30000	次のいずれかであった: • n < 1 • nz < 0 • nfcnz[n] ≠ nz + 1 • nsizefactorl < 1 • nsizefactoru < 1 • nsizeindex < 1 • isw < 1 • isw > 2	
30100	nperm に指定した置換行列が正しくない.	
30200	nrow[j-1]に格納された *i* 列目の行指標 k が k<1 または k>n	
30300	*i* 列目の行指標の数 nfcnz[i] – nfcnz[i-1] > n	
30500	istatic = 1 のとき満たすべき条件をみたしていない. epsz は標準値 16***u*** より大きかった. または isclitermax < 10 であった. または spepsz > thepsz であった.	
30700	行列 ***A*** が構造的に対称でない.	
31000	panelfactorl の大きさ nsizefactorl または npanelindexl および npanelindexu の大きさ nsizeindex または panelfactoru の大きさ nsizefactoru が小さすぎます.	nsizefactorl または nsizeindex または nsizefactoru で指定された 大きさで panelfactorl または npanelindexl および npanelindexu または panelfactoru を割り付けて isw = 2 として再度呼び出す.

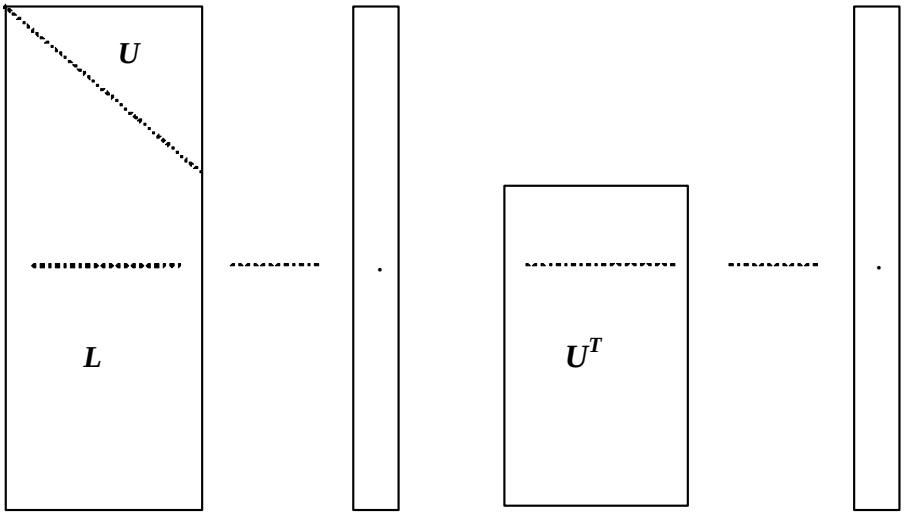


図 c_dm_vssslu-1 分解結果の格納概念図

j = nassign[i-1] → *i* 番目の supernode は *j* 番目に格納されます.

p = nfcnzfactorl[j-1] → *j* 番目の panel は panelfactorl の *p* 番目の要素から ndim[j-1][0] × ndim[j-1][1] の長さを占めます.

q = nfcnzindexl[j-1] → *j* 番目の panel の行指標を表すベクトルは npanelindexl の *q* 番目の要素から ndim[j-1][0] の長さを占めます.

panel は大きさ ndim[j-1][0] × ndim[j-1][1] の配列と見なせます.

panel[t-1][s-1] , s ≥ t, s = 1,…, ndim[j-1][0], t = 1,…, ndim[j-1][1] に分解結果の下三角行列 **L** が格納されます.

panel[t-1][s-1], s < t, s = 1,…, ndim[j-1][1], t = 1,…, ndim[j-1][1] に単位上三角行列 **U** の対角ブロック部分が対角要素を除き格納されます.

u = nfcnzfactoru[j-1] → *j* 番目の panel は panelfactoru の *u* 番目の要素から(ndim[j-1][0] – ndim[j-1][1]) × ndim[j-1][1] の長さを占めます.

v = nfcnzindexu[j-1] → *j* 番目の panel の列指標を表すベクトルは npanelindexu の *v* 番目の要素から ndim[j-1][0] の長さを占めます.

panel は大きさ(ndim[j-1][0] – ndim[j-1][1]) × ndim[j-1][1]の配列と見なせます.

panel[y-1][x-1], x = 1,…, ndim[j-1][0] - ndim[j-1][1], y = 1,…, ndim[j-1][1]

に分解結果の単位上三角行列 **U** の転置行列 $\mathbf{U}^{\mathrm{T}}$ の対角ブロック部分を除いた部分が格納されます.

指標の値は行列 **A** の列番号をもつ node を post order で並び換えた $\mathbf{QAQ}^{\mathrm{T}}$ での列番号を表します.

3. 使用上の注意

a)

直交行列である置換行列 **P** の要素 $\mathbf{p_{ij}} = \mathbf{1}$ のとき, nperm[i-1] = j と表現します.
逆は以下のようにして求めることができます.

```
for (i = 1; i <= n; i++) {
 j = nperm[i-1];
 nperminv[j-1] = i;
}
```

Fill-Reducing Ordering は METIS などを使って求めることができます.
詳細は“付録 参考文献一覧表”の[43], [44]を参照願います.

b)

分解過程で対角要素の大きさの絶対値の相対零判定値にある値を設定したとすると, この値は次の意味を持っています. すなわち, LU 分解の過程で, 対角要素の大きさの絶対値がその値より小さくなった場合に, その値を相対的に零と見なし, icon = 20000 として処理を打ち切ります. epsz の標準値は, 丸め誤差の単位を u としたとき, epsz = 16u です.
なお, 対角要素の大きさの絶対値が小さくなっても, 処理を続行させたい場合には, epsz に極小の値を与えれば良いのですが, 結果の精度は保証されません.
Static Pivot を指定した場合, 対角要素が spepsz より小さいとき, spepsz で近似して LU 分解を行います.

c)

分解結果を格納する配列 panelfactorl, npanelindexl, panelfactoru, npanelindexu の必要な大きさは,事前には分かりません. 十分大きな配列を割り当てるか, 本関数を呼び出して symbolic decomposition を行った解析の結果を使って割り当てを行うことができます.
例えば, 大きさ 1 の 1 次元配列などを割付けます. そしてその大きさ 1 などの小さな値を nsizefactorl, nsizeindex, nsizefactoru に指定して, isw = 1 で呼び出します.
symbolic decomposition を行い, icon = 31000 で終了し, nsizefactorl, nsizeindex, nsizefactoru に必要な大きさが返却されます. 必要な大きさの配列を割付け直して引数に指定して, isw = 2 で呼び出すことで, symbolic decomposition 以降の処理を続けることができます. 使用例を参照願います.
LU 分解の結果を利用して c_dm_vssslux を利用して連立 1 次方程式を解く上での注意は“使用上の注意” e)参照.

d)

列番号に対応するノードを考えます. これを post order で順番を並び換えたときの番号の対応関係が nposto に格納されています. post order の i 番目のノードが並び換える前の何番目のノードに対応するかを表します. j = nposto[i-1]は j 番目であることを表します.
a)同様にこれは直交行列である置換行列 **Q** を表し,行列 **A** を $\mathbf{QAQ^T}$ と並び換えることに相当します.
逆変換 $\mathbf{Q^T}$ は以下のようにして求めることができます.

```
for (i = 1; i <= n; i++) {
 j = nposto[i-1];
 npostoinv[j-1] = i;
}
```

e)

本関数で得られた LU 分解の結果を使い, c_dm_vssslux を引き続いて呼びだすことで連立 1 次方程式 $\mathbf{Ax} = \mathbf{b}$ を解くことができます.
このとき, c_dm_vssslu で利用した以下の引数を指定します. 使用例を参照願います.
a, nz, nrow, nfcnz, n,
iordering, nperm,

nassign, nsupnum,
nfcnzfactorl, panelfactorl,
nsizefactorl, nfcnzindexl, npanelindexl,
nsizeindex, ndim,
nfcnzfactoru, panelfactoru, nsizefactoru,
nfcnzindexu, npanelindexu, nposto,
sclrow, sclcol,
iw

4. 使用例

連立 1 次方程式 **Ax** = **f** を解きます.
行列は境界条件として立方体の境界で 0 となる楕円型の偏微分方程式に有限差分法を適用して生成されたものを利用します.

$$-\Delta u + a\nabla u + cu = f$$

ここで $a = (a_1, a_2, a_3)$.
行列は関数 init_mat_diag によって生成されます.
対角形式格納法で生成し,圧縮列格納法で格納します. 結果生成された構造的に対称な行列 **A** に関する連立 1 次方程式を解きます.
(並列実行を行うスレッド数は環境変数(OMP_NUM_THREADS)などで指定できます.
例えば, 4 プロセッサーのシステムで 4 つのスレッドで並列実行するときは, OMP_NUM_THREADS を 4 に設定して実行します.)

```
/* **EXAMPLE** */
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <malloc.h>
#include <omp.h>
#include "cssl.h"

#define  NORD  39
#define  NX    NORD
#define  NY    NORD
#define  NZ    NORD
#define  N     (NX * NY * NZ)
#define  NXY   (NX * NY)
#define  K     (N + 1)
#define  NDIAG  7
#define  NALL  (NDIAG * N)
#define  IWL   (36 * N + 36 + 2 * NALL + 3 * (N + 1))
#define  IPRINT  0

void  init_mat_diag(double, double, double, double, double*, int*, int, int,
                    int, double, double, double, int, int, int);
double  errnrm(double*, double*, int);

int MAIN__() {

  int  nofst[NDIAG];
```

```
double  diag[NDIAG][K], diag2[NDIAG][K];
double  c[K * NDIAG], wc[K * NDIAG];
int  nrowc[K * NDIAG], nfcnzc[N + 1], iwc[K * NDIAG][2];
double  w[NDIAG * N + N];
int  nperm[N],
     nposto[N], ndim[N][2],
     nassign[N],
     iw[IWL];
double  *panelfactorl, *panelfactoru;
int  *npanelindexl,
     *npanelindexu;
double  dummyfl, dummyfu;
int  ndummyil, ndummyiu;
long  nsizefactorl, nsizeindex,
      nsizefactoru,
      nfcnzfactorl[N + 1],
      nfcnzfactoru[N + 1],
      nfcnzindexl[N + 1],
      nfcnzindexu[N + 1];
double  x[N], b[N], solex[N];
int  i, j, nbase, length, numnzc, ntopcfgc, ncol, nnzc;
double  va1, va2, va3, vc, xl, yl, zl;

double  thepsz,
        epsr,
        sepsz,
        sclrow[N], sclcol[N];
double  epsz, err;

int  ipivot, istatic,
     isclitermax,
     irefine, itermax, iter, icon;
int  iordering, isw, nsupnum;



printf("    DIRECT METHOD\n");
printf("    FOR SPARSE STRUCTURALLY SYMMETRIC REAL MATRICES\n");
printf("    IN COMPRESSED COLUMN STORAGE\n\n");

for (i = 0; i < N; i++) {
  solex[i] = 1.0;
}
printf("    EXPECTED SOLUTIONS\n");
printf("    X(1) = %19.16lf X(N) = %19.16lf\n\n", solex[0], solex[N - 1]);

va1 = 1.0;
va2 = 2.0;
va3 = 3.0;
```

```
vc = 4.0;
xl = 1.0;
yl = 1.0;
zl = 1.0;
init_mat_diag(va1, va2, va3, vc, (double *)diag, nofst,
              NX, NY, NZ, xl, yl, zl, NDIAG, N, K);

for (i = 0; i < NDIAG; i++) {
  for (j = 0; j < K; j++) {
    diag2[i][j] = 0;
  }
}

for (i = 0; i < NDIAG; i++) {

  if (nofst[i] < 0) {
    nbase = - nofst[i];
    length = N - nbase;
    for (j = 0; j < length; j++) {
      diag2[i][j] = diag[i][nbase + j];
    }
  } else {
    nbase = nofst[i];
    length = N - nbase;
    for (j = 0; j < length; j++) {
      diag2[i][nbase + j] = diag[i][j];
    }
  }

}

numnzc = 0;

for (j = 0; j < N; j++) {
  ntopcfgc = 1;

  for (i = NDIAG - 1; i >= 0; i--) {

    if (diag2[i][j] != 0.0) {

      ncol = (j + 1) - nofst[i];
      c[numnzc] = diag2[i][j];
      nrowc[numnzc] = ncol;

      if (ntopcfgc == 1) {
        nfcnzc[j] = numnzc + 1;
        ntopcfgc = 0;
      }
```

```
      numnzc++;

    }
  }
}

nfcnzc[N] = numnzc + 1;
nnzc = numnzc;

c_dm_vmvscc(c, nnzc, nrowc, nfcnzc, N, solex,
            b, wc, (int *)iwc, &icon);


for (i = 0; i < N; i++) {
  x[i] = b[i];
}
iordering = 0;
isclitermax = 10;
isw = 1;
epsz = 1.0e-16;
nsizefactorl = 1;
nsizefactoru = 1;
nsizeindex = 1;
thepsz = 1.0e-2;
epsr = 1.0e-8;
sepsz = 1.0e-10;
ipivot = 40;
istatic = 1;
irefine = 1;
itermax = 10;

c_dm_vssslu(c, nnzc, nrowc, nfcnzc, N,
            isclitermax, iordering,
            nperm, isw,
            nassign,
            &nsupnum,
            nfcnzfactorl, &dummyfl,
            &nsizefactorl, nfcnzindexl,
            &ndummyil, &nsizeindex, (int *)ndim,
            nfcnzfactoru, &dummyfu,
            &nsizefactoru,
            nfcnzindexu, &ndummyiu,
            nposto,
            sclrow, sclcol,
            &epsz,
            &thepsz,
            ipivot, istatic, &sepsz,
            w, iw, &icon);
printf("    ICON=%6d NSIZEFACTORL=%9ld NSIZEFACTORU=%9ld NSIZEINDEX=%9ld\n",
```

```
       icon, nsizefactorl, nsizefactoru, nsizeindex);
printf("    NSUPNUM=%d\n\n", nsupnum);

panelfactorl = (double *)malloc(sizeof(double) * nsizefactorl);
panelfactoru = (double *)malloc(sizeof(double) * nsizefactoru);
npanelindexl = (int *)malloc(sizeof(int) * nsizeindex);
npanelindexu = (int *)malloc(sizeof(int) * nsizeindex);

isw = 2;
c_dm_vssslu(c, nnzc, nrowc, nfcnzc, N,
            isclitermax, iordering,
            nperm, isw,
            nassign,
            &nsupnum,
            nfcnzfactorl, panelfactorl,
            &nsizefactorl, nfcnzindexl,
            npanelindexl, &nsizeindex, (int *)ndim,
            nfcnzfactoru, panelfactoru,
            &nsizefactoru,
            nfcnzindexu, npanelindexu,
            nposto,
            sclrow, sclcol,
            &epsz,
            &thepsz,
            ipivot, istatic, &sepsz,
            w, iw, &icon);

c_dm_vssslux(N,
             iordering,
             nperm,
             x,
             nassign,
             nsupnum,
             nfcnzfactorl, panelfactorl,
             nsizefactorl, nfcnzindexl,
             npanelindexl, nsizeindex, (int *)ndim,
             nfcnzfactoru, panelfactoru,
             nsizefactoru,
             nfcnzindexu, npanelindexu,
             nposto,
             sclrow, sclcol,
             irefine, epsr, itermax, &iter,
             c, nnzc, nrowc, nfcnzc,
             iw,
             &icon);


err = errnrm(solex, x, N);
```

```
  printf("    COMPUTED VALUES\n");
  printf("    X(1) = %19.16lf X(N) = %19.16lf\n\n", x[0], x[N - 1]);
  printf("    ICON = %6d\n\n", icon);
  printf("    N = %d :: NX = %d  NY = %d NZ = %d\n\n", N, NX, NY, NZ);
  printf("    ERROR = %10.3le\n", err);
  printf("    ITER=%d\n\n\n", iter);

  if (err < 1.0e-8 && icon == 0) {
    printf("    ********** OK **********\n");
  } else {
    printf("    ********** NG **********\n");
  }

  free(panelfactorl);
  free(panelfactoru);
  free(npanelindexl);
  free(npanelindexu);

  return(0);
}

/* ========================================
      INITIALIZE COEFFICIENT MATRIX
   ======================================== */
void init_mat_diag(double va1, double va2, double va3, double vc, double *d_l,
                   int *offset, int nx, int ny, int nz, double xl, double yl,
                   double zl, int ndiag, int len, int ndivp) {

  if (ndiag < 1) {
    printf("SUB FUNCTION INIT_MAT_DIAG:\n");
    printf(" NDIAG SHOULD BE GREATER THAN OR EQUAL TO 1\n");
    return;
  }

#pragma omp parallel default(shared)
{
  int  ndiag_loc, i, j, l, nxy, i0, j0, k0, js;
  double  hx, hy, hz, hx2, hy2, hz2;

/* NDIAG CANNOT BE GREATER THAN 7 */
  ndiag_loc = ndiag;
  if (ndiag > 7) ndiag_loc = 7;

/* INITIAL SETTING */
  hx = xl / (nx + 1);
  hy = yl / (ny + 1);
  hz = zl / (nz + 1);

#pragma omp for
```

```
  for (i = 0; i < ndivp * ndiag; i++) {
    d_l[i] = 0.0;
  }

  nxy = nx * ny;

/* OFFSET SETTING */
#pragma omp single
  {
    l = 0;
    if (ndiag_loc >= 7) {
      offset[l] = -nxy;
      l++;
    }
    if (ndiag_loc >= 5) {
      offset[l] = -nx;
      l++;
    }
    if (ndiag_loc >= 3) {
      offset[l] = -1;
      l++;
    }
    offset[l] = 0;
    l++;
    if (ndiag_loc >= 2) {
      offset[l] = 1;
      l++;
    }
    if (ndiag_loc >= 4) {
      offset[l] = nx;
      l++;
    }
    if (ndiag_loc >= 6) {
      offset[l] = nxy;
    }
  }

/* MAIN LOOP */
#pragma omp for
  for (j = 0; j < len; j++) {
    js = j + 1;

/* DECOMPOSE JS-1 = (K0-1)*NX*NY+(J0-1)*NX+I0-1 */
    k0 = (js - 1) / nxy + 1;
    if (k0 > nz) {
      printf("ERROR; K0.GH.NZ \n");
      continue;
    }
    j0 = (js - 1 - nxy * (k0 - 1)) / nx + 1;
```

```
      i0 = js - nxy * (k0 - 1) - nx * (j0 - 1);
      l = 0;

      if (ndiag_loc >= 7) {
        if (k0 > 1) d_l[l * ndivp + j] = -(1.0 / hz + 0.5 * va3) / hz;
        l++;
      }
      if (ndiag_loc >= 5) {
        if (j0 > 1) d_l[l * ndivp + j] = -(1.0 / hy + 0.5 * va2) / hy;
        l++;
      }
      if (ndiag_loc >= 3) {
        if (i0 > 1) d_l[l * ndivp + j] = -(1.0 / hx + 0.5 * va1) / hx;
        l++;
      }
      hx2 = hx * hx;
      hy2 = hy * hy;
      hz2 = hz * hz;
      d_l[l * ndivp + j] = 2.0 / hx2 + vc;
      if (ndiag_loc >= 5) {
        d_l[l * ndivp + j] += 2.0 / hy2;
        if (ndiag_loc >= 7) {
          d_l[l * ndivp + j] += 2.0 / hz2;
        }
      }
      l++;
      if (ndiag_loc >= 2) {
        if (i0 < nx) d_l[l * ndivp + j] = -(1.0 / hx - 0.5 * va1) / hx;
        l++;
      }
      if (ndiag_loc >= 4) {
        if (j0 < ny) d_l[l * ndivp + j] = -(1.0 / hy - 0.5 * va2) / hy;
        l++;
      }
      if (ndiag_loc >= 6) {
        if (k0 < nz) d_l[l * ndivp + j] = -(1.0 / hz - 0.5 * va3) / hz;
      }
    }

  }

    return;
  }

  /* ======================================
    * SOLUTE ERROR
    * | X1 - X2 |
     ====================================== */
  double errnrm(double *x1, double *x2, int len) {
```

```
  double  s, ss, rtc;
  int  i;

  s = 0.0;
  for (i = 0; i < len; i++) {
    ss = x1[i] - x2[i];
    s += ss * ss;
  }

  rtc = sqrt(s);
  return(rtc);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VSSSLU の項目及び[2], [19], [22], [48], [63], [68], [69]を参照してください.

c_dm_vssslux

LU 分解された構造的に対称な実スパース行列の連立 1 次方程式
ierr = c_dm_vssslux(n, iordering, nperm, b, nassign, nsupnum, nfcnzfactorl, panelfactorl, nsizefactorl, nfcnzindexl, npanelindexl, nsizeindex, ndim, nfcnzfactoru, panelfactoru, nsizefactoru, nfcnzindexu, npanelindexu, nposto, sclrow, sclcol, irefine, epsr, itermax, &iter, a, nz, nrow, nfcnz, iw, &icon);

1. 機能

$n \times n$ の構造的に対称な実スパース行列 **A** に行および列の均衡化を行うスケーリングを行い, そしてスーパーノード内で Pivot をとり, 以下のように LU 分解されています. LU 分解の結果を利用して以下の連立 1 次方程式を解きます.

$$\mathbf{Ax = b}$$

行列 **A** は以下のように分解されています.

$$\mathbf{P_{rs}QPD_rAD_cP^TQ^TP_{cs} = LU}$$

ここで, 構造的に対称な実スパース行列 **A** は以下のように変換されています.

$$\mathbf{A_1 = D_rAD_c}$$

ここで $\mathbf{D_r}$ は行のスケーリングを行う対角行列, $\mathbf{D_c}$ は列のスケーリングを行う対角行列です.

$$\mathbf{A_2 = QPA_1P^TQ^T}$$

$\mathbf{A_2}$ は, スーパーノード内に閉じて指定された Pivot を行い, 行および列の入れ換えを行い LU 分解されています.
$\mathbf{P_{rs}}$, $\mathbf{P_{cs}}$ はそれぞれ行の入れ換えおよび列の入れ換えを示す直交行列です. 実際の入れ換えはスーパーノードに属する限られた行列の部分で行われます.
ただし, **P** は, **A** の非ゼロパターンに対して求めた `ordering` による行列要素の並び換えを示す置換行列で, **Q** は post order による行列要素の並び換えを示す置換行列を示します. **P**, **Q** は直交行列です.

L は下三角行列, **U** は単位上三角行列です.
解の反復改良で精度を改良することを指定できます.

2. 引数

呼出し形式:

```
ierr = c_dm_vsrlux(n, iordering, nperm, b, nassign, nsupnum, nfcnzfactorl,
            panelfactorl, nsizefactorl, nfcnzindexl, npanelindexl,
            nsizeindex, (int *)ndim, nfcnzfactoru, panelfactoru,
            nsizefactoru, nfcnzindexu, npanelindexu, nposto,
            sclrow, sclcol,
            irefine, &epsr, itermax, &iter, a, nz, nrow, nfcnz, iw, &icon);
```

引数の説明:

n	int	入力	行列 **A** の次数 n.
iordering	int	入力	1 のとき, nperm に ordering を指定して LU 分解されたことを示します. $\mathbf{PA_1P^T}$ を LU 分解しました. 1 以外のとき, ordering は指定されていません. ("使用上の注意" a)参照)
nperm	int nperm[n]	入力	iordering = 1 のとき使用する置換行列をベクトルで指定します. ("使用上の注意" b)参照)
b	double b[n]	入力	**Ax** = **b** の右辺定数ベクトル.
		出力	解ベクトル.
nassign	int nassign[n]	入力	各 supernode に対する **L**, **U** は圧縮して 2 次元の panel に格納します.この panel を panelfactorl および panelfactoru の 1 次元部分配列として順番に割り付けたとき何番目になるかを示します. これらの指標ベクトルも同じように npanelindexl, npanelindexu に割り付けます. j = nassign[i-1]のとき, *i* 番目の supernode を *j* 番目に割り付けたことを示します. 分解結果の格納方法については図 c_dm_vssslux-1 を参照してください.
nsupnum	int	入力	supernode の総数. (≤ n)
nfcnzfactorl	long nfcnzfactorl[n+1]	入力	構造的に対称な実スパース行列の LU 分解の結果の行列 **L** および **U** はスーパーノードに対応しておのおの求めます. 各 supernode に対応する **L** の列ベクトルは, 共通の行指標ベクトルを持つように圧縮してブロック対角部分に **U** の対応部分を含めて 2 次元の panel に格納します. この panel を panelfactorl の 1 次元部分配列として順番に割り付けたとき, *i* 番目の panel の先頭要素 panel[0][0]が 1 次元配列の panelfactorl の何番目の要素になるかを示します. 結果の格納方法については図 c_dm_vssslux-1 を参照してください.
panelfactorl	double panelfactorl [nsizefactorl]	入力	各 supernode の分解結果に対応する行列 **L** の複数の列ベクトルと行列 **U** のブロック対角部分について, 共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します. panel を順番に割り付けます. *i* 番目の supernode に対応する panel が何番目の位置に割り当てられるかは j = nassign[i-1]から分かります. その先頭位置は nfcnzfactorl[j-1]に格納されています. panel ごとに分解結果が格納されます. *i* 番目に割付けられた panel の大きさは ndim[i-1][0] × ndim[i-1][1]の 2 次元配列と見なせます. *i* 番目の panel の panel[t-1][s-1], s ≥ t, s = 1,…,

			ndim[i-1][0], t = 1,..., ndim[i-1][1]に下三角行列 **L** が格納されます. panel の panel[t-1][s-1], s < t, t = 1,..., ndim[i-1][1]には単位上三角行列 **U** の対角要素を除いたブロック対角部分が格納されます. 結果の格納方法については図 c_dm_vssslux-1 を参照してください.
nsizefactorl	long	入力	panelfactorl の大きさを示す.
nfcnzindexl	long nfcnzindexl[n+1]	入力	各 supernode の分解結果に対応する行列 **L** の複数の列ベクトルと行列 **U** のブロック対角部分について, 共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します. この行指標ベクトルを npanelindexl に順番に割り付けたとき *i* 番目の行指標ベクトルの先頭要素が 1 次元配列である npanelindexl の何番目の要素になるかを示します.
npanelindexl	int npanelindexl [nsizeindex]	入力	各 supernode の分解結果に対応する行列 **L** の複数の列ベクトルと行列 **U** のブロック対角部分について, 共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します. この行指標ベクトルを npanelindexl に順番に割り付けます. *i* 番目の supernode に対応する panel の行の指標ベクトルが何番目に割り当てられるかは j = nassign[i-1]から分かります. その先頭位置は nfcnzindexl[j-1]に格納されています. この行指標は post order で並び換えたときの行の番号です. 結果の格納方法については図 c_dm_vssslux-1 を参照してください.
nsizeindex	long	入力	npanelindexl および npanelindexu の大きさを示す.
ndim	int ndim[n][2]	入力	ndim[i-1][0]および ndim[i-1][1]は行列 **L** に関して *i* 番目に割り付けられた panel の 1 次元目と 2 次元目の大きさを示します. ndim[i-1][0] – ndim[i-1][1]および ndim[i-1][1]は行列 **U** に関して割り付けられた panel を転置したときの 1 次元目と 2 次元目の大きさを示します. 結果の格納方法については図 c_dm_vssslux-1 を参照してください.
nfcnzfactoru	long nfcnzfactoru[n+1]	入力	構造的に対称な実スパース行列の LU 分解の結果の行列 **U** に関しては, 各 supernode に対応する **U** の行ベクトルは, ブロック対角部分を除いて共通の列指標ベクトルを持つように圧縮し, 転置して 2 次元の panel に格納します. この panel を panelfactoru の 1 次元部分配列として順番に割り付けたとき, i 番目の panel の先頭要素 panel[0][0]が 1 次元配列の panelfactoru の何番目の要素になるかを示します. 結果の格納方法については図 c_dm_vssslux-1 を参照してください.
panelfactoru	double panelfactoru [nsizefactoru]	入力	各 supernode の分解結果に対応する行列 **U** のブロック対角部分を除いた複数の行ベクトルについて, 共通の列指標ベクトルを持つように圧縮し, 転置して 2 次元の

			panel に格納します. panel を順番に割り付けます. *i* 番目の supernode に対応する panel が何番目の位置に割り当てられるかは j = nassign[i-1]から分かります. その先頭位置は nfcnzfactoru[j-1]に格納されています. panel ごとに分解結果が格納されます. *i* 番目に割付けられた panel の大きさは{ndim[i-1][0] – ndim[i-1][1]} × ndim[i-1][1]の 2 次元配列と見なせます. *i* 番目の panel の panel[t-1][s-1], s = 1,…, ndim[i-1][0] – ndim[i-1][1], t = 1,…, ndim[i-1][1]に単位上三角行列 **U** のブロック対角部分を除いた部分が行ベクトルを圧縮し, 転置したものが格納されます. 結果の格納方法については図 c_dm_vssslux-1 を参照してください.
nsizefactoru	long	入力	panelfactoru の大きさを示す.
nfcnzindexu	long nfcnzindexu[n+1]	入力	各 supernode の分解結果に対応する行列 **U** は対角ブロック部分を除いて複数の行ベクトルについて, 共通の列指標ベクトルを持つように圧縮し, 転置した形で 2 次元の panel に格納します. 対角ブロック部分も含んだこの列指標ベクトルを npanelindexu に順番に割り付けたとき *i* 番目の列指標ベクトルの先頭要素が 1 次元配列である npanelindexu の何番目の要素になるかを示します. 結果の格納方法については図 c_dm_vssslux-1 を参照してください.
npanelindexu	int npanelindexu [nsizeindex]	入力	各 supernode の分解結果に対応する行列 **U** の対角ブロックを除いた部分を転置して圧縮して 2 次元の panel に格納します. 対応する列指標ベクトルには対角ブロック部分を含めたものを npanelindexu に順番に割り付けます. *i* 番目の supernode に対応する panel の行の指標ベクトルが何番目に割り当てられるかは j = nassign[i-1]から分かります. その先頭位置は nfcnzindexu[j-1]に格納されています. この行指標は post order で並び換えたときの列の番号です. 結果の格納方法については図 c_dm_vssslux-1 を参照してください.
nposto	int nposto[n]	入力	post order で *i* 番目の node が行列 **A** の何番目の列番号に対応するかを表す 1 次元ベクトル. (“使用上の注意” c)参照)
sclrow	double sclrow[n]	入力	スケーリング対角行列 $\mathbf{D}_r$. 対角要素が 1 次元配列に格納されます.
sclcol	double sclcol[n]	入力	スケーリング対角行列 $\mathbf{D}_c$. 対角要素が 1 次元配列に格納されます.
irefine	int	入力	LU 分解結果を利用して解を求めるときに, 解の反復改良を行うかどうかを示す. 残差ベクトルを計算するときに 4 倍精度で計算します. =1 : 解の反復改良を行う. 反復改良して得られる残差ベクトル $\mathbf{r}_k$ の絶対値の差分がひとつ前の差分の 1/4 より大きくなるまで, 反復改良を行います.

			≠1: 解の反復改良を行わない.
epsr	double	入力	解の残差ベクトル **b-Ax** の絶対値が, **b** の絶対値に対して十分小さいかを判定する判定値 epsr ≤ 0.0 のとき 10^{-6} が設定されます.
itermax	int	入力	(≥ 1) 反復改良を行うときの最大反復回数.
iter	int	出力	反復改良を行った回数.
a	double a[nz]	入力	構造的に対称な実スパースな係数行列 **A** を圧縮列格納法で格納します. 圧縮列格納法については, 実スパース行列と実ベクトル(圧縮列格納法), c_dm_vmvscc の図 c_dm_vmvscc-1 を参照してください.
nz	int	入力	構造的に対称な実スパースな係数行列 **A** の格納する非零要素の総数.
nrow	int nrow[nz]	入力	圧縮列格納法で使用される行指標で a に格納される要素が何番目の行ベクトルに属するかを示します.
nfcnz	int nfcnz[n+1]	入力	行列 **A** の各列の格納する非零要素を列方向に圧縮して順次配列 a に格納するとき,対応する列の最初の格納する非零要素が格納される位置を表します. nfcnz[n] = nz + 1.
iw	int iw[36*n+36+2*nz+3*(n+1)]	作業域	構造的に対称な実スパース行列の LU 分解を行う c_dm_vssslu からのデータの受け渡しに使われます. 呼び出しの間で値を変更してはいけません.
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.
20400	LU 分解された行列の対角要素にゼロがあった	処理を打ち切る.
20500	求めた解ベクトルに対する残差ベクトルのノルムが方程式 **Ax** = **b** の右辺ベクトルのノルムの epsr 倍より大きい. 係数行列は特異に近い可能性がある.	
30000	次のいずれかであった: • n < 1 • nz < 0 • nfcnz[n] ≠ nz + 1 • nsizefactorl < 1 • nsizefactoru < 1 • nsizeindex < 1 • irefine = 1 のとき itermax < 1	
30100	nperm に指定した置換行列が正しくない.	
30200	nrow[j-1]に格納された *i* 列目の行指標 k が k < 1 または k > n	
30300	*i* 列目の行指標の数 nfcnz[i] – nfcnz[i-1] > n	

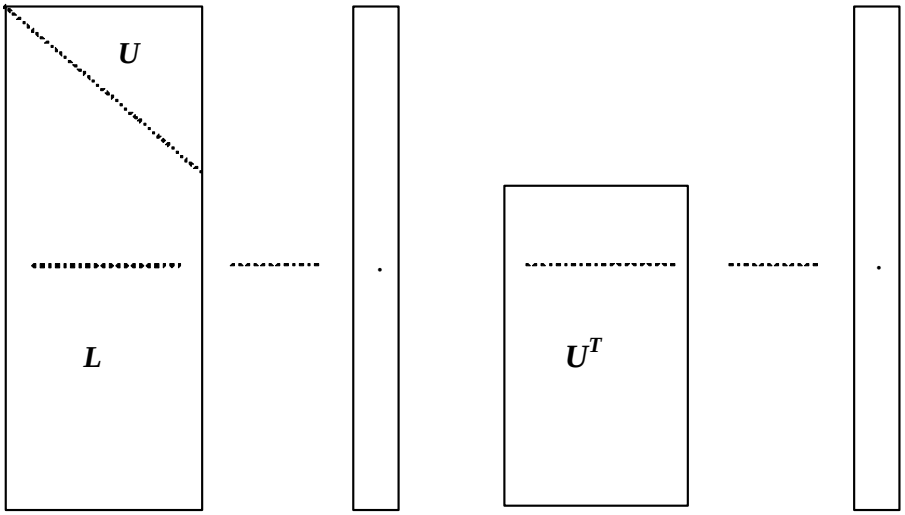


図 c_dm_vssslux-1 分解結果の格納概念図

j = nassign[i-1] → *i* 番目の supernode は *j* 番目に格納されます.

p = nfcnzfactorl[j-1] → *j* 番目の panel は panelfactorl の *p* 番目の要素から ndim[j-1][0] × ndim[j-1][1] の長さを占めます.

q = nfcnzindexl[j-1] → *j* 番目の panel の行指標を表すベクトルは npanelindexl の *q* 番目の要素から ndim[j-1][0] の長さを占めます.

panel は大きさ ndim[j-1][0] × ndim[j-1][1] の配列と見なせます.

panel[t-1][s-1], s ≥ t, s = 1,… , ndim[j-1][0], t = 1,…, ndim[j-1][1] に分解結果の下三角行列 **L** が格納されます.

panel[t-1][s-1], s < t, s = 1,…, ndim[j-1][1], t = 1,…, ndim[j-1][1] に単位上三角行列 **U** の対角ブロック部分が対角要素を除き格納されます.

u = nfcnzfactoru[j-1] → *j* 番目の panel は panelfactoru の *u* 番目の要素から(ndim[j-1][0] – ndim[j-1][1]) × ndim[j-1][1] の長さを占めます.

v = nfcnzindexu[j-1] → *j* 番目の panel の列指標を表すベクトルは npanelindexu の *v* 番目の要素から ndim[j-1][0] の長さを占めます.

panel は大きさ (ndim[j-1][0] – ndim[j-1][1]) × ndim[j-1][1] の配列と見なせます.

panel[y-1][x-1], x = 1,…, ndim[j-1][0] - ndim[j-1][1], y = 1,…, ndim[j-1][1]

に分解結果の単位上三角行列 **U** の転置行列 $\mathbf{U}^{\mathrm{T}}$ の対角ブロック部分を除いた部分が格納されます.

指標の値は行列 **A** の列番号をもつ node を post order で並び換えた $\mathbf{QAQ}^{\mathrm{T}}$ での列番号を表します.

3. 使用上の注意

a)

c_dm_vssslu で LU 分解を行った結果を利用します.
c_dm_vssslu の“使用上の注意” e)参照や c_dm_vssslux の使用例を参照願います.

b)

直交行列である置換行列 **P** の要素 $\mathbf{p_{ij}} = \mathbf{1}$ のとき, nperm[i-1] = j と表現します.
逆は以下のようにして求めることができます.

```
for (i = 1; i <= n; i++) {
 j = nperm[i-1];
 nperminv[j-1] = i;
}
```

c)

列番号に対応するノードを考えます. これを post order で順番を並び換えたときの番号の対応関係が nposto に格納されています. post order の *i* 番目のノードが並び換える前の何番目のノードに対応するかを表します. j=nposto[i-1]は *j* 番目であることを表します.
b)同様にこれは直交行列である置換行列 **Q** を表し,行列 **A** を $\mathbf{QAQ^T}$ と並び換えることに相当します.
逆変換 $\mathbf{Q^T}$ は以下のようにして求めることができます.

```
for (i = 1; i <= n; i++) {
 j = nposto[i-1];
 npostoinv[j-1] = i;
}
```

4. 使用例

連立 1 次方程式 **Ax** = **f** を解きます.
行列は境界条件として立方体の境界で 0 となる楕円型の偏微分方程式に有限差分法を適用して生成されたものを利用します.

$$-\Delta u + a\nabla u + cu = f$$

ここで $a = (a_1, a_2, a_3)$.
行列は関数 init_mat_diag によって生成されます.
対角形式格納法で生成し,圧縮列格納法で格納します. 結果生成された構造的に対称な行列 **A** に関する連立 1 次方程式を解きます.
(並列実行を行うスレッド数は環境変数(OMP_NUM_THREADS)などで指定できます.
例えば, 4 プロセッサーのシステムで 4 つのスレッドで並列実行するときは, OMP_NUM_THREADS を 4 に設定して実行します.)

```
/* **EXAMPLE** */
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <malloc.h>
#include <omp.h>
#include "cssl.h"


#define  NORD  39
#define  NX    NORD
#define  NY    NORD
#define  NZ    NORD
#define  N     (NX * NY * NZ)
#define  NXY   (NX * NY)
```

```
#define  K     (N + 1)
#define  NDIAG  7
#define  NALL  (NDIAG * N)
#define  IWL   (36 * N + 36 + 2 * NALL + 3 * (N + 1))
#define  IPRINT  0

void  init_mat_diag(double, double, double, double, double*, int*, int, int,
                    int, double, double, double, int, int, int);
double  errnrm(double*, double*, int);

int MAIN__() {

  int  nofst[NDIAG];
  double  diag[NDIAG][K], diag2[NDIAG][K];
  double  c[K * NDIAG], wc[K * NDIAG];
  int  nrowc[K * NDIAG], nfcnzc[N + 1], iwc[K * NDIAG][2];
  double  w[NDIAG * N + N];
  int  nperm[N],
       nposto[N], ndim[N][2],
       nassign[N],
       iw[IWL];
  double  *panelfactorl, *panelfactoru;
  int  *npanelindexl,
       *npanelindexu;
  double  dummyfl, dummyfu;
  int  ndummyil, ndummyiu;
  long  nsizefactorl, nsizeindex,
        nsizefactoru,
        nfcnzfactorl[N + 1],
        nfcnzfactoru[N + 1],
        nfcnzindexl[N + 1],
        nfcnzindexu[N + 1];
  double  x[N], b[N], solex[N];
  int  i, j, nbase, length, numnzc, ntopcfgc, ncol, nnzc;
  double  va1, va2, va3, vc, xl, yl, zl;

  double  thepsz,
          epsr,
          sepsz,
          sclrow[N], sclcol[N];
  double  epsz, err;

  int  ipivot, istatic,
       isclitermax,
       irefine, itermax, iter, icon;
  int  iordering, isw, nsupnum;
```

```
printf("    DIRECT METHOD\n");
printf("    FOR SPARSE STRUCTURALLY SYMMETRIC REAL MATRICES\n");
printf("    IN COMPRESSED COLUMN STORAGE\n\n");

for (i = 0; i < N; i++) {
  solex[i] = 1.0;
}
printf("    EXPECTED SOLUTIONS\n");
printf("    X(1) = %19.16lf X(N) = %19.16lf\n\n", solex[0], solex[N - 1]);

va1 = 1.0;
va2 = 2.0;
va3 = 3.0;
vc = 4.0;
xl = 1.0;
yl = 1.0;
zl = 1.0;
init_mat_diag(va1, va2, va3, vc, (double *)diag, nofst,
              NX, NY, NZ, xl, yl, zl, NDIAG, N, K);

for (i = 0; i < NDIAG; i++) {
  for (j = 0; j < K; j++) {
    diag2[i][j] = 0;
  }
}

for (i = 0; i < NDIAG; i++) {

  if (nofst[i] < 0) {
    nbase = - nofst[i];
    length = N - nbase;
    for (j = 0; j < length; j++) {
      diag2[i][j] = diag[i][nbase + j];
    }
  } else {
    nbase = nofst[i];
    length = N - nbase;
    for (j = 0; j < length; j++) {
      diag2[i][nbase + j] = diag[i][j];
    }
  }

}

numnzc = 0;

for (j = 0; j < N; j++) {
  ntopcfgc = 1;
```

```
  for (i = NDIAG - 1; i >= 0; i--) {

    if (diag2[i][j] != 0.0) {

      ncol = (j + 1) - nofst[i];
      c[numnzc] = diag2[i][j];
      nrowc[numnzc] = ncol;

      if (ntopcfgc == 1) {
        nfcnzc[j] = numnzc + 1;
        ntopcfgc = 0;
      }

      numnzc++;

    }
  }
}

nfcnzc[N] = numnzc + 1;
nnzc = numnzc;

c_dm_vmvscc(c, nnzc, nrowc, nfcnzc, N, solex,
            b, wc, (int *)iwc, &icon);


for (i = 0; i < N; i++) {
  x[i] = b[i];
}
iordering = 0;
isclitermax = 10;
isw = 1;
epsz = 1.0e-16;
nsizefactorl = 1;
nsizefactoru = 1;
nsizeindex = 1;
thepsz = 1.0e-2;
epsr = 1.0e-8;
sepsz = 1.0e-10;
ipivot = 40;
istatic = 1;
irefine = 1;
itermax = 10;

c_dm_vssslu(c, nnzc, nrowc, nfcnzc, N,
            isclitermax, iordering,
            nperm, isw,
            nassign,
            &nsupnum,
```

```
            nfcnzfactorl, &dummyfl,
            &nsizefactorl, nfcnzindexl,
            &ndummyil, &nsizeindex, (int *)ndim,
            nfcnzfactoru, &dummyfu,
            &nsizefactoru,
            nfcnzindexu, &ndummyiu,
            nposto,
            sclrow, sclcol,
            &epsz,
            &thepsz,
            ipivot, istatic, &sepsz,
            w, iw, &icon);
printf("    ICON=%6d NSIZEFACTORL=%9ld NSIZEFACTORU=%9ld NSIZEINDEX=%9ld\n",
       icon, nsizefactorl, nsizefactoru, nsizeindex);
printf("    NSUPNUM=%d\n\n", nsupnum);

panelfactorl = (double *)malloc(sizeof(double) * nsizefactorl);
panelfactoru = (double *)malloc(sizeof(double) * nsizefactoru);
npanelindexl = (int *)malloc(sizeof(int) * nsizeindex);
npanelindexu = (int *)malloc(sizeof(int) * nsizeindex);

isw = 2;
c_dm_vssslu(c, nnzc, nrowc, nfcnzc, N,
            isclitermax, iordering,
            nperm, isw,
            nassign,
            &nsupnum,
            nfcnzfactorl, panelfactorl,
            &nsizefactorl, nfcnzindexl,
            npanelindexl, &nsizeindex, (int *)ndim,
            nfcnzfactoru, panelfactoru,
            &nsizefactoru,
            nfcnzindexu, npanelindexu,
            nposto,
            sclrow, sclcol,
            &epsz,
            &thepsz,
            ipivot, istatic, &sepsz,
            w, iw, &icon);

c_dm_vssslux(N,
             iordering,
             nperm,
             x,
             nassign,
             nsupnum,
             nfcnzfactorl, panelfactorl,
             nsizefactorl, nfcnzindexl,
             npanelindexl, nsizeindex, (int *)ndim,
```

```
                nfcnzfactoru, panelfactoru,
                nsizefactoru,
                nfcnzindexu, npanelindexu,
                nposto,
                sclrow, sclcol,
                irefine, epsr, itermax, &iter,
                c, nnzc, nrowc, nfcnzc,
                iw,
                &icon);


  err = errnrm(solex, x, N);

  printf("    COMPUTED VALUES\n");
  printf("    X(1) = %19.16lf X(N) = %19.16lf\n\n", x[0], x[N - 1]);
  printf("    ICON = %6d\n\n", icon);
  printf("    N = %d :: NX = %d  NY = %d NZ = %d\n\n", N, NX, NY, NZ);
  printf("    ERROR = %10.3le\n", err);
  printf("    ITER=%d\n\n\n", iter);

  if (err < 1.0e-8 && icon == 0) {
    printf("    ********** OK **********\n");
  } else {
    printf("    ********** NG **********\n");
  }

  free(panelfactorl);
  free(panelfactoru);
  free(npanelindexl);
  free(npanelindexu);

  return(0);
}

/* ======================================
      INITIALIZE COEFFICIENT MATRIX
   ====================================== */
void init_mat_diag(double va1, double va2, double va3, double vc, double *d_l,
                   int *offset, int nx, int ny, int nz, double xl, double yl,
                   double zl, int ndiag, int len, int ndivp) {

  if (ndiag < 1) {
    printf("SUB FUNCTION INIT_MAT_DIAG:\n");
    printf(" NDIAG SHOULD BE GREATER THAN OR EQUAL TO 1\n");
    return;
  }

#pragma omp parallel default(shared)
{
```

```
  int  ndiag_loc, i, j, l, nxy, i0, j0, k0, js;
  double  hx, hy, hz, hx2, hy2, hz2;

/* NDIAG CANNOT BE GREATER THAN 7 */
  ndiag_loc = ndiag;
  if (ndiag > 7) ndiag_loc = 7;

/* INITIAL SETTING */
  hx = xl / (nx + 1);
  hy = yl / (ny + 1);
  hz = zl / (nz + 1);

#pragma omp for
  for (i = 0; i < ndivp * ndiag; i++) {
    d_l[i] = 0.0;
  }

  nxy = nx * ny;

/* OFFSET SETTING */
#pragma omp single
  {
    l = 0;
    if (ndiag_loc >= 7) {
      offset[l] = -nxy;
      l++;
    }
    if (ndiag_loc >= 5) {
      offset[l] = -nx;
      l++;
    }
    if (ndiag_loc >= 3) {
      offset[l] = -1;
      l++;
    }
    offset[l] = 0;
    l++;
    if (ndiag_loc >= 2) {
      offset[l] = 1;
      l++;
    }
    if (ndiag_loc >= 4) {
      offset[l] = nx;
      l++;
    }
    if (ndiag_loc >= 6) {
      offset[l] = nxy;
    }
  }
```

```
/* MAIN LOOP */
#pragma omp for
  for (j = 0; j < len; j++) {
    js = j + 1;

/* DECOMPOSE JS-1 = (K0-1)*NX*NY+(J0-1)*NX+I0-1 */
    k0 = (js - 1) / nxy + 1;
    if (k0 > nz) {
      printf("ERROR; K0.GH.NZ \n");
      continue;
    }
    j0 = (js - 1 - nxy * (k0 - 1)) / nx + 1;
    i0 = js - nxy * (k0 - 1) - nx * (j0 - 1);
    l = 0;

    if (ndiag_loc >= 7) {
      if (k0 > 1) d_l[l * ndivp + j] = -(1.0 / hz + 0.5 * va3) / hz;
      l++;
    }
    if (ndiag_loc >= 5) {
      if (j0 > 1) d_l[l * ndivp + j] = -(1.0 / hy + 0.5 * va2) / hy;
      l++;
    }
    if (ndiag_loc >= 3) {
      if (i0 > 1) d_l[l * ndivp + j] = -(1.0 / hx + 0.5 * va1) / hx;
      l++;
    }
    hx2 = hx * hx;
    hy2 = hy * hy;
    hz2 = hz * hz;
    d_l[l * ndivp + j] = 2.0 / hx2 + vc;
    if (ndiag_loc >= 5) {
      d_l[l * ndivp + j] += 2.0 / hy2;
      if (ndiag_loc >= 7) {
        d_l[l * ndivp + j] += 2.0 / hz2;
      }
    }
    l++;
    if (ndiag_loc >= 2) {
      if (i0 < nx) d_l[l * ndivp + j] = -(1.0 / hx - 0.5 * va1) / hx;
      l++;
    }
    if (ndiag_loc >= 4) {
      if (j0 < ny) d_l[l * ndivp + j] = -(1.0 / hy - 0.5 * va2) / hy;
      l++;
    }
    if (ndiag_loc >= 6) {
      if (k0 < nz) d_l[l * ndivp + j] = -(1.0 / hz - 0.5 * va3) / hz;
```

```
    }
  }

}

  return;
}

/* ======================================
  * SOLUTE ERROR
  * | X1 - X2 |
   ====================================== */
double errnrm(double *x1, double *x2, int len) {

  double  s, ss, rtc;
  int  i;

  s = 0.0;
  for (i = 0; i < len; i++) {
    ss = x1[i] - x2[i];
    s += ss * ss;
  }

  rtc = sqrt(s);
  return(rtc);
}
```

c_dm_vssss

構造的に対称な実スパース行列の連立 1 次方程式(LU 分解法)
ierr = c_dm_vssss(a, nz, nrow, nfcnz, n, isclitermax, iordering, nperm, isw, b, nassign, &nsupnum, nfcnzfactorl, panelfactorl, &nsizefactorl, nfcnzindexl, npanelindexl, &nsizeindex, ndim, nfcnzfactoru, panelfactoru, &nsizefactoru, nfcnzindexu, npanelindexu, nposto, sclrow, sclcol, &epsz, &thepsz, ipivot, istatic, &spepsz, irefine, epsr, itermax, &iter, w, iw, &icon);

1. 機能

$n \times n$ の構造的に対称な実スパース行列 **A** に行および列の均衡化を行うスケーリングを行います. スーパーノードの対角ブロック内で指定した Pivot をとり, LU 分解します.

(構造的に対称な実スパース行列の非零要素はそれと対称な位置に非零要素を持ち, その値は必ずしも等しくはない行列です.)

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

構造的に対称な実スパース行列 **A** は以下のように変換できます.

$$\mathbf{A}_1 = \mathbf{D}_r\mathbf{A}\mathbf{D}_c$$

ここで $\mathbf{D}_r$ は行のスケーリングを行う対角行列, $\mathbf{D}_c$ は列のスケーリングを行う対角行列です.

$$\mathbf{A}_2 = \mathbf{Q}\mathbf{P}\mathbf{A}_1\mathbf{P}^T\mathbf{Q}^T$$

$\mathbf{A}_2$ は, スーパーノードの対角ブロックで指定された Pivot を行い, 行および列の入れ換えを行い LU 分解されます.
ただし, **P** は, **SYM** = **A** の非ゼロパターンに対して求めた ordering による行列要素の並び換えを示す置換行列で, **Q** は post order による行列要素の並び換えを示す置換行列を示します. **P**, **Q** は直交行列です.

LU 分解の結果の行列 **L** と **U** の非零要素のパターンはそれぞれに対して対称になります.
L は下三角行列, **U** は単位上三角行列です.

Pivot 処理で, 閾値 thepsz より絶対値が大きな Pivot を見つけることが出来なかったとき, 対角ブロック内の絶対値が最大の要素を Pivot の候補とします. この値が小さ過ぎるときに Static Pivot を与えて近似的に LU 分解を続けることができます.そして LU 分解の結果を利用して,解を求めます.また,解の反復改良で精度を改良することを指定できます.

2. 引数

呼出し形式:

```
ierr = c_dm_vssss(a, nz, nrow, nfcnz, n, isclitermax,
            iordering, nperm, isw, b, nassign, &nsupnum,
```

```
            nfcnzfactorl, panelfactorl, &nsizefactorl, nfcnzindexl,
            npanelindexl, &nsizeindex, (int *)ndim, nfcnzfactoru,
            panelfactoru, &nsizefactoru, nfcnzindexu, npanelindexu,
            nposto, sclrow, sclcol, &epsz, &thepsz, ipivot,
            istatic, &spepsz, irefine, epsr, itermax, &iter, w, iw, &icon);
```

引数の説明:

a	double a[nz]	入力	構造的に対称な実スパースな係数行列 **A** を圧縮列格納法で格納します. 圧縮列格納法については, 実スパース行列と実ベクトル(圧縮列格納法), c_dm_vmvscc の図 c_dm_vmvscc-1 を参照してください.
nz	int	入力	構造的に対称な実スパースな係数行列 **A** の格納する非零要素の総数.
nrow	int nrow[nz]	入力	圧縮列格納法で使用される行指標で a に格納される要素が何番目の行ベクトルに属するかを示します.
nfcnz	int nfcnz[n+1]	入力	行列 **A** の各列の格納する非零要素を列方向に圧縮して順次配列 a に格納するとき,対応する列の最初の格納する非零要素が格納される位置を表します. nfcnz[n] = nz + 1.
n	int	入力	行列 **A** の次数 *n*.
isclitermax	int	入力	係数行列のスケーリングを行う対角行列 $\mathbf{D_r}$ と $\mathbf{D_c}$ を反復して求める反復回数. isclitermax ≤ 0 のときスケーリングは行わすに, $\mathbf{D_r}$ および $\mathbf{D_c}$ は単位行列が設定されます. isclitermax ≥ 10 のときは 10 回を上限値とします.
iordering	int	入力	ordering を表す置換行列 **P** で $\mathbf{PAP^T}$ と変換した行列を LU 分解するかを指定します. 置換行列は直交行列です. 1 のとき, $\mathbf{PAP^T}$ と変換した行列を LU 分解します. 1 以外のとき, 行列をそのまま LU 分解します. ("使用上の注意" a)参照)
nperm	int nperm[n]	入力	iordering = 1 のとき使用する置換行列をベクトルで指定します. ("使用上の注意" a)参照)
isw	int	入力	呼び出しに関する制御情報を示します. 1) 1 のとき 初回の呼び出し.Symbolic decomposition を行い, 必要な領域が割り当てられているか調べ計算を行います. 2) 2 のとき isw = 1 で呼び出したとき, panelfactorl, panelfactoru, npanelindexl または npanelindexu の大きさが不足して icon = 31000 で終了した処理を継続します. このとき, nsizefactorl, nsizefactoru, nsizeindex に返却された必要な大きさで panelfactorl, panelfactoru, npanelindexl, npanelindexu を確保し直して指定しなおして再度呼び出します. これらの引数と実行を制御する引数 isw 以外の引数に格納されている値は変更してはいけません. 3) 3 のとき

			先立つ呼び出しで LU 分解された同じ係数行列に対する連立 1 次方程式の右辺ベクトル **b** を変えて再度解くことを指定します. このとき先立つ呼び出しで LU 分解した結果を使います. 実行を制御する引数 isw と右辺ベクトル **b** を格納する引数 B 以外の引数に格納されている値を変更していけません.
b	double b[n]	入力	**Ax** = **b** の右辺定数ベクトル.
		出力	解ベクトル.
nassign	int nassign[n]	出力	各 supernode に対する L, U は圧縮して 2 次元の panel に格納します. この panel を panelfactorl および panelfactoru の 1 次元部分配列として順番に割り付けたとき何番目になるかを示します. これらの指標ベクトルも同じように npanelindexl, npanelindexu に割り付けます. j = nassign[i-1]のとき, *i* 番目の supernode を *j* 番目に割り付けたことを示します. 分解結果の格納方法については図 c_dm_vssss-1 を参照してください.
		入力	isw ≠ 1 のとき, 初回呼び出しの値を再利用します.
nsupnum	int	出力	supernode の総数.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します. (≤ n)
nfcnzfactorl	long nfcnzfactorl[n+1]	出力	構造的に対称な実スパース行列の LU 分解の結果の行列 **L** および **U** はスーパーノードに対応しておのおの求めます. 各 supernode に対応する **L** の列ベクトルは, 共通の行指標ベクトルを持つように圧縮してブロック対角部分に **U** の対応部分を含めて 2 次元の panel に格納します. この panel を panelfactorl の 1 次元部分配列として順番に割り付けたとき, *i* 番目の panel の先頭要素 panel[0][0]が 1 次元配列の panelfactorl の何番目の要素になるかを示します. 結果の格納方法については図 c_dm_vssss-1 を参照してください.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します.
panelfactorl	double panelfactorl [nsizefactorl]	出力	各 supernode の分解結果に対応する行列 **L** の複数の列ベクトルと行列 **U** のブロック対角部分について, 共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します. panel を順番に割り付けます. i 番目の supernode に対応する panel が何番目の位置に割り当てられるかは j = nassign[i-1]から分かります. その先頭位置は nfcnzfactorl[j]に格納されています. panel ごとに分解結果が格納されます. i 番目に割付けられた panel の大きさは ndim[i-1][0] × ndim[i-1][1]の 2 次元配列と見なせます. *i* 番目の panel の panel[t-1][s-1], s ≥ t, s = 1,…, ndim[i-1][0], t=1,…, ndim[i-1][1]に下三角行列 **L** が格納されます. panel の panel[t-1][s-1], s < t, t = 1,…, ndim[i][1]には単位上三角行列 **U** の対角要素を除いたブロック対角部分が格納されます.

			結果の格納方法については図 c_dm_vssss-1 を参照してください. (“使用上の注意” c)参照)
nsizefactorl	long	入力	panelfactorl の大きさを示す.
		出力	panelfactorl の大きさとして必要な大きさが返却されます. (“使用上の注意” c)参照)
nfcnzindexl	long nfcnzindexl[n+1]	出力	各 supernode の分解結果に対応する行列 **L** の複数の列ベクトルと行列 **U** のブロック対角部分について, 共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します. この行指標ベクトルを npanelindexl に順番に割り付けたとき *i* 番目の行指標ベクトルの先頭要素が 1 次元配列である npanelindexl の何番目の要素になるかを示します. 結果の格納方法については図 c_dm_vssss-1 を参照してください.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します.
npanelindexl	int npanelindexl [nsizeindex]	出力	各 supernode の分解結果に対応する行列 **L** の複数の列ベクトルと行列 **U** のブロック対角部分について, 共通の行指標ベクトルを持つように圧縮して 2 次元の panel に格納します. この行指標ベクトルを npanelindexl に順番に割り付けます. *i* 番目の supernode に対応する panel の行の指標ベクトルが何番目に割り当てられるかは j = nassign[i-1]から分かります. その先頭位置は nfcnzindexl[j-1]に格納されています. この行指標は post order で並び換えたときの行の番号です. 結果の格納方法については図 c_dm_vssss-1 を参照してください. (“使用上の注意” c)参照)
nsizeindex	long	入力	npanelindexl および npanelindexu の大きさを示す.
		出力	npanelindexl および npanelindexu の大きさとして必要な大きさが返却されます. (“使用上の注意” c)参照)
ndim	int ndim[n][2]	出力	ndim[i-1][0]および ndim[i-1][1]は行列 **L** に関して *i* 番目に割り付けられた panel の 1 次元目と 2 次元目の大きさを示します. ndim[i-1][0]- ndim[i-1][1] および ndim[i-1][1]は行列 **U** に関して割り付けられた panel を転置したときの 1 次元目と 2 次元目の大きさを示します. 結果の格納方法については図 c_dm_vssss-1 を参照してください.
		入力	isw≠1 のとき,初回呼び出しの値を再利用します.
nfcnzfactoru	long nfcnzfactoru[n+1]	出力	構造的に対称な実スパース行列の LU 分解の結果の行列 **U** に関しては, 各 supernode に対応する **U** の行ベクトルは, ブロック対角部分を除いて共通の列指標ベクトルを持つように圧縮し, 転置したものを 2 次元の panel に格納します. この panel を panelfactoru の 1 次元部分配列として順番に割り付けたとき, *i* 番目の panel の先頭要素 panel[0][0]が 1 次元配列の panelfactoru の何番目の要素になるかを示します.

			結果の格納方法については図 c_dm_vssss-1 を参照してください.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します.
panelfactoru	double panelfactoru [nsizefactoru]	出力	各 supernode の分解結果に対応する行列 **U** のブロック対角部分を除いた複数の行ベクトルについて, 共通の列指標ベクトルを持つように圧縮し, 転置して 2 次元の panel に格納します. panel を順番に割り付けます. *i* 番目の supernode に対応する panel が何番目の位置に割り当てられるかは j = nassign[i-1]から分かります.その先頭位置は nfcnzfactoru[j-1]に格納されています. panel ごとに分解結果が格納されます. *i* 番目に割付けられた panel の大きさは{ndim[i-1][0] – ndim[i-1][1]} × ndim[i-1][1]の 2 次元配列と見なせます. *i* 番目の panel の panel[t-1][s-1], s = 1,…, ndim[i-1][0] – ndim[i-1][1], t = 1,…, ndim[i-1][1]に単位上三角行列 **U** のブロック対角部分を除いた部分が行ベクトルを圧縮し, 転置して格納します. 結果の格納方法については図 c_dm_vssss-1 を参照してください. (“使用上の注意” c)参照)
nsizefactoru	long	入力	panelfactoru の大きさを示す.
		出力	panelfactoru の大きさとして必要な大きさが返却されます. (“使用上の注意” c)参照)
nfcnzindexu	long nfcnzindexu[n+1]	出力	各 supernode の分解結果に対応する行列 **U** は対角ブロック部分を除いて複数の行ベクトルについて, 共通の列指標ベクトルを持つように圧縮し, 転置して 2 次元の panel に格納します. 対角ブロック部分も含んだこの列指標ベクトルを npanelindexu に順番に割り付けたとき *i* 番目の列指標ベクトルの先頭要素が 1 次元配列である npanelindexu の何番目の要素になるかを示します. 結果の格納方法については図 c_dm_vssss-1 を参照してください.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します.
npanelindexu	int npanelindexu [nsizeindex]	出力	各 supernode の分解結果に対応する行列 ***U*** の対角ブロックを除いた部分を転置して圧縮して 2 次元の panel に格納します. この列指標ベクトルを対角ブロック部分を含めて npanelindexu に順番に割り付けます. i 番目の supernode に対応する panel の行の指標ベクトルが何番目に割り当てられるかは j = nassign[i-1]から分かります. その先頭位置は nfcnzindexu[j-1]に格納されています. この行指標は post order で並び換えたときの列の番号です. 結果の格納方法については図 c_dm_vssss-1 を参照してください. (“使用上の注意” c)参照)
nposto	int nposto[n]	出力	post order で i 番目の node が行列 **A** の何番目の列番号に対応するかを表す 1 次元ベクトル.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します. (“使用上の注意” d)参照)

sclrow	double sclrow[n]	出力	スケーリング対角行列 $\mathbf{D}_r$. 対角要素が 1 次元配列に格納されます.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します.
sclcol	double sclcol[n]	出力	スケーリング対角行列 $\mathbf{D}_c$. 対角要素が 1 次元配列に格納されます.
		入力	isw ≠ 1 のとき,初回呼び出しの値を再利用します.
epsz	double	入力	分解過程で対角要素の大きさの絶対値の相対判定値
		出力	epsz ≤ 0.0 のときは標準値が設定されます. ("使用上の注意" b)参照)
thepsz	double	入力	ピボットの判定での閾値(threshold). これより大きな値はピボットとして採用します. 見つかればその時点で, ピボット検索は打ち切ります. 例えば 10^{-2} 程度.
		出力	Thepsz ≤ 0.0 のときは 10^{-2} が設定されます. epsz ≥ thepsz > 0.0 のときは, epsz が設定されます.
ipivot	int	入力	スーパーノード内でピボットを行うか, 行う場合どのようなピボットを行うかを指定します. 例えば, 完全ピボットとして 40 を指定. ipivot < 10 : または ipivot ≥ 50: ピボットなし. 10 ≤ ipivot < 20 : 部分ピボット 20 ≤ ipivot < 30 : 対角ピボット 21 : スーパーノード内で対角ピボットがとれないとき, Rook ピボットに変更します. 22 : スーパーノード内で対角ピボットがとれないときは, Rook ピボットに, Rook ピボットが取れないときは完全ピボットに変更します. 30 ≤ ipivot < 40 :Rook ピボット 32 : スーパーノード内で Rook ピボットがとれないとき, 完全ピボットをとります. 40 ≤ ipivot < 50 : 完全ピボット
istatic	int	入力	Pivot を指定したとき, Static Pivot を行うかを示します. 1) = 1 のとき スーパーノード内で指定したピボットが spepsz より大きくないとき, ピボットとして copysign(spepsz, Pivot) で近似します. ピボットの値が 0.0 なら spepsz に近似します. このとき, 以下を設定しなければなりません. a) epsz は epsz の標準値以下にしなければなりません. b) isclitermax は 10 を設定しスケーリングを行う必要があります. c)thepsz ≥ spepsz でなければなりません. d)解の反復改良を行うため irefine = 1 を指定しなければなりません. 2) ≠ 1 のとき Static Pivot は行いません.
spepsz	double	入力	istatic = 1 のとき, Statc Pivot として使う値. 10^{-10} ≥ spepsz ≥ epsz でなければなりません.
		出力	spepsz < epsz のときは, 10^{-10} が設定されます.
irefine	int	入力	LU 分解結果を利用して解を求めるときに, 解の反復改

			良を行うかどうかを示す. 残差ベクトルを計算するときに 4 倍精度で計算します. = 1 : 解の反復改良を行う. 反復改良して得られる残差ベクトル r_k の絶対値の差分がひとつ前の差分の 1/4 より大きくなるまで, 反復改良を行います. ≠ 1: 解の反復改良を行わない. istatic = 1 のとき, irefine = 1 でなければなりません.
epsr	double	入力	解の残差ベクトル **b - Ax** の絶対値が, **b** の絶対値に対して十分小さいかを判定する判定値. epsr ≤ 0.0 のとき 10^{-6} が設定されます.
itermax	int	入力	(≥ 1) 反復改良を行うときの最大反復回数.
iter	int	出力	反復改良を行った回数.
w	double w[nz+n]	作業域	isw = 1, 2 で続けて呼び出すとき,呼び出しの間で必要なデータを受け渡すために使われます. 呼び出しの間で値を変更してはいけません.
iw	int iw[36*n+36+2*nz+3(n+1)]	作業域	isw = 1, 2, 3 で続けて呼び出すとき,呼び出しの間で必要なデータを受け渡すために使われます. 呼び出しの間で値を変更してはいけません.
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になった.	処理を打ち切る.
20200	行および列の均衡化を行う対角行列の対角要素を求める過程で,元の行列 A の行もしくは列にゼロベクトルがあった. 行列が特異である可能性がある.	
20400	LU 分解された行列の対角要素にゼロがあった.	
20500	求めた解ベクトルに対する残差ベクトルのノルムの大きさが方程式 Ax = b の右辺ベクトルのノルムの epsr 倍より大きい. 係数行列は特異に近い可能性がある.	
30000	次のいずれかであった: • n < 1 • nz < 0 • nfcnz[n] ≠ nz + 1 • nsizefactorl < 1 • nsizefactoru < 1 • nsizeindex < 1 • isw < 1 • isw > 3 • irefine = 1 のとき itermax < 1	

コード	意　味	処理内容
30100	nperm に指定した置換行列が正しくない.	処理を打ち切る.
30200	nrow[j-1]に格納された *i* 列目の行指標 k が k < 1 または k > n	
30300	*i* 列目の行指標の数 nfcnz[i] – nfcnz[i-1] > n	
30500	istatic = 1 のとき満たすべき条件をみたしていない. epsz は標準値 $16\boldsymbol{u}$ より大きかった. または isclitermax < 10 であった. または irefine≠1 であった. または spepsz > thepsz であった. または spepsz > 10^{-10} であった.	
30700	行列 ***A*** が構造的に対称でない.	
31000	panelfactorl の大きさ nsizefactorl または npanelindexl および npanelindexu の大きさ nsizeindex または panelfactoru の大きさ nsizefactoru が小さすぎる.	nsizefactorl または nsizeindex または nsizefactoru で指定された大きさで panelfactorl または npanelindexl または npanelindexu または panelfactoru を割り付けて isw = 2 として再度呼び出す.

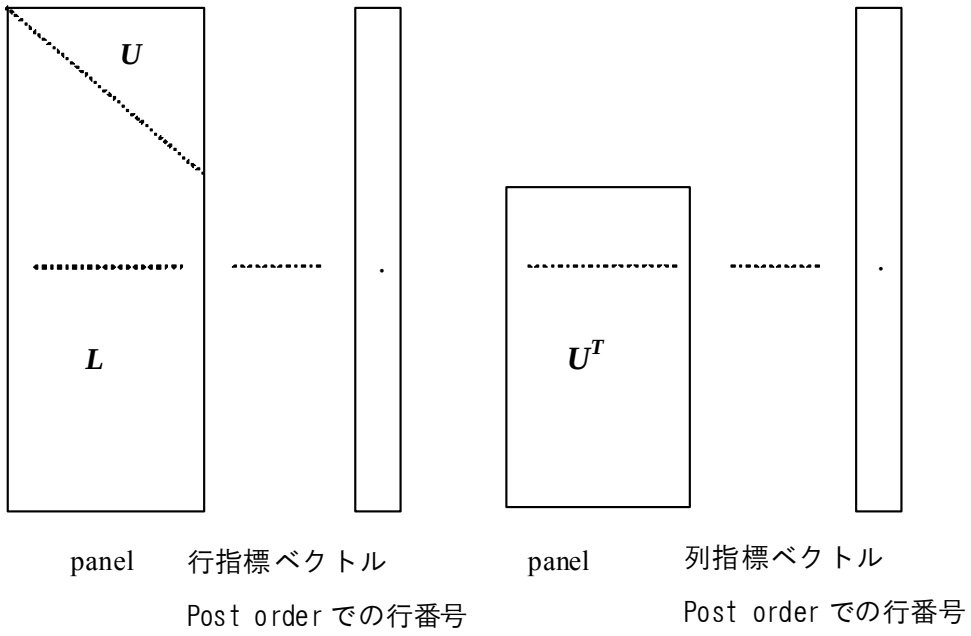


図 c_dm_vssss-1 分解結果の格納概念図

j = nassign[i-1]　→ *i* 番目の supernode は *j* 番目に格納されます.

p = nfcnzfactorl[j-1] → *j* 番目の panel は panelfactorl の *p* 番目の要素から ndim[j-1][0] × ndim[j-1][1] の長さを占めます.

q = nfcnzindexl[j-1] → *j* 番目の panel の行指標を表すベクトルは npanelindexl の *q* 番目の要素から ndim[j-1][0] の長さを占めます.

panel は大きさ ndim[j-1][0] × ndim[j-1][1] の配列と見なせます.

panel[t-1][s-1], s ≥ t, s = 1,…, ndim[j-1][0], t = 1,…, ndim[j-1][1] に分解結果の下三角行列 **L** が格納されます.

panel[t-1][s-1], s < t, s = 1,…, ndim[j-1][1], t = 1,…, ndim[j-1][1] に単位上三角行列 **U** の対角ブロック部分が対角要素を除き格納されます.

u = nfcnzfactoru[j-1] → *j* 番目の panel は panelfactoru の *u* 番目の要素から(ndim[j-1][0] – ndim[j-1][1]) × ndim[j-1][1] の長さを占めます.

v = nfcnzindexu[j-1] → j 番目の panel の列指標を表すベクトルは npanelindexu の *v* 番目の要素から ndim[j-1][0]の長さを占めます.

panel は大きさ(ndim[j-1][0] – ndim[j-1][1]) × ndim[j-1][1] の配列と見なせます.

panel[y-1][x-1], x = 1,…, ndim[j-1][0] – ndim[j-1][1], y = 1,…, ndim[j-1][1]

に分解結果の単位上三角行列 **U** の転置行列 $\mathbf{U}^{\mathrm{T}}$ の対角ブロック部分を除いた部分が格納されます.

指標の値は行列 **A** の列番号をもつ node を post order で並び換えた $\mathbf{QAQ}^{\mathrm{T}}$ での列番号を表します.

3. 使用上の注意

a)

直交行列である置換行列 **P** の要素 $\mathbf{p}_{ij} = 1$ のとき, nperm[i-1] = j と表現します.
逆は以下のようにして求めることができます.

```
for (i = 1; i <= n; i++) {
 j = nperm[i-1];
 nperminv[j-1] = i;
}
```

Fill-Reducing Ordering は METIS などを使って求めることができます.
詳細は“付録 参考文献一覧表”の[43], [44]を参照願います.

b)

分解過程で対角要素の大きさの絶対値の相対零判定値にある値を設定したとすると, この値は次の意味を持っています. すなわち, LU 分解の過程で, 対角要素の大きさの絶対値がその値より小さくなった場合に, その値を相対的に零と見なし, icon = 20000 として処理を打ち切ります. epsz の標準値は, 丸め誤差の単位を *u* としたとき, epsz = 16*u* です.
なお, 対角要素の大きさの絶対値が小さくなっても, 処理を続行させたい場合には, epsz に極小の値を与えれば良いのですが, 結果の精度は保証されません.
Static Pivot を指定した場合, 対角要素が spepsz より小さいとき, spepsz で近似して LU 分解を行います. このとき, 解の反復改良を指定しなければなりません.

c)

分解結果を格納する配列 panelfactorl, npanelindexl, panelfactoru, npanelindexu の必要な大きさは,事前には分かりません. 十分大きな配列を割り当てるか, 本関数を呼び出して symbolic decomposition を行った解析の結果を使って割り当てを行うことができます.

例えば, 大きさ 1 の 1 次元配列などを割付けます. そしてその大きさ 1 などの小さな値を nsizefactorl, nsizeindex, nsizefactoru に指定して, isw = 1 で呼び出します.
symbolic decomposition を行い, icon = 31000 で終了し, nsizefactorl, nsizeindex, nsizefactoru に必要な大きさが返却されます. 必要な大きさの配列を割付け直して引数に指定して, isw = 2 で呼び出すことで, symbolic decomposition 以降の処理を続けることができます. 使用例を参照願います.

d)

列番号に対応するノードを考えます. これを post order で順番を並び換えたときの番号の対応関係が nposto に格納されています. post order の *i* 番目のノードが並び換える前の何番目のノードに対応するかを表します. j = nposto[i-1]は *j* 番目であることを表します.
a)同様にこれは直交行列である置換行列 **Q** を表し,行列 **A** を $\mathbf{QAQ}^{\mathbf{T}}$ と並び換えることに相当します.
逆変換 $\mathbf{Q}^{\mathbf{T}}$ は以下のようにして求めることができます.

```
for (i = 1; i <= n; i++) {
 j = nposto[i-1];
 npostoinv[j-1] = i;
}
```

e)

連立1次方程式**Ax** = **b**は, c_dm_vsssluで構造的に対称な実スパース行列**A**をLU分解して, 分解結果を利用して引き続いてc_dm_vssluxを呼び出して解くこともできます.

4. 使用例

連立 1 次方程式 **Ax** = **f** を解きます.
行列は境界条件として立方体の境界で 0 となる楕円型の偏微分方程式に有限差分法を適用して生成されたものを利用します.

$$-\Delta u + a\nabla u + cu = f$$

ここで $a = (a_1, a_2, a_3)$.
行列は関数 init_mat_diag によって生成されます.
対角形式格納法で生成し,圧縮列格納法で格納します. 結果生成された構造的に対称な行列 **A** に関する連立 1 次方程式を解きます.
(並列実行を行うスレッド数は環境変数(OMP_NUM_THREADS)などで指定できます.
例えば, 4 プロセッサーのシステムで 4 つのスレッドで並列実行するときは, OMP_NUM_THREADS を 4 に設定して実行します.)

```
/* **EXAMPLE** */
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <malloc.h>
#include <omp.h>
#include "cssl.h"

#define  NORD  39
#define  NX    NORD
#define  NY    NORD
#define  NZ    NORD
#define  N     (NX * NY * NZ)
#define  NXY   (NX * NY)
#define  K     (N + 1)
#define  NDIAG  7
#define  NALL  (NDIAG * N)
```

```
#define  IWL   (36 * N + 36 + 2 * NALL + 3 * (N + 1))
#define  IPRINT  0

void  init_mat_diag(double, double, double, double, double*, int*, int, int,
                    int, double, double, double, int, int, int);
double  errnrm(double*, double*, int);

int MAIN__() {

  int  nofst[NDIAG];
  double  diag[NDIAG][K], diag2[NDIAG][K];
  double  c[K * NDIAG], wc[K * NDIAG];
  int  nrowc[K * NDIAG], nfcnzc[N + 1], iwc[K * NDIAG][2];
  double  w[NDIAG * N + N];
  int  nperm[N],
       nposto[N], ndim[N][2],
       nassign[N],
       iw[IWL];
  double  *panelfactorl, *panelfactoru;
  int  *npanelindexl,
       *npanelindexu;
  double  dummyfl, dummyfu;
  int  ndummyil, ndummyiu;
  long  nsizefactorl, nsizeindex,
        nsizefactoru,
        nfcnzfactorl[N + 1],
        nfcnzfactoru[N + 1],
        nfcnzindexl[N + 1],
        nfcnzindexu[N + 1];
  double  x[N], b[N], solex[N];
  int  i, j, nbase, length, numnzc, ntopcfgc, ncol, nnzc;
  double  va1, va2, va3, vc, xl, yl, zl;

  double  thepsz,
          epsr,
          sepsz,
          sclrow[N], sclcol[N];
  double  epsz, err;

  int  ipivot, istatic,
       isclitermax,
       irefine, itermax, iter, icon;
  int  iordering, isw, nsupnum;



  printf("    DIRECT METHOD\n");
  printf("    FOR SPARSE STRUCTURALLY SYMMETRIC REAL MATRICES\n");
  printf("    IN COMPRESSED COLUMN STORAGE\n\n");
```

```
for (i = 0; i < N; i++) {
  solex[i] = 1.0;
}
printf("    EXPECTED SOLUTIONS\n");
printf("    X(1) = %19.16lf X(N) = %19.16lf\n\n", solex[0], solex[N - 1]);

va1 = 1.0;
va2 = 2.0;
va3 = 3.0;
vc = 4.0;
xl = 1.0;
yl = 1.0;
zl = 1.0;
init_mat_diag(va1, va2, va3, vc, (double *)diag, nofst,
              NX, NY, NZ, xl, yl, zl, NDIAG, N, K);

for (i = 0; i < NDIAG; i++) {
  for (j = 0; j < K; j++) {
    diag2[i][j] = 0;
  }
}

for (i = 0; i < NDIAG; i++) {

  if (nofst[i] < 0) {
    nbase = -nofst[i];
    length = N - nbase;
    for (j = 0; j < length; j++) {
      diag2[i][j] = diag[i][nbase + j];
    }
  } else {
    nbase = nofst[i];
    length = N - nbase;
    for (j = 0; j < length; j++) {
      diag2[i][nbase + j] = diag[i][j];
    }
  }

}

numnzc = 0;

for (j = 0; j < N; j++) {
  ntopcfgc = 1;

  for (i = NDIAG - 1; i >= 0; i--) {

    if (diag2[i][j] != 0.0) {
```

```
      ncol = (j + 1) - nofst[i];
      c[numnzc] = diag2[i][j];
      nrowc[numnzc] = ncol;

      if (ntopcfgc == 1) {
        nfcnzc[j] = numnzc + 1;
        ntopcfgc = 0;
      }

      numnzc++;

    }
  }
}

nfcnzc[N] = numnzc + 1;
nnzc = numnzc;

c_dm_vmvscc(c, nnzc, nrowc, nfcnzc, N, solex,
            b, wc, (int *)iwc, &icon);


for (i = 0; i < N; i++) {
  x[i] = b[i];
}
iordering = 0;
isclitermax = 10;
isw = 1;
epsz = 1.0e-16;
nsizefactorl = 1;
nsizefactoru = 1;
nsizeindex = 1;
thepsz = 1.0e-2;
epsr = 1.0e-8;
sepsz = 1.0e-10;
ipivot = 40;
istatic = 1;
irefine = 1;
itermax = 10;

c_dm_vssss(c, nnzc, nrowc, nfcnzc, N,
           isclitermax, iordering,
           nperm, isw,
           x,
           nassign,
           &nsupnum,
           nfcnzfactorl, &dummyfl,
           &nsizefactorl, nfcnzindexl,
```

```
            &ndummyil, &nsizeindex, (int *)ndim,
            nfcnzfactoru, &dummyfu,
            &nsizefactoru,
            nfcnzindexu, &ndummyiu,
            nposto,
            sclrow, sclcol,
            &epsz,
            &thepsz,
            ipivot, istatic, &sepsz,
            irefine, epsr, itermax, &iter,
            w, iw, &icon);

printf("    ICON=%6d NSIZEFACTORL=%9ld NSIZEFACTORU=%9ld NSIZEINDEX=%9ld\n",
       icon, nsizefactorl, nsizefactoru, nsizeindex);
printf("    NSUPNUM=%d\n\n", nsupnum);

panelfactorl = (double *)malloc(sizeof(double) * nsizefactorl);
panelfactoru = (double *)malloc(sizeof(double) * nsizefactoru);
npanelindexl = (int *)malloc(sizeof(int) * nsizeindex);
npanelindexu = (int *)malloc(sizeof(int) * nsizeindex);

isw = 2;
c_dm_vssss(c, nnzc, nrowc, nfcnzc, N,
           isclitermax, iordering,
           nperm, isw,
           x,
           nassign,
           &nsupnum,
           nfcnzfactorl, panelfactorl,
           &nsizefactorl, nfcnzindexl,
           npanelindexl, &nsizeindex, (int *)ndim,
           nfcnzfactoru, panelfactoru,
           &nsizefactoru,
           nfcnzindexu, npanelindexu,
           nposto,
           sclrow, sclcol,
           &epsz,
           &thepsz,
           ipivot, istatic, &sepsz,
           irefine, epsr, itermax, &iter,
           w, iw, &icon);


err = errnrm(solex, x, N);

printf("    COMPUTED VALUES\n");
printf("    X(1) = %19.16lf X(N) = %19.16lf\n\n", x[0], x[N - 1]);
printf("    ICON = %6d\n\n", icon);
printf("    N = %d :: NX = %d  NY = %d NZ = %d\n\n", N, NX, NY, NZ);
```

```
  printf("    ERROR = %10.3le\n", err);
  printf("    ITER=%d\n\n\n", iter);

  if (err < 1.0e-8 && icon == 0) {
    printf("    ********** OK **********\n");
  } else {
    printf("    ********** NG **********\n");
  }

  free(panelfactorl);
  free(panelfactoru);
  free(npanelindexl);
  free(npanelindexu);

  return(0);
}

/* ========================================
       INITIALIZE COEFFICIENT MATRIX
   ======================================== */
void init_mat_diag(double va1, double va2, double va3, double vc, double *d_l,
                   int *offset, int nx, int ny, int nz, double xl, double yl,
                   double zl, int ndiag, int len, int ndivp) {

  if (ndiag < 1) {
    printf("SUB FUNCTION INIT_MAT_DIAG:\n");
    printf(" NDIAG SHOULD BE GREATER THAN OR EQUAL TO 1\n");
    return;
  }

#pragma omp parallel default(shared)
{
  int  ndiag_loc, i, j, l, nxy, i0, j0, k0, js;
  double  hx, hy, hz, hx2, hy2, hz2;

/* NDIAG CANNOT BE GREATER THAN 7 */
  ndiag_loc = ndiag;
  if (ndiag > 7) ndiag_loc = 7;

/* INITIAL SETTING */
  hx = xl / (nx + 1);
  hy = yl / (ny + 1);
  hz = zl / (nz + 1);

#pragma omp for
  for (i = 0; i < ndivp * ndiag; i++) {
    d_l[i] = 0.0;
  }
```

```
  nxy = nx * ny;

/* OFFSET SETTING */
#pragma omp single
  {
    l = 0;
    if (ndiag_loc >= 7) {
      offset[l] = -nxy;
      l++;
    }
    if (ndiag_loc >= 5) {
      offset[l] = -nx;
      l++;
    }
    if (ndiag_loc >= 3) {
      offset[l] = -1;
      l++;
    }
    offset[l] = 0;
    l++;
    if (ndiag_loc >= 2) {
      offset[l] = 1;
      l++;
    }
    if (ndiag_loc >= 4) {
      offset[l] = nx;
      l++;
    }
    if (ndiag_loc >= 6) {
      offset[l] = nxy;
    }
  }

/* MAIN LOOP */
#pragma omp for
  for (j = 0; j < len; j++) {
    js = j + 1;

/* DECOMPOSE JS-1 = (K0-1)*NX*NY+(J0-1)*NX+I0-1 */
    k0 = (js - 1) / nxy + 1;
    if (k0 > nz) {
      printf("ERROR; K0.GH.NZ \n");
      continue;
    }
    j0 = (js - 1 - nxy * (k0 - 1)) / nx + 1;
    i0 = js - nxy * (k0 - 1) - nx * (j0 - 1);
    l = 0;

    if (ndiag_loc >= 7) {
```

```
      if (k0 > 1) d_l[l * ndivp + j] = -(1.0 / hz + 0.5 * va3) / hz;
      l++;
    }
    if (ndiag_loc >= 5) {
      if (j0 > 1) d_l[l * ndivp + j] = -(1.0 / hy + 0.5 * va2) / hy;
      l++;
    }
    if (ndiag_loc >= 3) {
      if (i0 > 1) d_l[l * ndivp + j] = -(1.0 / hx + 0.5 * va1) / hx;
      l++;
    }
    hx2 = hx * hx;
    hy2 = hy * hy;
    hz2 = hz * hz;
    d_l[l * ndivp + j] = 2.0 / hx2 + vc;
    if (ndiag_loc >= 5) {
      d_l[l * ndivp + j] += 2.0 / hy2;
      if (ndiag_loc >= 7) {
        d_l[l * ndivp + j] += 2.0 / hz2;
      }
    }
    l++;
    if (ndiag_loc >= 2) {
      if (i0 < nx) d_l[l * ndivp + j] = -(1.0 / hx - 0.5 * va1) / hx;
      l++;
    }
    if (ndiag_loc >= 4) {
      if (j0 < ny) d_l[l * ndivp + j] = -(1.0 / hy - 0.5 * va2) / hy;
      l++;
    }
    if (ndiag_loc >= 6) {
      if (k0 < nz) d_l[l * ndivp + j] = -(1.0 / hz - 0.5 * va3) / hz;
    }
  }

}

  return;
}

/* ======================================
  * SOLUTE ERROR
  * | X1 - X2 |
   ====================================== */
double errnrm(double *x1, double *x2, int len) {

  double  s, ss, rtc;
  int  i;
```

```
  s = 0.0;
  for (i = 0; i < len; i++) {
    ss = x1[i] - x2[i];
    s += ss * ss;
  }

  rtc = sqrt(s);
  return(rtc);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VSSSS の項目及び[2], [19], [22], [48], [63], [68], [69]を参照してください.

c_dm_vtdevc

実 3 重対角行列の固有値及び固有ベクトル
`ierr = c_dm_vtdevc(d, sl, su, n, nf, nl, ivec, &etol, &ctol, nev, e, maxne, ev, k, m, &icon);`

1. 機能

実 3 重対角行列の指定された固有値を求めます. 指定に応じて対応する固有ベクトルを求めます.

$$\mathbf{Tx} = \lambda\mathbf{x}$$

T は n 次元の実 3 重対角行列です. 実 3 重対角行列 **T** は, 次の条件を満たすものとします.

$$l_i\, u_{i-1} > 0\ , \quad i = 2, \ldots , n,$$

ここで, 3 重対角行列 **T** の要素を t_{ij} としたとき, d_i は対角要素を, $l_i = t_{i,i-1}$, $u_i = t_{i,i+1}$ は副対角要素を表します. ただし, $l_1 = u_n = 0$.

$$(\mathbf{Tv})_i = l_i v_{i-1} + d_i v_i + u_i\, v_{i+1}, \quad i = 1, 2, \ldots , n$$

2. 引数

呼出し形式:

```
ierr = c_dm_vtdevc(d, sl, su, n, nf, nl, ivec, &etol, &ctol, nev, e, maxne,
            (double*)ev, k, (int*)m, &icon);
```

引数の説明:

`d`	`double d[n]`	入力	**T** の対角要素 d_i を格納します.
`sl`	`double sl[n]`	入力	**T** の下副対角要素を `sl[i–1]`$= l_i, i = 2, \ldots , n$ のように格納します. `sl[0]` $= 0$.
`su`	`double su[n]`	入力	**T** の上副対角要素を `su[i–1]`$= u_i, i = 1, \ldots , n–1$ のように格納します. `su[n–1]` $= 0$.
`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ます. `ivec` $\neq 1$ のとき固有値のみ求めます.
`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.
		出力	`etol` $< 3 \times 10^{-16}$ のときは, `etol` $= 3 \times 10^{-16}$ が標準値として設定されます. (“使用上の注意” b)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`). $10^{-6} \geq$ `ctol` $\geq$ `etol`.
		出力	`ctol` $> 10^{-6}$ のときは, `ctol` $= 10^{-6}$ と設定されます. `ctol` $<$ `etol` のときは, `ctol` $= 10 \times$ `etol` が標準値として設定されます. (“使用上の注意” b)参照)

nev	int nev[5]	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. nev[3]は, 求めた固有値の内最初の固有値の番号. nev[4]は, 求めた固有値の内最後の固有値の番号.
e	double e[maxne]	出力	固有値が格納されます. 求められた固有値は e[i–1], i = 1, ... , nev[2]に格納されます.
maxne	int	入力	計算できる固有値の最大個数. 配列 e の大きさ. 多重度 m の固有値がいくつかあると考えられるとき, n を上限として $\mathtt{nl} - \mathtt{nf} + 1 + 2 \times m$ を設定する必要があります. nev[2] > maxne のとき, 固有ベクトルの計算はできません. ("使用上の注意" c)参照)
ev	double ev[maxne][k]	出力	ivec = 1 のとき, 固有値に対応して固有ベクトルが格納されます. 求められた固有ベクトルは ev[i–1][j–1], i = 1,...,nev[2], j = 1, ... ,n に格納されます.
k	int	入力	ev の整合寸法 (≥ n)
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します.
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.
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"c)参照)
30000	次のいずれかであった. • n < 1 • k < 1 • k < n • nf < 1 • nl < nf • nl > n • maxne < nl–nf+1	処理を打ち切る.
30100	sl[i] × su[i–1] ≤ 0, 行列が対称化できなかった.	

3. 使用上の注意

a) 本関数の適用条件について

本関数は $l_i\, u_{i-1} \geq 0$ のみを要求します. このため, 例えば(1)左側の一般化固有値問題は, 対角行列 **D** の要素が全て正ならば, 両辺に左から $\mathbf{D}^{-1}$ をかけて, 本関数で扱える三重対角行列の固有値問題の形に変換できます.

$$\mathbf{Tx} = \lambda\mathbf{Dx} \longrightarrow \mathbf{T'x} = \lambda\mathbf{x}\ ,\ \mathbf{T'} = \mathbf{D}^{-1}\mathbf{T} \qquad (1)$$

また, 非対称三重対角行列の固有値問題 $\mathbf{Tx} = \lambda\mathbf{x}$ は対角行列を使って以下にしめす, 対称な一般化固有値問題に変換することができます.

$$\mathbf{DTx} = \lambda\mathbf{Dx} \tag{2}$$

ここで, 対角行列 **D** の対角要素は $d_1 = 1$, $d_i = u_{i-1}\, d_{i-1}/l_i$, $i = 2, \ldots, n$. このことは, 対称三重対角行列の一般化固有値問題も本関数で扱う形に変換できることを意味します. ただ, 変換に際し $\mathbf{D}^{-1}$ を乗することでスケーリング問題を引き起こす場合は, (2)と等価な以下の対称な問題を考えるほうが望ましいです.

$$\mathbf{D}^{1/2}\mathbf{TD}^{-1/2}\mathbf{w} = \lambda\mathbf{w}\ ,\ \mathbf{w} = \mathbf{D}^{1/2}\mathbf{x}$$

b) `etol` 及び `ctol`

本ルーチンは, 求める固有値を交わりのない順序付けられた集合に分けて独立に計算することにより並列化しています.

ε = `etol` としたとき, 連続する固有値λ_j, $j = s, s+1, \ldots, s+k$, $(k \geq 0)$に対して,

$$\frac{|\lambda_i - \lambda_{i-1}|}{1 + \max(|\lambda_{i-1}|, |\lambda_i|)} \leq \varepsilon\ , \tag{3}$$

$i = s, s+1, \ldots, s+k$ となる i に対して(3)を満たし, $i = s-1$ 及び $i = s+k+1$ について(3)を満たさないとき, これらの固有値 λ_j, $j = s-1, s, \ldots, s+k$ は数値的に多重であると見なされます.

`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ます.

(3)が ε =`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ます.

ε =`etol` で異なる固有値と見なされた連続する固有値λ_m, $m = t-1, t, \ldots, t+k$, $(k \geq 0)$に対して, ε =`ctol` としたとき, $i = t, t+1, \ldots, t+k$ について(3)を満たし, $i = t-1$ 及び $i = t+k+1$ について(3)を満たさないとき, これらの異なる固有値 λ_m, $m = t-1, t, \ldots, t+k$ を近似的に多重(cluster)と見なします. 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ます.

c) `maxne`

`maxne` に, 計算できる固有値の最大個数を指定できます. `ctol` を大きくすると, cluster の大きさが大きくなり, 計算する固有値の総数が `maxne` を超えるかもしれません. この場合, `ctol` を小さくするか, `maxne` を大きくする必要があります.

計算する固有値の総数が `maxne` を超えた場合, `icon`＝20000 を返却します. 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ません. 固有値は求められていますが, 多重度分だけ繰り返して格納はされていません.

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ます.

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` は求める固有値の総数を nt (nt = `nl` – `nf` + 1)として nt で十分です. 多重な固有値がいくつかあり, 多重度が m と考えられるときは, 少なくとも `maxne` は $nt + 2 \times m$ とする必要があります.

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値が `nev[2]`に返却されます. この値で領域を確保して再度呼び出すことで計算を続けることができます.

4. 使用例

実 3 重対角行列の固有値を計算します.

```
#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 P1        (70)
#define Q1        (100)
```

```
#define N         (P1*Q1)
#define K         (N+1)
#define N0        (6001)
#define N1        (7000)
#define NE        (N1-N0+1)
#define MAX_CLUS  (2*Q1)
#define MAXNE     (NE+MAX_CLUS)
#define NW        (2*N+2)

MAIN__()
{
  double d[N], sl[N], su[N], e[MAXNE], ev[MAXNE][K], w[NW];
  double tmp, error, etol, ctol;
  int    m[2][MAXNE], nev[5], nf, nl, ivec, icon;
  int    i, j, l, ii;

  etol=3e-16;
  ctol=5e-12;
  j = (P1+1)/2;
  d[j-1] = 0.0;
  for (i=1; i<=j-1; i++) {
    sl[i+1-1]   = 1.0;
    su[i-1]     = 1.0;
    sl[j+i-1]   = 1.0;
    su[j+i-2]   = 1.0;
    d[i-1]      = (double)(j-i);
    d[j*2-i-1]  = d[i-1];
  }
  sl[0]    = 0.0;
  su[P1-1] = 0.0;

  for (l=2; l<=Q1; l++) {
    ii = (l-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]   = 0.0;
  su[N-1] = 0.0;

  nf      = N0;
  nl      = N1;
  ivec    = 1;

  c_dm_vtdevc(d, sl, su, N, nf, nl, ivec, &etol, &ctol, nev, e, MAXNE, (double*)ev, K,
              (int*)m, &icon);

  printf("icon   = %d\n", icon);
  printf("nev[0] = %d\n", nev[0]);
  printf("nev[1] = %d\n", nev[1]);
  printf("nev[2] = %d\n", nev[2]);
  printf("nev[3] = %d\n", nev[3]);
  printf("nev[4] = %d\n", nev[4]);
  error = tmp = 0.0;
  for (i=0; i<nev[2]; i++) {
    for (j=0; j<N; j++) {
      w[j+1] = ev[i][j];
    }
    w[0]   = 0.0;
    w[N+1] = 0.0;

    for (j=0; j<N; j++) {
      tmp = sl[j]*w[j]+d[j]*w[j+1]+su[j]*w[j+2]-e[i]*w[j+1];
      error = max(fabs(tmp/(fabs(e[i])+1)), error);
    }
  }

  printf("maximum element error in ||T*x-eig*x|| = %e\n", tmp);

  return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VTDEVC の項目及び[20], [61], [71], [80]を参照してください.

c_dm_vtfqd

非対称または不定値のスパース行列の連立 1 次方程式 (TFQMR 法, 対角形式格納法)

`ierr = c_dm_vtfqd(a, k, ndiag, n, nofst, b,`
`itmax, eps, iguss, x, &iter,`
`&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)で解きます.

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

$n \times n$ の係数行列 **A** は, 対角形式の格納法で格納します. ベクトル **b** および **x** は n 次元ベクトルです.

反復解法の収束および利用指針に関しては, “SSL II 拡張機能使用手引書 II 第 I 部 概説 第 4 章 連立 1 次方程式の反復解法と収束性” を参照してください.

2. 引数

呼出し形式:

```
ierr = c_dm_vtfqd((double*)a, k, ndiag, n, nofst, b, itmax, eps, iguss, x,
            &iter, &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で格納します.
k	int	入力	配列 a の 2 次元目の大きさ. (≥ 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	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します. 上対角ベクトル列は正, 下対角ベクトル列は負の値で表します.
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します.
itmax	int	入力	TFQMR 法の反復回数の上限値. 2000 程度で十分です.
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} が設定されます.
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ません. 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します.
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ます.
		出力	解ベクトルが格納されます.
iter	int	出力	TFQMR 法での実際の反復回数.
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.
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 • n > k • ndiag < 1 • itmax ≤ 0	処理を打ち切る.
32001	|nofst[i]| > n–1	

3. 使用上の注意

a) eps について

本関数は, 残差のユークリッドノルムが最初の残差のユークリッドノルムと eps の積以下になったとき解が収束したと見なします. 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 の積に等しくなります.

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ります. 対角ベクトル列を配列 a に格納する順序に制限は特にありません.

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す. しかし, 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す.

4. 使用例

実スパース行列の連立 1 次方程式を解きます.

```
#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 a[UBANDW+LBANDW+1][NMAX], b[NMAX], x[NMAX];
  double one=1.0, bcoef=10.0, eps=1.e-6;
  int    ierr, icon, ndiag, nub, nlb, n, i, j, k;
  int    itmax, iguss, iter;
  int    nofst[UBANDW + LBANDW + 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;
  ierr = c_dm_vtfqd ((double*)a, k, ndiag, n, nofst, b, itmax, eps,
                  iguss, x, &iter, &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

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VTFQD の項目を参照してください.

c_dm_vtfqe

非対称または不定値のスパース行列の連立 1 次方程式 (TFQMR 法, ELLPACK 形式格納法)

`ierr = c_dm_vtfqe(a, k, iwidt, n, icol, b,`
`itmax, eps, iguss, x, &iter,`
`&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)で解きます.

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

$n \times n$ の係数行列 **A** は, ELLPACK 形式の格納法で格納します. ベクトル **b** および **x** は n 次元ベクトルです.

反復解法の収束および利用指針に関しては, “SSL II 拡張機能使用手引書 II 第 I 部 概説 第 4 章 連立 1 次方程式の反復解法と収束性” を参照してください.

2. 引数

呼出し形式:

```
ierr = c_dm_vtfqe((double*)a, k, iwidt, n, icol, b, itmax, eps, iguss, x,
            &iter, &icon);
```

引数の説明:

a	double a[iwidt][k]	入力	係数行列の非零要素を ELLPACK 形式で格納します.
k	int	入力	配列 a および icol の 2 次元目の大きさ. (≥ 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. 配列 a および icol の 1 次元目の大きさ.
n	int	入力	行列 **A** の次数 n.
icol	int icol[iwidt][k]	入力	ELLPACK 形式で使用される列指標で, 配列 a の対応する要素がいずれの列ベクトルに属すかを示します.
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します.
itmax	int	入力	TFQMR 法の反復回数の上限値. 2000 程度で十分です.
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} が設定されます.
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ません. 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します.
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ます.
		出力	解ベクトルが格納されます.
iter	int	出力	TFQMR 法での実際の反復回数.
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.
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 • n > k • iwidt < 1 • itmax ≤ 0	処理を打ち切る.
30001	バンド巾がゼロ.	

3. 使用上の注意

a) eps について

本関数は, 残差のユークリッドノルムが最初の残差のユークリッドノルムと eps の積以下になったとき解が収束したと見なします. 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 の積に等しくなります.

4. 使用例

実スパース行列の連立 1 次方程式を解きます.

```
#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 a[UBANDW+LBANDW+1][NMAX], b[NMAX], x[NMAX];
  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, iter, i, j, ix;
  int    icol[UBANDW + LBANDW + 1][NMAX];

  /* initialize matrix and vector */
  nub   = UBANDW;
  nlb   = LBANDW;
  iwidt = UBANDW + LBANDW + 1;
  n     = NMAX;
  k     = NMAX;

  for (i=0; i<n; i++) b[i] = bcoef;

  for (i=0; i<iwidt; 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;
  ierr = c_dm_vtfqe ((double*)a, k, iwidt, n, (int*)icol, b, itmax,
                     eps, iguss, x, &iter, &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

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VTFQE の項目を参照してください.

c_dm_vtrid

実対称行列の実対称三重対角行列への変換 (ハウスホルダー法)
`ierr = c_dm_vtrid (a, k, n, d, sl, &icon);`

1. 機能

実対称行列 **A** を Householder 変換で三重対角化します.

$$\mathbf{T} = \mathbf{Q}^{\mathrm{T}}\mathbf{A}\mathbf{Q}$$

A は $n \times n$ 実対称行列, **Q** は $n \times n$ 直交行列, **T** は三重対角行列です.

2. 引数

呼出し形式:

```
ierr = c_dm_vtrid((double*)a, k, n, d, sl, &icon);
```

引数の説明:

a	double a[n][k]	入力	a の上三角部分{a[*i*-1][*j*-1], $i \le j$}に実対称行列 **A** の上三角部分$\{a_{ij} \mid i \le j\}$を格納します.
		出力	a[*i*-1][*j*-1], *i*=1, … ,*n*, *j*=1, … ,*n*-2 の上三角部分{a[*i*-1][*j*-1], $i \le j$}に三重対角化を行うために利用した Householder 変換に関する情報が格納されます. 演算後, 下三角部分の値は保証されません. ("使用上の注意"a)参照)
k	int	入力	配列 a の整合寸法 (≥ n).
n	int	入力	実対称行列 **A** の次数 *n*.
d	double d[n]	出力	三重対角化された三重対角行列の対角要素を格納します.
sl	double sl[n]	出力	三重対角化された三重対角行列の下副対角要素をsl[*i*-1], *i* = 2, ... , *n* に格納します. sl[0] = 0.
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, k < n.	処理を打ち切る.

3. 使用上の注意

a) a について

k=1, ... , *n*-2 まで, 以下の変換を繰り返して三重対角化します.

$$\mathbf{A}^{k} = \mathbf{Q}_k^{\mathrm{T}}\mathbf{A}^{k-1}\mathbf{Q}_k \ , \quad \mathbf{A}^{0} = \mathbf{A}$$

$\mathbf{b}^{\mathrm{T}} = (0, \ldots, 0, \mathbf{A}^{k-1}(k+1, k), \ldots, \mathbf{A}^{k-1}(n, k))$ とおきます .($\mathbf{A}^{k-1}(i, j)$は $\mathbf{A}^{k-1}$ の i, j 要素)

$\mathbf{b}^{\mathrm{T}} = (0, \ldots, 0, b_{k+1}, \ldots, b_n)$

$\mathbf{b}^{\mathrm{T}} \cdot \mathbf{b} = S^2$ とし, $\mathbf{w}^{\mathrm{T}} = (0, \ldots, 0, b_{k+1}+S, b_{k+2}, \ldots, b_n)$とおきます.

b_{k+1} と S は同符号となるように S の符合を選びます.

すると, $\mathbf{Q}_k = \mathrm{I} - \alpha \mathbf{w} \cdot \mathbf{w}^{\mathrm{T}}, \alpha = \frac{1}{\mathrm{S}^2 + |b_{k+i}\,\mathrm{S}|}$ と表せます.
a[*k*-1][*i*-1] に $\mathbf{w}$ (*i*-1) (*i*=*k*+1, … , *n*)また a[*k*-1][*k*-1]にαを格納します.

4. 使用例

固有値が分かっている実対称行列の三重対角化を行います.

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


#define N               2000
#define K               N
#define NE              N
#define MAX_NEV         NE

MAIN__()
{
    double a[N][K], b[N][K], c[N][K], d[N][K], ac[N][K];
    double dd[N], sld[N], sud[N];
    double eval[MAX_NEV], evec[MAX_NEV][K];
    double pai, coef, eval_tol, clus_tol;
    int    nev[5], mult[2][MAX_NEV];
    int    i, j, nf, nl, ivec, icon;

    pai  = 4.0 * atan(1.0);
    coef = sqrt(2.0/(N+1));

    for (j=0; j<N; j++) {
      for (i=0; i<N; i++) {
        d[j][i] = coef*sin(pai/(N+1)*(i+1)*(j+1));
      }
    }

    for (j=0; j<N; j++) {
      for (i=0; i<N; i++) {
        if (i == j) { c[j][i]=i+1; }
        else        { c[j][i]=0.0; }
      }
    }

    c_dm_vmggm ((double*)d, K, (double*)c, K, (double*)b, K, N, N, N, &icon);
    c_dm_vmggm ((double*)b, K, (double*)d, K, (double*)a, K, N, N, N, &icon);

    for (i=0; i<N; i++) {
      for (j=i; j<N; j++) {
        ac[i][j] = a[i][j];
      }
    }

    c_dm_vtrid ((double*)ac, K, N, dd, sld, &icon);
    if (icon != 0) {
      printf(" icon of c_dm_vtrid =%d\n", icon);
      exit(0);
    }

    for (i=1; i<N; i++) {
      sud[i-1]=sld[i];
    }
    sud[N-1]=0.0;

    nf  = 1;
    nl  = N;
    ivec        = 0;
    eval_tol    = 1.0e-15;
    clus_tol    = 1.0e-10;
```

```
        c_dm_vtdevc( dd, sld, sud, N, nf, nl, ivec, &eval_tol, &clus_tol, nev, eval,
                     MAX_NEV, (double*)evec, K, (int*)mult, &icon);

        for (i=0; i<NE; i=i+N/20) {
          printf("eigen value in eval(%d) = %f\n",i+1,eval[i]);
        }

        return(0);
    }
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_VTRID の項目及び[30]を参照してください.

c_dm_v1dcft

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_dm_v1dcft(x, kx, y, ky, n1, n2, isn, &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, 混合基底 FFT により行います. 変換データ長 $n(= n_1 \times n_2)$ は, 2, 3, 5 及び 7 の巾の積で表すことのできる数です.

a) 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, $\{n\alpha_k\}$を求めます.

$$n\alpha_k = \sum_{j=0}^{n-1} x_j \omega_n^{-jk} \quad ,k = 0,1,...,n-1 \qquad (1)$$
$$,\omega_n = \exp(2\pi i / n)$$

b) 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, $\{x_j\}$を求めます.

$$x_j = \sum_{k=0}^{n-1} \alpha_k \omega_n^{jk} \quad , j = 0,1,...,n-1 \qquad (2)$$
$$,\omega_n = \exp(2\pi i / n)$$

2. 引数

呼出し形式:

```
ierr = c_dm_v1dcft((dcomplex*)x, kx, (dcomplex*)y, ky, n1, n2, isn, &icon);
```

引数の説明:

x	dcomplex x[n2][kx]	入力	複素数データ.
kx	int	入力	データ配列 x の 2 次元目の大きさ.
y	dcomplex y[n1][ky]	出力	変換された複素数データ.
ky	int	入力	データ配列 y の 2 次元目の大きさ.
n1	int	入力	変換データ長($n = n1 \times n2$)を 2 次元データとして見なしたときの 1 次元目の大きさ. n1 は 2, 3, 5 及び 7 の巾の積で表すことのできる数.
n2	int	入力	変換データ長($n = n1 \times n2$)を 2 次元データとして見なしたときの 2 次元目の大きさ. n1 は 2, 3, 5 及び 7 の巾の積で表すことのできる数.
isn	int	入力	変換か逆変換かを指定します. 変換　: 1 逆変換 : –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	配列の次元が 0 または 0 以下であった.	処理を打ち切る.
30002	先導する次元の大きさが実際の次元の大きさよりも小さかった.	
30008	変換数が, 2, 3, 5, 7 を基底としていない.	
30016	isn の値が適切でない.	

3. 使用上の注意

a) 入力配列及び出力配列について

$n = n_1 \times n_2$個の 1 次元データを $k = 0, \ldots, n-1$ と番号付けした時, 次のように書けます.

$$k = k_1 + k_2 \times n_1 \quad , k_1 = 0, \ldots, n_1 - 1 \quad , k_2 = 0, \ldots, n_2 - 1$$
$$i = i_1 + i_2 \times n_2 \quad , i_1 = 0, \ldots, n_2 - 1 \quad , i_2 = 0, \ldots, n_1 - 1$$

入力のデータは$[k_2][k_1]$, 出力のデータは$[i_2][i_1]$を添字とする 2 次元データと見なします. (図 c_dm_v1dcft-1 参照)

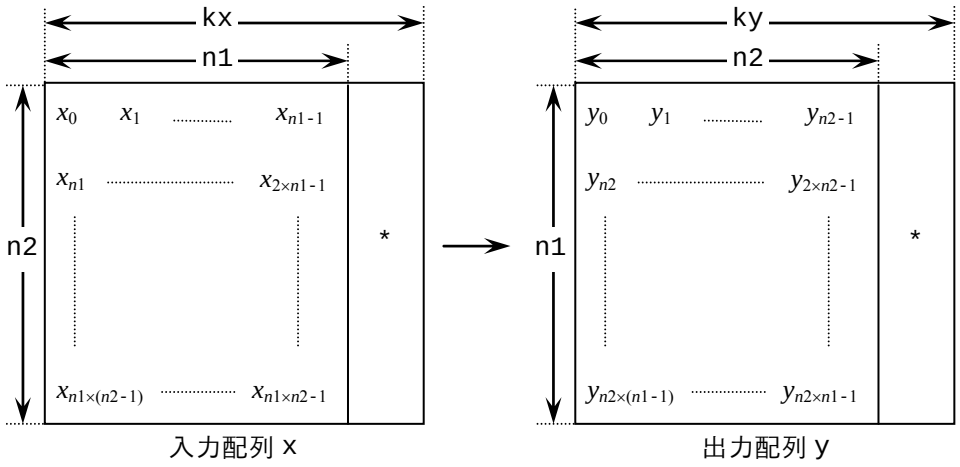


図 c_dm_v1dcft-1. 入力配列及び出力配列の格納形式

b) 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(3), (4)で定義されます.

$$\alpha_k = \frac{1}{n}\sum_{j=0}^{n-1} x_j \omega_n^{-jk}, k = 0, 1, ..., n-1 \tag{3}$$

$$x_j = \sum_{k=0}^{n-1} \alpha_k \omega_n^{jk}, j = 0, 1, ..., n-1 \tag{4}$$

ここで, $\omega_n = \exp(2\pi i/n)$

本関数では, (3), (4)の左辺に対応して$\{n\alpha_k\}$または$\{x_j\}$を求めます. したがって, 結果の正規化は必要に応じて行ってください.

4. 使用例

1 次元 FFT を計算します.

```
#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 N1  4000
#define N2  3000
#define KX  (N1+1)
#define KY  (N2+1)

MAIN__()
{
  int      isn, i, j, icon, ierr;
  double   error;
  dcomplex x[N2][KX], y[N1][KY];

  /* Set up the input data arrays */

#pragma omp parallel for shared(x) private(i,j)
  for(i=0; i<N2; i++) {
    for(j=0; j<N1; j++) {
      x[i][j].re = N1*i+j+1;
      x[i][j].im = 0.0;
    }
  }

  /* Do the forward transform */
  isn = 1;
  ierr = c_dm_v1dcft((dcomplex*)x, KX, (dcomplex*)y, KY, N1, N2, isn, &icon);

  if (icon != 0) {
    printf("ERROR: c_dm_v1dcft failed with icon = %d\n", icon);
    exit(1);
  }

  /* Do the reverse transform */
  isn = -1;
  ierr = c_dm_v1dcft((dcomplex*)y, KY, (dcomplex*)x, KX, N2, N1, isn, &icon);

  if (icon != 0) {
    printf("ERROR: c_dm_v1dcft failed with icon = %d\n", icon);
    exit(1);
  }

  /* Find the error after the forward and inverse transform. */
  error = 0.0;

  for(i=0; i<N2; i++) {
    for(j=0; j<N1; j++) {
      error = max(fabs(x[i][j].re)/N2/N1-(N1*i+j+1), error);
      error = max(fabs(x[i][j].im)/N2/N1, error);
    }
  }

  printf("error = %e\n", error);
  return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_V1DCFT の項目を参照してください.

c_dm_v1dcft2

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_dm_v1dcft2(x, n, y, isn, &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, 混合基底 FFT により行います. 変換データ長 *n* は, 2, 3, 5 及び 7 の巾の積で表すことのできる数です.

a) 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, $\{n\alpha_k\}$を求めます.

$$n\alpha_k = \sum_{j=0}^{n-1} x_j \omega_n^{-jk} \quad , k = 0,1,...,n-1 \qquad , \omega_n = \exp(2\pi i / n) \tag{1}$$

b) 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, $\{x_j\}$を求めます.

$$x_j = \sum_{k=0}^{n-1} \alpha_k \omega_n^{jk} \quad , j = 0,1,...,n-1 \qquad , \omega_n = \exp(2\pi i / n) \tag{2}$$

2. 引数

呼出し形式:

```
ierr = c_dm_v1dcft2(x, n, y, isn, &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*).
y	dcomplex y[n]	出力	変換された複素数データ.
isn	int	入力	変換か逆変換かを指定します. 変換　:　1 逆変換　: –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.
30008	変換数が, 2, 3, 5, 7 を基底としていない.	処理を打ち切る.
30016	isn の値が適切でない.	

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(3), (4)で定義されます.

$$\alpha_k = \frac{1}{n}\sum_{j=0}^{n-1} x_j \omega_n^{-jk},\ k = 0,\ 1,\ ...,\ n-1 \tag{3}$$

$$x_j = \sum_{k=0}^{n-1} \alpha_k \omega_n^{jk},\ j = 0,\ 1,\ ...,\ n-1 \tag{4}$$

ここで, $\omega_n = \exp(2\pi i/n)$

本関数では, (3), (4)の左辺に対応して$\{n\alpha_k\}$または$\{x_j\}$を求めます. したがって, 結果の正規化は必要に応じて行ってください.

4. 使用例

1 次元 FFT を計算します.

```
#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 N1  (1024)
#define N2  (N1)
#define N   (N1*N2)

MAIN__()
{
  dcomplex x[N], y[N], xx[N];
  double   tmp;
  int      isn, icon, i;

  for (i=0; i<N; i++) {
    xx[i].re = x[i].re = (double)(i);
    xx[i].im = x[i].im = 0.0;
  }

  isn = 1;
  c_dm_v1dcft2(x, N, y, isn, &icon);
  printf("icon = %d\n", icon);

  isn = -1;
  c_dm_v1dcft2(y, N, x, isn, &icon);
  printf("icon = %d\n", icon);

  tmp = 0.0;
  for (i=0; i<N; i++) {
    tmp = max((fabs(x[i].re/(double)N-xx[i].re))
              +(fabs(x[i].im/(double)N-xx[i].im)),tmp);
  }

  printf("error = %e\n", tmp);

  return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_V1DCFT2 の項目を参照してください.

c_dm_v1dmcft

1 次元多重離散型複素フーリエ変換 (2, 3, 5 及び 7 の混合基底)
`ierr = c_dm_v1dmcft(x, kx, n, m, isn, &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, 混合基底 FFT により多重に行います. 変換データ長 n は, 2, 3, 5 及び 7 の巾の積で表すことのできる数です.

a) 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, $\{n\alpha_k\}$を求めます.

$$n\alpha_k = \sum_{j=0}^{n-1} x_j \omega_n^{-jk} \quad , k = 0,1,\ldots,n-1 \qquad , \omega_n = \exp(2\pi i / n) \tag{1}$$

b) 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, $\{x_j\}$を求めます.

$$x_j = \sum_{k=0}^{n-1} \alpha_k \omega_n^{jk} \quad , j = 0,1,\ldots,n-1 \qquad , \omega_n = \exp(2\pi i / n) \tag{2}$$

2. 引数

呼出し形式:

```
ierr = c_dm_v1dmcft((dcomplex*)x, kx, n, m, isn, &icon);
```

引数の説明:

x	dcomplex x[m][kx]	入力	複素数データを x[i][j], i=0, … , m–1, j=0, … , n–1 に格納します.
		出力	変換された複素数データは, x[i][j], i=0, … , m–1, j=0, … , n–1 に格納されます.
kx	int	入力	データ配列 x の 2 次元目の大きさ.
n	int	入力	変換データ長. n は 2, 3, 5 及び 7 の巾の積で表すことのできる数.
m	int	入力	変換するデータの多重度.
isn	int	入力	変換か逆変換かを指定します. 変換　:　1 逆変換　: –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	配列の次元が 0 または 0 以下であった.	処理を打ち切る.
30002	先導する次元の大きさが実際の次元の大きさより小さかった.	

コード	意　味	処理内容
30008	変換数が, 2, 3, 5, 7 を基底としていない.	処理を打ち切る.
30016	isn の値が適切でない.	

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(3), (4)で定義されます.

$$\alpha_k = \frac{1}{n}\sum_{j=0}^{n-1} x_j \omega_n^{-jk},\ k = 0,\ 1,\ ...,\ n-1 \tag{3}$$

$$x_j = \sum_{k=0}^{n-1} \alpha_k \omega_n^{jk},\ j = 0,\ 1,\ ...,\ n-1 \tag{4}$$

ここで, $\omega_n = \exp(2\pi i/n)$

本関数では, (3), (4)の左辺に対応して$\{n\alpha_k\}$または$\{x_j\}$を求めます. したがって, 結果の正規化は必要に応じて行ってください.

4. 使用例

1 次元 FFT を計算します.

```
#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 N   2048
#define M   256
#define KX  (N+1)

MAIN__()
{
  int      isn, i, j, icon, ierr;
  double   error;
  dcomplex x[N][KX];

  /* Set up the input data arrays */
#pragma omp parallel for shared(x) private(i,j)
  for(i=0; i<M; i++) {
    for(j=0; j<N; j++) {
      x[i][j].re = N*i+j+1;
      x[i][j].im = 0.0;
    }
  }

  /* Do the forward transform */
  isn = 1;
  ierr = c_dm_v1dmcft((dcomplex*)x, KX, N, M, isn, &icon);

  if (icon != 0) {
    printf("ERROR: c_dm_v1dmcft failed with icon = %d\n", icon);
    exit(1);
  }

  /* Do the reverse transform */
  isn = -1;
  ierr = c_dm_v1dmcft((dcomplex*)x, KX, N, M, isn, &icon);

  if (icon != 0) {
    printf("ERROR: c_dm_v1dmcft failed with icon = %d\n", icon);
    exit(1);
  }

  /* Find the error after the forward and inverse transform. */
```

```
  error = 0.0;

  for(i=0; i<M; i++) {
    for(j=0; j<N; j++) {
      error = max(fabs(x[i][j].re)/N-(N*i+j+1), error);
      error = max(fabs(x[i][j].im)/N, error);
    }
  }

  printf("error = %e\n", error);
  return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_V1DMCFT の項目を参照してください.

c_dm_v1drcf

1 次元離散型実フーリエ変換 (2, 3, 5 及び 7 の混合基底)
`ierr = c_dm_v1drcf(x, kx, y, ky, n1, n2, isin, isn, &icon);`

1. 機能

1 次元実フーリエ変換の正変換または逆変換を, 混合基底 FFT により行います. 変換データ長 $n(=n_1 \times n_2)$は, 2, 3, 5 及び 7 の巾の積で表すことのできる数です.

a) 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, $\{n\alpha_k\}$を求めます.

$$n\alpha_k = \sum_{j=0}^{n-1} x_j \omega_n^{-jkr} \quad , k = 0,1,...,n-1$$
$$, \omega_n = \exp(2\pi i / n) \qquad (1)$$
$$, r = 1 \quad \text{または} \quad r = -1$$

b) 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, $\{x_j\}$を求めます.

$$x_j = \sum_{k=0}^{n-1} \alpha_k \omega_n^{jkr} \quad , j = 0,1,...,n-1$$
$$, \omega_n = \exp(2\pi i / n) \qquad (2)$$
$$, r = 1 \quad \text{または} \quad r = -1$$

2. 引数

呼出し形式:

```
ierr = c_dm_v1drcf((double*)x, kx, (dcomplex*)y, ky, nl, n2, isin, isn,
            &icon);
```

引数の説明:

x	double	入力	実データ.
	x[n2][kx]	／出力	データは, x[i][j], i=0, … , n2–1, j=0, … , n1–1 に格納します. 実数から複素数への変換のとき(isn=1)入力となり, 複素数から実数への変換のとき(isn=–1)出力となります. isn=1 のとき, 入力データは保存されません.
kx	int	入力	データ配列 x の 2 次元目の大きさ.
y	dcomplex	出力	変換された複素数データ.
	y[n1][ky]	／入力	データは, y[i][j], i=0, … , n1–1, j=0, … , n2/2 に格納します. 実数から複素数への変換のとき(isn=1)出力となり, 複素数から次数への変換のとき(isn=–1)入力となります. isn=–1 のとき, 入力データは保存されません.
ky	int	入力	データ配列 y の 2 次元目の大きさ. (ky ≥ n2/2+1)
n1	int	入力	変換データ長(n = n1×n2)を 2 次元データとして見なしたときの 1 次元目の大きさ. n1 は 2, 3, 5 及び 7 の巾の積で表すこと

			のできる数. n1×n2 は変換されるデータ列の長さです.
n2	int	入力	変換データ長(n = n1×n2)を 2 次元データとして見なしたときの 2 次元目の大きさ. n2 は 2, 3, 5 及び 7 の巾の積で表すことのできる数. n1×n2 は変換されるデータ列の長さです.
isin	int	入力	変換の方向を表します. 1 のとき : $r = 1$ –1 のとき : $r = -1$
isn	int	入力	変換か逆変換かを指定します. 変換 : 1 逆変換 : –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. • kx < n1 • ky < n2/2+1 • n1 < 1 • n2 < 1 • isin ≠ 1, –1 • isn ≠ 1, –1	処理を打ち切る.
30008	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) 入力配列及び出力配列について

$n = n_1 \times n_2$ 個の 1 次元データを $k = 0, \ldots, n-1$ と番号付けした時, 次のように書けます.

$$k = k_1 + k_2 \times n_1 \quad , k_1 = 0, \ldots, n_1 - 1 \quad , k_2 = 0, \ldots, n_2 - 1$$
$$i = i_1 + i_2 \times n_2 \quad , i_1 = 0, \ldots, n_2 - 1 \quad , i_2 = 0, \ldots, n_1 - 1$$

入力のデータは[k_2][k_1], 出力のデータは[i_2][i_1]を添字とする 2 次元データと見なします. (図 c_dm_v1drcf-1 参照)

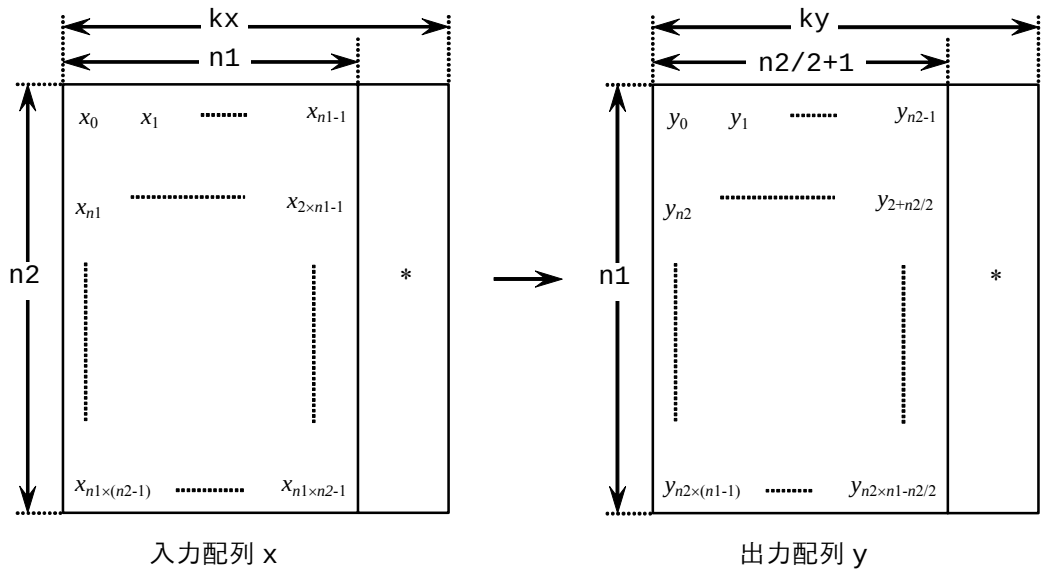


図 c_dm_v1drcf-1. 入力配列及び出力配列の格納形式

b) 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(3), (4)で定義されます.

$$\alpha_k = \frac{1}{n}\sum_{j=0}^{n-1} x_j \omega_n^{-jk},\ k = 0,\ 1,\ ...,\ n-1 \tag{3}$$

$$x_j = \sum_{k=0}^{n-1} \alpha_k \omega_n^{jk},\ j = 0,\ 1,\ ...,\ n-1 \tag{4}$$

ここで, $\omega_n = \exp(2\pi i/n)$

本関数では, (3), (4)の左辺に対応して$\{n\alpha_k\}$または$\{x_j\}$を求めます. したがって, 結果の正規化は必要に応じて行ってください.

c) 複素共役について

1 次元実フーリエ変換の結果は, 次の複素共役(¯で示します)の関係があります.

$$\alpha_k = \overline{\alpha_{n-k}},\ k = 1, ..., n-1$$

$n = n_1 \times n_2,\ i_1 = 0, 1, ..., n_2-1,\ i_2 = 0, 1, ..., n_1-1,\ k = i_1 + i_2 \times n_2$

とすると

$$n - k = n_2 - i_1 + (n_1 - 1 - i_2) \times n_2$$

であり, $i_1 = 1, ..., n_2/2$ までのデータ(0 を除いた前半部分)から残りのデータは求めることができます.

d) 性能について

n がほぼ等しい十分大きな数 n_1 および n_2 の積に分割できるとき, 最も性能が良くなります.

4. 使用例

1 次元実 FFT を計算します.

```
#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 N1  (1024)
#define N2  (N1)
#define KX  (N1+1)
#define KY  (N2/2+2)

MAIN__()
{
  dcomplex y[N1][KY];
  double   x[N2][KX], xx[N2][KX], tmp;
  int      isw, isin, icon, i, j;

  for (i=0; i<N2; i++) {
    for (j=0; j<N1; j++) {
      xx[i][j] = x[i][j] = N1*i+j+1;
    }
  }

  isin = 1;
  isw  = 1;
  c_dm_v1drcf((double*)x, KX, (dcomplex*)y, KY, N1, N2, isin, isw, &icon);
  printf("icon = %d\n", icon);

  isw  = -1;
  c_dm_v1drcf((double*)x, KX, (dcomplex*)y, KY, N1, N2, isin, isw, &icon);
  printf("icon = %d\n", icon);

  tmp = 0.0;
  for (i=0; i<N2; i++) {
    for (j=0; j<N1; j++) {
      tmp = max(fabs(x[i][j]/(double)N1/(double)N2-xx[i][j]),tmp);
    }
  }

  printf("error = %e\n", tmp);

  return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_V1DRCF の項目を参照してください.

c_dm_v1drcf2

1 次元離散型実フーリエ変換 (2, 3, 5 及び 7 の混合基底)
`ierr = c_dm_v1drcf2(x, n, y, isin, isn, &icon);`

1. 機能

1 次元実フーリエ変換の正変換または逆変換を, 混合基底 FFT により行います. 1 次元データの大きさ n は, 2, 3, 5 及び 7 の巾の積で表すことのできる数です.

a) フーリエ変換

実数列$\{x_j\}$を入力し, 複素数列$\{n\alpha_k\}$を求めます.

$$n\alpha_k = \sum_{j=0}^{n-1} x_j \omega_n^{-jkr} \quad ,k = 0,1,...,n-1$$
$$,\omega_n = \exp(2\pi i / n)$$
$$,r = 1 \quad \text{または} \quad r = -1$$

b) フーリエ逆変換

複素数列$\{\alpha_k\}$を入力し, 実数列$\{x_j\}$を求めます.

$$x_j = \sum_{k=0}^{n-1} \alpha_k \omega_n^{jkr} \quad , j = 0,1,...,n-1$$
$$,\omega_n = \exp(2\pi i / n)$$
$$,r = 1 \quad \text{または} \quad r = -1$$

2. 引数

呼出し形式:

```
ierr = c_dm_v1drcf2(x, n, y, isin, isn, &icon);
```

引数の説明:

x	double x[n]	入力	実数列を x[i], i=0, … , n–1 に格納します.
		／出力	変換(実数から複素数への変換)のとき入力となり, 逆変換(複素数から実数への変換)のとき出力となります.
n	int	入力	変換する実数データの数. n は 2, 3, 5, 7 の巾乗の積に分解できる偶数であること.
y	dcomplex	出力	約半分の複素数データが, y[i], i=0, … , n/2 に格納されます.
	y[n/2+1]	／入力	変換(実数から複素数への変換)のとき出力となり, 逆変換(複素数から実数への変換)のとき入力となります.
isin	int	入力	変換の方向を表します. 1 のとき : $r = 1$ –1 のとき : $r = -1$
isn	int	入力	変換か逆変換かを指定します. 変換 (実数から複素数への変換) : 1 逆変換 (複素数から実数への変換) : –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 が 2 の倍数でない. • n が 2, 3, 5 及び 7 の巾乗で表現できない. • isin ≠ 1, -1 • isn ≠ 1, -1	処理を打ち切る.

3. 使用上の注意

a) 複素共役について

1 次元実フーリエ変換の結果は, 次の複素共役(¯で示します)の関係があります.

$$\alpha_k = \overline{\alpha_{n-k}}\ ,\ k = 1, \ldots, n-1$$

b) 一般的なフーリエ変換の定義

1 次元実フーリエ変換および, 逆変換は一般的に(1), (2)で定義されます.

実数列$\{x_j\}$を入力して, 複素数列$\{\alpha_k\}$を求めます.

$$\alpha_k = \frac{1}{n}\sum_{j=0}^{n-1} x_j \omega_n^{-jk}, k = 0,\ 1,\ \ldots,\ n-1 \qquad (1)$$

複素数列$\{\alpha_k\}$を入力して, 実数列$\{x_j\}$を求めます.

$$x_j = \sum_{k=0}^{n-1} \alpha_k \omega_n^{jk}, j = 0,\ 1,\ \ldots,\ n-1 \qquad (2)$$

本関数では, (1), (2)の左辺に対応して$\{n\alpha_k\}$または$\{x_j\}$を求めます. したがって, 結果の正規化は必要に応じて行ってください.

4. 使用例

1 次元実 FFT を計算します.

```
#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 N1  (1024)
#define N2  (N1)
#define N   (N1*N2)

MAIN__()
{
  dcomplex y[N/2+1];
  double   x[N], xx[N], tmp;
  int      isin, isn, icon, i;

  for (i=0; i<N; i++) {
    xx[i] = x[i] = (double)(i+1);
  }

  isin = 1;
  isn  = 1;
```

```
  c_dm_v1drcf2(x, N, y, isin, isn, &icon);
  printf("icon = %d\n", icon);

  isn  = -1;
  c_dm_v1drcf2(x, N, y, isin, isn, &icon);
  printf("icon = %d\n", icon);

  tmp = 0.0;
  for (i=0; i<N; i++) {
    tmp = max(fabs(x[i]/(double)N-xx[i]),tmp);
  }

  printf("error = %e\n", tmp);

  return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_V1DRCF2 の項目を参照してください.

c_dm_v2dcft

2 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_dm_v2dcft(x, kx, n1, n2, isn, &icon);`

1. 機能

2 次元複素フーリエ変換の正変換または逆変換を, 混合基底 FFT により行います. 2 次元データ(n_1, n_2)の各次元の大きさは, 2, 3, 5 及び 7 の巾の積で表すことのできる数です.

a) 2 次元フーリエ変換

$\{x_{j1j2}\}$を入力し, (1)で定義する変換を行い, $\{n_1 n_2 \alpha_{k1k2}\}$を求めます.

$$
\begin{aligned}
n_1 n_2 \alpha_{k1k2} &= \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} x_{j1j2} \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \\
&, k_1 = 0,1,...,n_1 - 1 \\
&, k_2 = 0,1,...,n_2 - 1 \\
&, \omega_{n1} = \exp(2\pi i / n_1) \\
&, \omega_{n2} = \exp(2\pi i / n_2)
\end{aligned}
\tag{1}
$$

b) 2 次元フーリエ逆変換

$\{\alpha_{k1k2}\}$を入力し, (2)で定義する変換を行い, $\{x_{j1j2}\}$を求めます.

$$
\begin{aligned}
x_{j1j2} &= \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \alpha_{k1k2} \omega_{n1}^{j1k1} \omega_{n2}^{j2k2} \\
&, j_1 = 0,1,...,n_1 - 1 \\
&, j_2 = 0,1,...,n_2 - 1 \\
&, \omega_{n1} = \exp(2\pi i / n_1) \\
&, \omega_{n2} = \exp(2\pi i / n_2)
\end{aligned}
\tag{2}
$$

2. 引数

呼出し形式:

```
ierr = c_dm_v2dcft((dcomplex*)x, kx, n1, n2, isn, &icon);
```

引数の説明:

x	dcomplex x[n2][kx]	入力	複素数データを, x[i][j], i=0, … , n2–1, j=0, … , n1–1 に格納します.
		出力	変換された複素数データは, x[i][j], i=0, … , n2–1, j=0, … , n1–1 に格納されます.
kx	int	入力	データ配列 x の 2 次元目の大きさ.
n1	int	入力	変換する 2 次元データの 1 次元目の大きさ. n1 は 2, 3, 5 及び 7 の巾の積で表すことのできる数.
n2	int	入力	変換する 2 次元データの 2 次元目の大きさ. n1 は 2, 3, 5 及び 7 の巾の積で表すことのできる数.
isn	int	入力	変換か逆変換かを指定します. 変換 : 1 逆変換 : –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	配列の次元が 0 または 0 以下であった.	処理を打ち切る.
30002	先導する次元の大きさが実際の次元の大きさよりも小さかった.	
30008	変換数が, 2, 3, 5, 7 を基底としていない.	
30016	isn の値が適切でない.	

3. 使用上の注意

a) 一般的なフーリエ変換の定義

2 次元離散型複素フーリエ変換および, 逆変換は一般的に(3), (4)で定義されます.

$$\alpha_{k1k2} = \frac{1}{n_1 n_2} \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} x_{j1j2} \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \quad , k_1 = 0,\ 1,\ ...,\ n_1 - 1 \quad , k_2 = 0,\ 1,\ ...,\ n_2 - 1 \tag{3}$$

$$x_{j1j2} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \alpha_{k1k2} \omega_{n1}^{j1k1} \omega_{n2}^{j2k2} \quad , j_1 = 0,\ 1,\ ...,\ n_1 - 1 \quad , j_2 = 0,\ 1,\ ...,\ n_2 - 1 \tag{4}$$

ここで, $\omega_{n1} = \exp(2\pi i/n_1)$, $\omega_{n2} = \exp(2\pi i/n_2)$

本関数では, (3), (4)の左辺に対応して$\{n_1 n_2 \alpha_{k1k2}\}$または$\{x_{j1j2}\}$を求めます. したがって, 結果の正規化は必要に応じて行ってください.

4. 使用例

2 次元 FFT を計算します.

```
#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 N1  4000
#define N2  3000
#define KX  (N1+400)

MAIN__()
{
  int      isn, i, j, icon, ierr;
  double   error;
  dcomplex x[N2][KX];

  /* Set up the input data arrays */
#pragma omp parallel for shared(x) private(i,j)
  for(i=0; i<N2; i++) {
    for(j=0; j<N1; j++) {
      x[i][j].re = N1*i+j+1;
      x[i][j].im = 0.0;
    }
  }

  /* Do the forward transform */
  isn = 1;
  ierr = c_dm_v2dcft((dcomplex*)x, KX, N1, N2, isn, &icon);

  if (icon != 0) {
```

```
    printf("ERROR: c_dm_v2dcft failed with icon = %d\n", icon);
    exit(1);
  }

  /* Do the reverse transform */
  isn = -1;
  ierr = c_dm_v2dcft((dcomplex*)x, KX, N1, N2, isn, &icon);

  if (icon != 0) {
    printf("ERROR: c_dm_v2dcft failed with icon = %d\n", icon);
    exit(1);
  }

  /* Find the error after the forward and inverse transform. */
  error = 0.0;

  for(i=0; i<N2; i++) {
    for(j=0; j<N1; j++) {
      error = max(fabs(x[i][j].re)/(N2*N1)-(N1*i+j+1), error);
      error = max(fabs(x[i][j].im)/(N2*N1), error);
    }
  }

  printf("error = %e\n", error);
  return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_V2DCFT の項目を参照してください.

c_dm_v2drcf

2 次元離散型実フーリエ変換 (2, 3, 5 及び 7 の混合基底)
`ierr = c_dm_v2drcf(x, k, n1, n2, isin, isn, &icon);`

1. 機能

2 次元実フーリエ変換の正変換または逆変換を, 混合基底 FFT により行います. 2 次元データ(n_1, n_2)の各次元の大きさは, 2, 3, 5 及び 7 の巾の積で表すことのできる数です.

a) 2 次元フーリエ変換

$\{x_{j1j2}\}$を入力し, (1)で定義する変換を行い, $\{n_1 n_2 \alpha_{k1k2}\}$を求めます.

$$
\begin{aligned}
n_1 n_2 \alpha_{k1k2} &= \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} x_{j1j2} \omega_{n1}^{-j1k1r} \omega_{n2}^{-j2k2r} \\
&, k_1 = 0,1,...,n_1 - 1 \\
&, k_2 = 0,1,...,n_2 - 1 \\
&, \omega_{n1} = \exp(2\pi i / n_1) \\
&, \omega_{n2} = \exp(2\pi i / n_2) \\
&, r = 1 \quad \text{または} \quad r = -1
\end{aligned}
\tag{1}
$$

b) 2 次元フーリエ逆変換

$\{\alpha_{k1k2}\}$を入力し, (2)で定義する変換を行い, $\{x_{j1j2}\}$を求めます.

$$
\begin{aligned}
x_{j1j2} &= \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \alpha_{k1k2} \omega_{n1}^{j1k1r} \omega_{n2}^{j2k2r} \\
&, j_1 = 0,1,...,n_1 - 1 \\
&, j_2 = 0,1,...,n_2 - 1 \\
&, \omega_{n1} = \exp(2\pi i / n_1) \\
&, \omega_{n2} = \exp(2\pi i / n_2) \\
&, r = 1 \quad \text{または} \quad r = -1
\end{aligned}
\tag{2}
$$

2. 引数

呼出し形式:

```
ierr = c_dm_v2drcf((double*)x, k, nl, n2, isin, isn, &icon);
```

引数の説明:

x	double x[n2][k]	入力 ／出力	2 次元実データ. データを, x[i][j], i=0, … , n2–1, j=0, … , n1–1 に格納します. 実数から複素数への変換(isn=1)のとき入力に, 複素数から実数への変換(isn=–1)のとき出力になります.

		出力	変換された複素データの実部および虚部.
		／入力	x を x[n2][k/2][2]なる配列とみなして, 実部は x[i][j][0], i=0, … , n2–1, j=0, … , n1/2 に, 虚部は x[i][j][1], i=0, … , n2–1, j=0, … , n1/2 に格納されます. 実数から複素数への変換(isn=1)のとき出力に, 複素数から実数への変換(isn=–1)のとき入力になります. フーリエ変換された複素数データには共役関係があり, 約半分が格納されます.
k	int	入力	データ配列 x の整合寸法(≥ 2×(n1/2+1)). 偶数.
n1	int	入力	変換する 2 次元データの 1 次元目の大きさ. n1 は 2, 3, 5 及び 7 の巾の積で表すことのできる数.
n2	int	入力	変換する 2 次元データの 2 次元目の大きさ. n2 は 2, 3, 5 及び 7 の巾の積で表すことのできる数.
isin	int	入力	変換の方向を表します. 1 のとき : $r=1$ –1 のとき : $r=-1$
isn	int	入力	変換か逆変換かを指定します. 変換 : 1 逆変換 : –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. • k < 2×(n1/2+1) • k が偶数でない. • n1 < 1 • n2 < 1 • isin ≠ 1, –1 • isn ≠ 1, –1	処理を打ち切る.
30008	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

2 次元離散型複素フーリエ変換および, 逆変換は一般的に(3), (4)で定義されます.

$$\alpha_{k1k2} = \frac{1}{n_1 n_2} \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} x_{j1\,j2} \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \quad (3)$$
$$, k_1 = 0,\ 1,\ ...,\ n_1 - 1$$
$$, k_2 = 0,\ 1,\ ...,\ n_2 - 1$$

$$x_{j1\,j2} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \alpha_{k1k2} \omega_{n1}^{j1k1} \omega_{n2}^{j2k2} \quad (4)$$
$$, j_1 = 0,\ 1,\ ...,\ n_1 - 1$$
$$, j_2 = 0,\ 1,\ ...,\ n_2 - 1$$

ここで, $\omega_{n1} = \exp(2\pi i/n_1)$, $\omega_{n2} = \exp(2\pi i/n_2)$

本関数では, (3), (4)の左辺に対応して$\{n_1 n_2 \alpha_{k1k2}\}$または$\{x_{j1j2}\}$を求めます. したがって, 結果の正規化は必要に応じて行ってください.

b) 複素共役について

2 次元の実データのフーリエ変換結果には次の複素共役(¯で示します)の関係があります.

$$\alpha_{k1k2} = \overline{\alpha_{n1-k1\,n2-k2}}$$

$k_1 = 0, \ldots, n_1/2, k_2 = 0, \ldots, n_2 - 1$ のデータから残りのデータを計算することができます.

4. 使用例

2 次元実 FFT を計算します.

```
#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 N1  (2048)
#define N2  (N1)
#define K   ((N1/2+1)*2)

MAIN__()
{
  double x[N2][K], xx[N2][K], tmp;
  int    isin, isn, icon, i, j;

  for (i=0; i<N2; i++) {
    for (j=0; j<N1; j++) {
      xx[i][j] = x[i][j] = (double)(N2*i+j+1);
    }
  }

  isin = 1;
  isn  = 1;
  c_dm_v2drcf((double*)x, K, N1, N2, isin, isn, &icon);
  printf("icon = %d\n", icon);

  isn  = -1;
  c_dm_v2drcf((double*)x, K, N1, N2, isin, isn, &icon);
  printf("icon = %d\n", icon);

  tmp = 0.0;
  for (i=0; i<N2; i++) {
    for (j=0; j<N1; j++) {
      tmp = max(fabs(x[i][j]/(double)N1/(double)N2-xx[i][j]),tmp);
    }
  }

  printf("error = %e\n", tmp);

  return(0);
```

```
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_V2DRCF の項目を参照してください.

c_dm_v3dcft

3 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_dm_v3dcft(x, kx, n1, n2, n3, isn, &icon);`

1. 機能

3 次元複素フーリエ変換の正変換または逆変換を, 混合基底 FFT により行います. 3 次元データ(n_1, n_2, n_3)の各次元の大きさは, 2, 3, 5 及び 7 の巾の積で表すことのできる数です.

a) 3 次元フーリエ変換

$\{x_{j1j2j3}\}$を入力し, (1)で定義する変換を行い, $\{n_1n_2n_3\alpha_{k1k2k3}\}$を求めます.

$$
\begin{aligned}
n_1n_2n_3\alpha_{k1k2k3} = \sum_{j1=0}^{n1-1}\sum_{j2=0}^{n2-1}\sum_{j3=0}^{n3-1} x_{j1j2j3}\omega_{n1}^{-j1k1}\omega_{n2}^{-j2k2}\omega_{n3}^{-j3k3} \\
,k_1 = 0,1,...,n_1-1 \\
,k_2 = 0,1,...,n_2-1 \\
,k_3 = 0,1,...,n_3-1 \\
,\omega_{n1} = \exp(2\pi i/n_1) \\
,\omega_{n2} = \exp(2\pi i/n_2) \\
,\omega_{n3} = \exp(2\pi i/n_3)
\end{aligned}
\tag{1}
$$

b) 3 次元フーリエ逆変換

$\{\alpha_{k1k2k3}\}$を入力し, (2)で定義する変換を行い, $\{x_{j1j2j3}\}$を求めます.

$$
\begin{aligned}
x_{j1}x_{j2}x_{j3} = \sum_{k1=0}^{n1-1}\sum_{k2=0}^{n2-1}\sum_{k3=0}^{n3-1} \alpha_{k1k2k3}\omega_{n1}^{j1k1}\omega_{n2}^{j2k2}\omega_{n3}^{j3k3} \\
,j_1 = 0,1,...,n_1-1 \\
,j_2 = 0,1,...,n_2-1 \\
,j_3 = 0,1,...,n_3-1 \\
,\omega_{n1} = \exp(2\pi i/n_1) \\
,\omega_{n2} = \exp(2\pi i/n_2) \\
,\omega_{n3} = \exp(2\pi i/n_3)
\end{aligned}
\tag{2}
$$

2. 引数

呼出し形式:

```
ierr = c_dm_v3dcft((dcomplex*)x, kx, n1, n2, n3, isn, &icon);
```

引数の説明:

x	dcomplex x[n3][n2][kx]	入力	複素数データを, x[i][j][k], i=0, … , n3–1, j=0, … , n2–1, k=0, … , n1–1 に格納します.
		出力	変換された複素数データは, x[i][j][k], i=0, … , n3–1, j=0, … , n2–1, k=0, … , n1–1 に格納されます.
kx	int	入力	データ配列 x の 3 次元目の大きさ.
n1	int	入力	変換する 3 次元データの 1 次元目の大きさ. n1 は 2, 3, 5 及び 7 の巾の積で表すことのできる数.
n2	int	入力	変換する 3 次元データの 2 次元目の大きさ. n2 は 2, 3, 5 及び 7

			の巾の積で表すことのできる数.
n3	int	入力	変換する 3 次元データの 3 次元目の大きさ. n3 は 2, 3, 5 及び 7 の巾の積で表すことのできる数.
isn	int	入力	変換か逆変換かを指定します. 変換　：1 逆変換：–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	配列の次元が 0 または 0 以下であった.	処理を打ち切る.
30002	先導する次元の大きさが実際の次元の大きさよりも小さかった.	
30008	変換数が, 2, 3, 5, 7 を基底としていない.	
30016	isn の値が適切でない.	

3. 使用上の注意

a) 一般的なフーリエ変換の定義

3 次元離散型複素フーリエ変換および, 逆変換は一般的に(3), (4)で定義されます.

$$\alpha_{k1k2k3} = \frac{1}{n_1 n_2 n_3} \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \sum_{j3=0}^{n3-1} x_{j1j2j3} \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \omega_{n3}^{-j3k3}$$
$$, k_1 = 0, 1, ..., n_1 - 1$$
$$, k_2 = 0, 1, ..., n_2 - 1 \quad (3)$$
$$, k_3 = 0, 1, ..., n_3 - 1$$

$$x_{j1j2j3} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \sum_{k3=0}^{n3-1} \alpha_{k1k2k3} \omega_{n1}^{j1k1} \omega_{n2}^{j2k2} \omega_{n3}^{j3k3}$$
$$, j_1 = 0, 1, ..., n_1 - 1$$
$$, j_2 = 0, 1, ..., n_2 - 1 \quad (4)$$
$$, j_3 = 0, 1, ..., n_3 - 1$$

ここで, $\omega_{n1} = \exp(2\pi i/n_1)$, $\omega_{n2} = \exp(2\pi i/n_2)$, $\omega_{n3} = \exp(2\pi i/n_3)$

本関数では, (3), (4)の左辺に対応して$\{n_1 n_2 n_3 \alpha_{k1k2k3}\}$または$\{x_{j1j2j3}\}$を求めます. したがって, 結果の正規化は必要に応じて行ってください.

4. 使用例

3 次元 FFT を計算します.

```
#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 N1  400
#define N2  100
#define N3  200
#define KX  (N1+40)

MAIN__()
{
  int      isn, i, j, k, icon, ierr;
```

```
  double   error;
  dcomplex x[N3][N2][KX];

  /* Set up the input data arrays */
#pragma omp parallel for shared(x) private(i,j)
  for(k=0; k<N3; k++) {
    for(i=0; i<N2; i++) {
      for(j=0; j<N1; j++) {
        x[k][i][j].re = N1*i+j+1;
        x[k][i][j].im = 0.0;
      }
    }
  }

  /* Do the forward transform */
  isn = 1;
  ierr = c_dm_v3dcft((dcomplex*)x, KX, N1, N2, N3, isn, &icon);

  if (icon != 0) {
    printf("ERROR: c_dm_v3dcft failed with icon = %d\n", icon);
    exit(1);
  }

  /* Do the reverse transform */
  isn = -1;
  ierr = c_dm_v3dcft((dcomplex*)x, KX, N1, N2, N3, isn, &icon);

  if (icon != 0) {
    printf("ERROR: c_dm_v3dcft failed with icon = %d\n", icon);
    exit(1);
  }

  /* Find the error after the forward and inverse transform. */
  error = 0.0;

  for(k=0; k<N3; k++) {
    for(i=0; i<N2; i++) {
      for(j=0; j<N1; j++) {
        error = max(fabs(x[k][i][j].re)/(N3*N2*N1)-(N1*i+j+1), error);
        error = max(fabs(x[k][i][j].im)/(N3*N2*N1), error);
      }
    }
  }

  printf("error = %e\n", error);
  return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_V3DCFT の項目を参照してください.

c_dm_v3dcft2

3 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_dm_v3dcft2(x, k1, k2, n1, n2, n3, isn, &icon);`

1. 機能

3 次元複素フーリエ変換の正変換または逆変換を行います. 3 次元データ(n_1, n_2, n_3)の各次元の大きさは, 2, 3, 5 及び 7 の巾の積で表すことのできる数です.

a) 3 次元フーリエ変換

$\{x_{j1j2j3}\}$を入力し, (1)で定義する変換を行い, $\{n_1n_2n_3\alpha_{k1k2k3}\}$を求めます.

$$
\begin{aligned}
n_1n_2n_3\alpha_{k1k2k3} = \sum_{j1=0}^{n1-1}\sum_{j2=0}^{n2-1}\sum_{j3=0}^{n3-1} x_{j1j2j3}\omega_{n1}^{-j1k1}\omega_{n2}^{-j2k2}\omega_{n3}^{-j3k3} \\
,k_1 = 0,1,...,n_1-1 \\
,k_2 = 0,1,...,n_2-1 \\
,k_3 = 0,1,...,n_3-1 \\
,\omega_{n1} = \exp(2\pi i/n_1) \\
,\omega_{n2} = \exp(2\pi i/n_2) \\
,\omega_{n3} = \exp(2\pi i/n_3)
\end{aligned}
\tag{1}
$$

b) 3 次元フーリエ逆変換

$\{\alpha_{k1k2k3}\}$を入力し, (2)で定義する変換を行い, $\{x_{j1j2j3}\}$を求めます.

$$
\begin{aligned}
x_{j1}x_{j2}x_{j3} = \sum_{k1=0}^{n1-1}\sum_{k2=0}^{n2-1}\sum_{k3=0}^{n3-1} \alpha_{k1k2k3}\omega_{n1}^{j1k1}\omega_{n2}^{j2k2}\omega_{n3}^{j3k3} \\
,j_1 = 0,1,...,n_1-1 \\
,j_2 = 0,1,...,n_2-1 \\
,j_3 = 0,1,...,n_3-1 \\
,\omega_{n1} = \exp(2\pi i/n_1) \\
,\omega_{n2} = \exp(2\pi i/n_2) \\
,\omega_{n3} = \exp(2\pi i/n_3)
\end{aligned}
\tag{2}
$$

2. 引数

呼出し形式:

```
ierr = c_dm_v3dcft2((dcomplex*)x, k1, k2, n1, n2, n3, isn, &icon);
```

引数の説明:

x	dcomplex x[n3][k2][k1]	入力	複素数データを, x[i][j][k], i=0, … , n3–1, j=0, … , n2–1, k=0, … , n1–1 に格納します.
		出力	変換された複素数データは, x[i][j][k], i=0, … , n3–1, j=0, … , n2–1, k=0, … , n1–1 に格納されます.
k1	int	入力	データ配列 x の 3 次元目の大きさ. (≥n1)
k2	int	入力	データ配列 x の 2 次元目の大きさ. (≥n2)
n1	int	入力	変換する 3 次元データの 1 次元目の大きさ. n1 は 2, 3, 5 及び 7 の巾の積で表すことのできる数.

n2	int	入力	変換する 3 次元データの 2 次元目の大きさ. n2 は 2, 3, 5 及び 7 の巾の積で表すことのできる数.
n3	int	入力	変換する 3 次元データの 3 次元目の大きさ. n3 は 2, 3, 5 及び 7 の巾の積で表すことのできる数.
isn	int	入力	変換か逆変換かを指定します. 変換　：1 逆変換　：–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, n_2, n_3 が 0 または 0 以下であった. • k1 < n1 • k2 < n2 • isn の値が適切でない.	処理を打ち切る.
30008	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

3 次元離散型複素フーリエ変換および, 逆変換は一般的に(3), (4)で定義されます.

$$\alpha_{k1k2k3} = \frac{1}{n_1 n_2 n_3} \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \sum_{j3=0}^{n3-1} x_{j1j2j3} \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \omega_{n3}^{-j3k3} \quad , k_1 = 0, 1, ..., n_1 - 1 \quad , k_2 = 0, 1, ..., n_2 - 1 \quad , k_3 = 0, 1, ..., n_3 - 1 \tag{3}$$

$$x_{j1j2j3} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \sum_{k3=0}^{n3-1} \alpha_{k1k2k3} \omega_{n1}^{j1k1} \omega_{n2}^{j2k2} \omega_{n3}^{j3k3} \quad , j_1 = 0, 1, ..., n_1 - 1 \quad , j_2 = 0, 1, ..., n_2 - 1 \quad , j_3 = 0, 1, ..., n_3 - 1 \tag{4}$$

ここで, $\omega_{n1} = \exp(2\pi i/n_1)$, $\omega_{n2} = \exp(2\pi i/n_2)$, $\omega_{n3} = \exp(2\pi i/n_3)$

本関数では, (3), (4)の左辺に対応して$\{n_1 n_2 n_3 \alpha_{k1k2k3}\}$または$\{x_{j1j2j3}\}$を求めます. したがって, 結果の正規化は必要に応じて行ってください.

4. 使用例

3 次元 FFT を計算します.

```
#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 N1 128
#define N2 128
#define N3 128
#define K1 (N1+1)
```

```
#define K2 N2

int MAIN__()
{
  dcomplex x[N3][K2][K1];
  double   error;
  int      i, j, k, isn, icon;

#pragma omp parallel for shared(x) private(i,j)
  for (k=0; k<N3; k++) {
    for (j=0; j<N2; j++) {
      for (i=0; i<N1; i++) {
        x[k][j][i].re = N1*j+i+1;
        x[k][j][i].im = 0.0;
      }
    }
  }

  isn = 1;
  c_dm_v3dcft2((dcomplex *)x, K1, K2, N1, N2, N3, isn, &icon);
  if (icon != 0) printf("error occurred : %d \n",icon);

  isn = -1;
  c_dm_v3dcft2((dcomplex *)x, K1, K2, N1, N2, N3, isn, &icon);
  if (icon != 0) printf("error occurred : %d \n",icon);

  /* find the error after the forward and inverse transform. */
  error = 0.0;
  for(k=0; k<N3; k++) {
    for(j=0; j<N2; j++) {
      for(i=0; i<N1; i++) {
        error = max(fabs(x[k][j][i].re)/(N3*N2*N1)-(N1*j+i+1), error);
        error = max(fabs(x[k][j][i].im)/(N3*N2*N1), error);
      }
    }
  }

  printf("error = %e\n", error);
  return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_V3DCFT2 の項目を参照してください.

c_dm_v3dcpf

3 次元素因子離散型複素フーリエ変換
`ierr = c_dm_v3dcpf(x, k1, k2, n1, n2, n3, isn, &icon);`

1. 機能

3 次元複素フーリエ変換の正変換または逆変換を行います. 3 次元データ(n_1, n_2, n_3)の各次元の大きさは, 次の条件を満たす数です.

- 次の数の中で互いに素な因子の積で表せること.
  因子 p($p \in \{2, 3, 4, 5, 7, 8, 9, 16, 25\}$)

a) 3 次元フーリエ変換

$\{x_{j1j2j3}\}$を入力し, (1)で定義する変換を行い, $\{n_1n_2n_3\alpha_{k1k2k3}\}$を求めます.

$$
\begin{aligned}
n_1n_2n_3\alpha_{k1k2k3} = &\sum_{j1=0}^{n1-1}\sum_{j2=0}^{n2-1}\sum_{j3=0}^{n3-1} x_{j1j2j3}\omega_{n1}^{-j1k1}\omega_{n2}^{-j2k2}\omega_{n3}^{-j3k3} \\
&, k_1 = 0,1,...,n_1-1 \\
&, k_2 = 0,1,...,n_2-1 \\
&, k_3 = 0,1,...,n_3-1 \\
&, \omega_{n1} = \exp(2\pi i/n_1) \\
&, \omega_{n2} = \exp(2\pi i/n_2) \\
&, \omega_{n3} = \exp(2\pi i/n_3)
\end{aligned}
\tag{1}
$$

b) 3 次元フーリエ逆変換

$\{\alpha_{k1k2k3}\}$を入力し, (2)で定義する変換を行い, $\{x_{j1j2j3}\}$を求めます.

$$
\begin{aligned}
x_{j1}x_{j2}x_{j3} = &\sum_{k1=0}^{n1-1}\sum_{k2=0}^{n2-1}\sum_{k3=0}^{n3-1} \alpha_{k1k2k3}\omega_{n1}^{j1k1}\omega_{n2}^{j2k2}\omega_{n3}^{j3k3} \\
&, j_1 = 0,1,...,n_1-1 \\
&, j_2 = 0,1,...,n_2-1 \\
&, j_3 = 0,1,...,n_3-1 \\
&, \omega_{n1} = \exp(2\pi i/n_1) \\
&, \omega_{n2} = \exp(2\pi i/n_2) \\
&, \omega_{n3} = \exp(2\pi i/n_3)
\end{aligned}
\tag{2}
$$

2. 引数

呼出し形式:

```
ierr = c_dm_v3dcpf((dcomplex*)x, k1, k2, n1, n2, n3, isn, &icon);
```

引数の説明:

x	dcomplex x[n3][k2][k1]	入力	複素数データを, x[i][j][k], i=0, … , n3–1, j=0, … , n2–1, k=0, … , n1–1 に格納します.
		出力	変換された複素数データは, x[i][j][k], i=0, … , n3–1, j=0, … , n2–1, k=0, … , n1–1 に格納されます.

k1	int	入力	データ配列 x の 3 次元目の大きさ. (≥n1)
k2	int	入力	データ配列 x の 2 次元目の大きさ. (≥n2)
n1	int	入力	変換する 3 次元データの 1 次元目の大きさ.
n2	int	入力	変換する 3 次元データの 2 次元目の大きさ.
n3	int	入力	変換する 3 次元データの 3 次元目の大きさ.
isn	int	入力	変換か逆変換かを指定します. 変換　：1 逆変換　：–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	n_1, n_2, n_3 が 2, 3, 4, 5, 7, 8, 9, 16, 25 の中の互いに素な因子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. • n_1, n_2, n_3 が 0 または 0 以下であった. • k1 < n1 • k2 < n2 • isn の値が適切でない.	

3. 使用上の注意

a) 一般的なフーリエ変換の定義

3 次元離散型複素フーリエ変換および, 逆変換は一般的に(3), (4)で定義されます.

$$\alpha_{k1k2k3} = \frac{1}{n_1 n_2 n_3} \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \sum_{j3=0}^{n3-1} x_{j1j2j3} \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \omega_{n3}^{-j3k3} \quad (3)$$
$$, k_1 = 0, 1, ..., n_1 - 1$$
$$, k_2 = 0, 1, ..., n_2 - 1$$
$$, k_3 = 0, 1, ..., n_3 - 1$$

$$x_{j1j2j3} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \sum_{k3=0}^{n3-1} \alpha_{k1k2k3} \omega_{n1}^{j1k1} \omega_{n2}^{j2k2} \omega_{n3}^{j3k3} \quad (4)$$
$$, j_1 = 0, 1, ..., n_1 - 1$$
$$, j_2 = 0, 1, ..., n_2 - 1$$
$$, j_3 = 0, 1, ..., n_3 - 1$$

ここで, $\omega_{n1} = \exp(2\pi i/n_1)$, $\omega_{n2} = \exp(2\pi i/n_2)$, $\omega_{n3} = \exp(2\pi i/n_3)$

本関数では, (3), (4)の左辺に対応して$\{n_1 n_2 n_3 \alpha_{k1k2k3}\}$または$\{x_{j1j2j3}\}$を求めます. したがって, 結果の正規化は必要に応じて行ってください.

4. 使用例

3 次元 FFT を計算します.

```
#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 N1 40
```

```
#define N2 240
#define N3 90
#define K1 N1
#define K2 N2

int MAIN__()
{
  dcomplex x[N3][K2][K1];
  double   error;
  int      i, j, k, isn, icon;

#pragma omp parallel for shared(x) private(i,j)
  for (k=0; k<N3; k++) {
    for (j=0; j<N2; j++) {
      for (i=0; i<N1; i++) {
        x[k][j][i].re = N1*j+i+1;
        x[k][j][i].im = 0.0;
      }
    }
  }

  isn = 1;
  c_dm_v3dcpf((dcomplex *)x, K1, K2, N1, N2, N3, isn, &icon);
  if (icon != 0) printf("error occurred : %d \n",icon);

  isn = -1;
  c_dm_v3dcpf((dcomplex *)x, K1, K2, N1, N2, N3, isn, &icon);
  if (icon != 0) printf("error occurred : %d \n",icon);

  /* find the error after the forward and inverse transform. */
  error = 0.0;
  for(k=0; k<N3; k++) {
    for(j=0; j<N2; j++) {
      for(i=0; i<N1; i++) {
        error = max(fabs(x[k][j][i].re)/(N3*N2*N1)-(N1*j+i+1), error);
        error = max(fabs(x[k][j][i].im)/(N3*N2*N1), error);
      }
    }
  }

  printf("error = %e\n", error);
  return(0);
}
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_V3DCPF の項目を参照してください.

c_dm_v3drcf

3 次元離散型実フーリエ変換 (2, 3, 5 及び 7 の混合基底)
`ierr = c_dm_v3drcf(x, k, n1, n2, n3, isin, isn, &icon);`

1. 機能

3 次元実フーリエ変換の正変換または逆変換を, 混合基底 FFT により行います. 3 次元データ(n_1, n_2, n_3)の各次元の大きさは, 2, 3, 5 及び 7 の巾の積で表すことのできる数です.

a) 3 次元フーリエ変換

$\{x_{j1j2j3}\}$を入力し, (1)で定義する変換を行い, $\{n_1n_2n_3\alpha_{k1k2k3}\}$を求めます.

$$
\begin{aligned}
n_1n_2n_3\alpha_{k1k2k3} &= \sum_{j1=0}^{n1-1}\sum_{j2=0}^{n2-1}\sum_{j3=0}^{n3-1} x_{j1j2j3}\omega_{n1}^{-j1k1r}\omega_{n2}^{-j2k2r}\omega_{n3}^{-j3k3r} \\
&, k_1 = 0,1,\ldots,n_1-1 \\
&, k_2 = 0,1,\ldots,n_2-1 \\
&, k_3 = 0,1,\ldots,n_3-1 \\
&, \omega_{n1} = \exp(2\pi i/n_1) \\
&, \omega_{n2} = \exp(2\pi i/n_2) \\
&, \omega_{n3} = \exp(2\pi i/n_3) \\
&, r = 1 \quad \text{または} \quad r = -1
\end{aligned}
\tag{1}
$$

b) 3 次元フーリエ逆変換

$\{\alpha_{k1k2k3}\}$を入力し, (2)で定義する変換を行い, $\{x_{j1j2j3}\}$を求めます.

$$
\begin{aligned}
x_{j1j2j3} &= \sum_{k1=0}^{n1-1}\sum_{k2=0}^{n2-1}\sum_{k3=0}^{n3-1} \alpha_{k1k2k3}\omega_{n1}^{j1k1r}\omega_{n2}^{j2k2r}\omega_{n3}^{j3k3r} \\
&, j_1 = 0,1,\ldots,n_1-1 \\
&, j_2 = 0,1,\ldots,n_2-1 \\
&, j_3 = 0,1,\ldots,n_3-1 \\
&, \omega_{n1} = \exp(2\pi i/n_1) \\
&, \omega_{n2} = \exp(2\pi i/n_2) \\
&, \omega_{n3} = \exp(2\pi i/n_3) \\
&, r = 1 \quad \text{または} \quad r = -1
\end{aligned}
\tag{2}
$$

2. 引数

呼出し形式:

```
ierr = c_dm_v3drcf((double*)x, k, nl, n2, n3, isin, isn, &icon);
```

引数の説明:

x	double x[n3][n2][k]	入力 ／出力	3 次元実データ. データを, x[i][j][k], i=0, … , n3−1, j=0, … , n2−1, k=0, … , n1−1 に格納します. 実数から複素数への変換(isn=1)のとき入力, 複素数から実数への変換(isn=−1)のとき出力となります.

		出力 ／入力	変換された複素データの実部および虚部. x を x[n3][n2][k/2][2]なる配列とみなして, 実部は x[i][j][k][0], i=0, … , n3–1, j=0, … , n2–1, k=0, … , n1/2 に, 虚部は x[i][j][k][1], i=0, … , n3–1, j=0, … , n2–1, k=0, … , n1/2 に格納されます. 実数から複素数への変換(isn=1)のとき出力に, 複素数から実数への変換(isn=–1)のとき入力になります. フーリエ変換された複素数データには共役関係があり, 約半分が格納されます.
k	int	入力	データ配列 x の整合寸法(≥ 2×(n1/2+1)). 偶数.
n1	int	入力	変換する 3 次元データの 1 次元目の大きさ. n1 は 2, 3, 5 及び 7 の巾の積で表すことのできる数.
n2	int	入力	変換する 3 次元データの 2 次元目の大きさ. n2 は 2, 3, 5 及び 7 の巾の積で表すことのできる数.
n3	int	入力	変換する 3 次元データの 2 次元目の大きさ. n3 は 2, 3, 5 及び 7 の巾の積で表すことのできる数.
isin	int	入力	変換の方向を表します. 1 のとき　：$r=1$ –1 のとき　：$r=-1$
isn	int	入力	変換か逆変換かを指定します. 変換　：1 逆変換　：–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. • k < 2×(n1/2+1) • k が偶数でない. • n1 < 1 • n2 < 1 • n3 < 1 • isin ≠ 1, –1 • isn ≠ 1, –1	処理を打ち切る.
30008	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

3 次元離散型複素フーリエ変換および, 逆変換は一般的に(3), (4)で定義されます.

$$\alpha_{k1k2k3} = \frac{1}{n_1 n_2 n_3} \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \sum_{j3=0}^{n3-1} x_{j1j2j3} \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \omega_{n3}^{-j3k3}$$
$$, k_1 = 0, 1, ..., n_1 - 1 \quad (3)$$
$$, k_2 = 0, 1, ..., n_2 - 1$$
$$, k_3 = 0, 1, ..., n_3 - 1$$

$$x_{j1j2j3} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \sum_{k3=0}^{n3-1} \alpha_{k1k2k3} \omega_{n1}^{j1k1} \omega_{n2}^{j2k2} \omega_{n3}^{j3k3}$$
$$, j_1 = 0, 1, ..., n_1 - 1 \quad (4)$$
$$, j_2 = 0, 1, ..., n_2 - 1$$
$$, j_3 = 0, 1, ..., n_3 - 1$$

ここで, $\omega_{n1} = \exp(2\pi i/n_1)$, $\omega_{n2} = \exp(2\pi i/n_2)$, $\omega_{n3} = \exp(2\pi i/n_3)$

本関数では, (3), (4)の左辺に対応して$\{n_1 n_2 n_3 \alpha_{k1k2k3}\}$または$\{x_{j1j2j3}\}$を求めます. したがって, 結果の正規化は必要に応じて行ってください.

b) 複素共役について

3 次元の実データのフーリエ変換結果には次の複素共役(¯で示します)の関係があります.

$$\alpha_{k1k2k3} = \overline{\alpha_{n1-k1\, n2-k2\, n3-k3}}$$

$k_1 = 0, \ldots, n_1/2, k_2 = 0, \ldots, n_2-1, k_3 = 0, \ldots, n_3-1$ のデータから残りのデータを計算することができます.

4. 使用例

3 次元実 FFT を計算します.

```
#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 N1  (128)
#define N2  (N1)
#define N3  (N1)
#define K   ((N1/2+1)*2)

MAIN__()
{
  double x[N3][N2][K], xx[N3][N2][K], tmp;
  int    isin, isn, icon, i, j, k;

  for (i=0; i<N3; i++) {
    for (j=0; j<N2; j++) {
      for (k=0; k<N1; k++) {
        xx[i][j][k] = x[i][j][k] = (double)(N1*N2*i+N1*j+k+1);
      }
    }
  }

  isin = 1;
  isn  = 1;
  c_dm_v3drcf((double*)x, K, N1, N2, N3, isin, isn, &icon);
  printf("icon = %d\n", icon);

  isn  = -1;
  c_dm_v3drcf((double*)x, K, N1, N2, N3, isin, isn, &icon);
  printf("icon = %d\n", icon);

  tmp = 0.0;
  for (i=0; i<N3; i++) {
    for (j=0; j<N2; j++) {
      for (k=0; k<N1; k++) {
```

```
            tmp = max(fabs(x[i][j][k]/(double)N1/(double)N2/(double)N3-xx[i][j][k]),tmp);
          }
        }
       }

      printf("error = %e\n", tmp);

      return(0);
    }
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_V3DRCF の項目を参照してください.

c_dm_v3drcf2

3 次元離散型実フーリエ変換 (2, 3, 5 及び 7 の混合基底)
`ierr = c_dm_v3drcf2(x, k1, k2, nl, n2, n3,` `isin, isn, &icon);`

1. 機能

3 次元実フーリエ変換の正変換または逆変換を, 混合基底 FFT により行います. 3 次元データ(n_1, n_2, n_3)の各次元の大きさは, 2, 3, 5 及び 7 の巾の積で表すことのできる数です.

a) 3 次元フーリエ変換

$\{x_{j1j2j3}\}$を入力し, (1)で定義する変換を行い, $\{n_1n_2n_3\alpha_{k1k2k3}\}$を求めます.

$$
\begin{aligned}
n_1n_2n_3\alpha_{k1k2k3} = \sum_{j1=0}^{n1-1}\sum_{j2=0}^{n2-1}\sum_{j3=0}^{n3-1} x_{j1j2j3}\omega_{n1}^{-j1k1r}\omega_{n2}^{-j2k2r}\omega_{n3}^{-j3k3r} \\
,k_1 = 0,1,...,n_1-1 \\
,k_2 = 0,1,...,n_2-1 \\
,k_3 = 0,1,...,n_3-1 \\
,\omega_{n1} = \exp(2\pi i/n_1) \\
,\omega_{n2} = \exp(2\pi i/n_2) \\
,\omega_{n3} = \exp(2\pi i/n_3) \\
,r = 1 \quad \text{または} \quad r = -1
\end{aligned}
\tag{1}
$$

b) 3 次元フーリエ逆変換

$\{\alpha_{k1k2k3}\}$を入力し, (2)で定義する変換を行い, $\{x_{j1j2j3}\}$を求めます.

$$
\begin{aligned}
x_{j1j2j3} = \sum_{k1=0}^{n1-1}\sum_{k2=0}^{n2-1}\sum_{k3=0}^{n3-1} \alpha_{k1k2k3}\omega_{n1}^{j1k1r}\omega_{n2}^{j2k2r}\omega_{n3}^{j3k3r} \\
,j_1 = 0,1,...,n_1-1 \\
,j_2 = 0,1,...,n_2-1 \\
,j_3 = 0,1,...,n_3-1 \\
,\omega_{n1} = \exp(2\pi i/n_1) \\
,\omega_{n2} = \exp(2\pi i/n_2) \\
,\omega_{n3} = \exp(2\pi i/n_3) \\
,r = 1 \quad \text{または} \quad r = -1
\end{aligned}
\tag{2}
$$

2. 引数

呼出し形式:

```
ierr = c_dm_v3drcf2((double*)x, k1, k2, nl, n2, n3, isin, isn, &icon);
```

引数の説明:

x	double x[n3][k2][k1]	入力 ／出力	3 次元実データ. データを, x[i][j][k], i=0, … , n3–1, j=0, … , n2–1, k=0, … , n1–1 に格納します. 実数から複素数への変換(isn=1)のとき入力, 複素数から実数への変換(isn=–1)のとき出力となります.

		出力 ／入力	変換された複素データの実部および虚部. x を x[n3][k2][k1/2][2]なる配列とみなして, 実部は x[i][j][k][0], i=0, … , n3–1, j=0, … , n2–1, k=0, … , n1/2 に, 虚部は x[i][j][k][1], i=0, … , n3–1, j=0, … , n2–1, k=0, … , n1/2 に格納されます. 実数から複素数への変換(isn=1)のとき出力に, 複素数から実数への変換(isn=–1)のとき入力になります. フーリエ変換された複素数データには共役関係があり, 約半分が格納されます.
k1	int	入力	データ配列 x の 3 次元目の大きさ ($\geq 2\times$(n1/2+1)). 偶数.
k2	int	入力	データ配列 x の 2 次元目の大きさ ($\geq$ n2).
n1	int	入力	変換する 3 次元データの 1 次元目の大きさ. n1 は 2, 3, 5 及び 7 の巾の積で表すことのできる数.
n2	int	入力	変換する 3 次元データの 2 次元目の大きさ. n2 は 2, 3, 5 及び 7 の巾の積で表すことのできる数.
n3	int	入力	変換する 3 次元データの 3 次元目の大きさ. n3 は 2, 3, 5 及び 7 の巾の積で表すことのできる数.
isin	int	入力	変換の方向を表します. 1 のとき　: $r=1$ –1 のとき　: $r=-1$
isn	int	入力	変換か逆変換かを指定します. 変換　: 1 逆変換 : –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. • k1 < 2×(n1/2+1) • k1 が偶数でない. • k2 < n2 • n1 < 1 • n2 < 1 • n3 < 1 • isin ≠ 1, –1 • isn ≠ 1, –1	処理を打ち切る.
30008	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

3 次元離散型複素フーリエ変換および, 逆変換は一般的に(3), (4)で定義されます.

$$\alpha_{k1k2k3} = \frac{1}{n_1 n_2 n_3} \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \sum_{j3=0}^{n3-1} x_{j1j2j3} \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \omega_{n3}^{-j3k3}$$
$$, k_1 = 0, 1, ..., n_1 - 1 \quad (3)$$
$$, k_2 = 0, 1, ..., n_2 - 1$$
$$, k_3 = 0, 1, ..., n_3 - 1$$

$$x_{j1j2j3} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \sum_{k3=0}^{n3-1} \alpha_{k1k2k3} \omega_{n1}^{j1k1} \omega_{n2}^{j2k2} \omega_{n3}^{j3k3}$$
$$, j_1 = 0, 1, ..., n_1 - 1 \quad (4)$$
$$, j_2 = 0, 1, ..., n_2 - 1$$
$$, j_3 = 0, 1, ..., n_3 - 1$$

ここで, $\omega_{n1} = \exp(2\pi i/n_1)$, $\omega_{n2} = \exp(2\pi i/n_2)$, $\omega_{n3} = \exp(2\pi i/n_3)$

本関数では, (3), (4)の左辺に対応して$\{n_1 n_2 n_3 \alpha_{k1k2k3}\}$または$\{x_{j1j2j3}\}$を求めます. したがって, 結果の正規化は必要に応じて行ってください.

b) 複素共役について

3 次元の実データのフーリエ変換結果には次の複素共役(¯で示します)の関係があります.

$$\alpha_{k1k2k3} = \overline{\alpha_{n1-k1\, n2-k2\, n3-k3}}$$

$k_1 = 0, \ldots, n_1/2, k_2 = 0, \ldots, n_2-1, k_3 = 0, \ldots, n_3-1$ のデータから残りのデータを計算することができます.

4. 使用例

3 次元実 FFT を計算します.

```
#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 N1  (128)
#define N2  (N1)
#define N3  (N1)
#define K1  ((N1/2+1)*2)
#define K2  (N2+1)

MAIN__()
{
  double x[N3][K2][K1], xx[N3][K2][K1], tmp;
  int    isin, isn, icon, i, j, k;

  for (i=0; i<N3; i++) {
    for (j=0; j<N2; j++) {
      for (k=0; k<N1; k++) {
        xx[i][j][k] = x[i][j][k] = (double)(N1*N2*i+N1*j+k+1);
      }
    }
  }

  isin = 1;
  isn  = 1;
  c_dm_v3drcf2((double*)x, K1, K2, N1, N2, N3, isin, isn, &icon);
  printf("icon = %d\n", icon);

  isn  = -1;
  c_dm_v3drcf2((double*)x, K1, K2, N1, N2, N3, isin, isn, &icon);
  printf("icon = %d\n", icon);

  tmp = 0.0;
  for (i=0; i<N3; i++) {
    for (j=0; j<N2; j++) {
```

```
            for (k=0; k<N1; k++) {
              tmp = max(fabs(x[i][j][k]/(double)N1/(double)N2/(double)N3-xx[i][j][k]),tmp);
            }
          }
         }

        printf("error = %e\n", tmp);

        return(0);
      }
```

5. 手法概要

計算方法の詳細については“FUJITSU SSL II スレッド並列機能使用手引書”の DM_V3DRCF2 の項目を参照してください.

付録 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します. 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.

[1] P.AMESTOY, M.DAYDE and I.DUFF
Use of computational kernels in the solution of full and sparse linear equations, M.COSNARD, Y.ROBERT, Q.QUINTON and M.RAYNAL, PARALLEL & DISTRIBUTED ALGORITHMS, North-Holland, 1989, pp.13-19.

[2] P.R.AMESTOY and C.PUGLISH
AN UNSYMMETRIZED MULTIFRONTAL LU FACTORIZATION, SIAM J. MATRIX ANAL. APPL. Vol. 24, No. 2, pp. 553-569, 2002

[3] A.A.Anda and H.Park
Fast Plane Rotations with Dynamic Scaling, to appear in SIAM J. Matrix Analysis and Applications, 1994.

[4] S.L.Anderson
Random number generators on vector supercomputers and other advanced architectures, SIAM Rev. 32 (1990), 221-251.

[5] C.Ashcraft
The distributed solution of linear systems using the torus wrap data mapping, Tech. Report ECA-TR-147, Boeing Computer Services, October 1990.

[6] O.Axelsson and M.Neytcheva
Algebraic multilevel iteration method for Stieltjes matrices. Num. Lin. Alg. Appl., 1:213-236, 1994.

[7] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, editors.
"Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide." SIAM, Philadelphia, 2000.

[8] Ǎ.Björck
Solving linear least squares problems by Gram-Schmidt orthogonalization, BIT, 7:1-21, 1967.

[9] R.P.Brent
Uniform random number generators for supercomputers, Proc. Fifth Australian Supercomputer Conference, Melbourne, Dec. 1992, 95-104.

[10] R.P.Brent
Uniform random number generators for vector and parallel computers, Report TR-CS-92-02, Computer Sciences Laboratory, Australian National University, Canberra, March 1992

[11] R.P.Brent
Fast normal random number generators on vector processors, Technical Report TR-CS-93-04, Computer Sciences Laboratory, Australian National University, Canberra, March 1993.

[12] R.P.Brent
A Fast Vectorised Implementation of Wallace's Normal Random Number Generator, Technical Report, Computer Sciences Laboratory, Australian National University, to appear.

[13] R.Burkard, M.Dell'Amico and S.Martello
Assignment Problems, SIAM Philadelphia, 2009

[14] J.Choi, J.Dongarra, R.Pozo, and D.Walker
ScaLAPACK : A scalable linear algebra library for distributed memory concurrent computers., Technical Report 53, LAPACK Working Note, 1993.

[15] A.Cleary
A comparison of algorithms for Cholesky factorization on a massively parallel MIMD computer, Parallel Processing for Scientific Computing, 1991.

[16] A.Cleary
A Scalable Algorithm for Triangular System Solution Using the Torus Wrap Mapping, ANU-CMA Tech Report, series 1994.
[17] T.H.CORMEN, C.E.LEISERSON, R.L.RIVEST and C.STEIN
INTRODUCTION TO ALGORITHMS, SECOND EDITION, The MIT Press, 2001
[18] J.K.Cullum and R.A.Willoughby
“Lanczos algorithm for large symmetric eigenvalue computations”, Birkhauser, 1985.
[19] T.Davis
Direct Methods for Sparse Linear Systems, SIAM 2006.
[20] J.Demmel and W.Kahan
Accurate singular values of bidiagonal matrices, SISSC 11, 873-912, 1990.
[21] J.J.Dongarra and R.A.Van de Geijn
Reduction to condensed form for the eigenvalue problem on distributed memory architectures, Parallel Computing, 18, pp.973-982, 1992.
[22] I.S.DUFF, A.M.ERISMAN and J.K.REID
Direct Methods for Sparse Matrices, OXFORD SCIENCE PUBLICATIONS, 1986
[23] I.S.DUFF and J.KOSTER
ON ALGORITHMS FOR PERMUTING LARGE ENTRIES TO THE DIAGONAL OF A SPARSE MATRIX, SIAM J. MATRIX ANAL. APPL. Vol. 22, No. 4, pp. 973-996, 2001
[24] A.M.Ferrenberg, D.P.Landau and Y.J.Wong
Monte Carlo simulations: Hidden errors from “good” random number generators, Phys. Rev. Lett. 69 (1992), 3382-3384.
[25] G.Fox
Square matrix decomposition - Symmetric, local, scattered, Caltech Publication Hm-97, California Institute of Technology, Pasadena, CA, 1985.
[26] R.Freund
“A transpose-free quasi-minimal residual algorithm for nonhermitian linear systems”, SIAM J.Sci.Comput. 14, 1993, pp.470-482.
[27] R.Freund and N.Nachtigal
“QMR: a quasi minimal resudual method for non-Hermitian linear systems”, Numer. Math. 60, 1991, pp.315-339.
[28] K.A.Gallivan, R.J.Plemmons, and A.H.Sameh
Parallel Algorithms for Dense Linear Algebra Computations, SIAM Review, 1990.
[29] Martin B. van Gijzen and Peter Sonneveld
"An elegant IDR(s) variant that efficiently exploits bi-orthogonality properties",
Delft university of technology, Report 08-21, 2008.
[30] G.H.Golub, C.F.van Loan
Matrix Computations Second Edition, The Johns Hopkins University Press, 1989.
[31] Marcus J. Grote and Thomas Huckle
"Parallel preconditioning with sparse approximate inverse",
SIAM J. Sci. Comput., Vol.18, No.3, pp838-853, May 1997.
[32] M.H.Gutknecht
Variants of BiCGStab for matrices with complex spectrum, IPS Research report No. 91-14, 1991.
[33] E. Hairer, S.P.Norsett, and G. Wanner
"Solving Ordinary Differential Equations I: Nonstiff Problems." Second Revised Edition, Springer, 2000.
[34] E. Hairer, and G. Wanner
“Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems.” Second Revised Edition, Springer, 2002.
[35] E.ハイラー/S.P.ネルセット/G.ヴァンナー著, 三井斌友 監訳
常微分方程式の数値解法 I : 基礎編 , Springer , 2007

[36] E.ハイラー/G.ヴァンナー著, 三井斌友 監訳
常微分方程式の数値解法 II : 発展編 , Springer , 2008
[37] Markus Hegland
An implementation of multiple and multi-variate Fourier transforms on vector processors, submitted to SIAM J.Sci. Comput., 1992.
[38] Markus Hegland
Block Algorithms for FFTs on Vector and Parallel Computers. PARCO 93, Grenoble, 1993.
[39] Markus Hegland
On the parallel solution of tridiagonal systems by wrap-around partitioning and incomplete LU factorization, Numer. Math. 59, 453-472, 1991.
[40] B.Hendrickson and D.Womble
The torus-wrap mapping for dense matrix calculations on massively parallel computers, SAND Report SAND 92-0792, Sandia National Laboratories, Albuquerque, NM, 1992.
[41] J.R.Heringa, H.W.J.Blöte and A.Compagner
New primitive trinomials of Mersenne-exponent degrees for random-number generation, International J. of Modern Physics C 3 (1992), 561-564.
[42] F.James
A review of pseudorandom number generators, Computer Physics Communications 60 (1990), 329-344.
[43] G.KARYPIS AND V.KUMAR
A fast and high quality multilevel scheme for partitioning irregurar graphs, SIAM J. Sci. Comput., 20 pp.359-392, 1998
[44] G.KARYPIS AND V.KUMAR
METIS
A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices
Version 4.0
University of Minnesota, Department of Computer Science / Army HPC Research Center
Minneapolis, MN 55455
Septenmber 20, 1998
[45] D.Kincaid, T.Oppe
ITPACK on supercomputers, Numerical methods, Lecture Notes in Mathematics 1005 (1982).
[46] D.E.Knuth
The Art of Computer Programming, Volume 2: Seminumerical Algorithms (second edition). Addison-Wesley, Menlo Park, 1981, Sec. 3.4.1, Algorithm P.
[47] Z.Leyk
Modified generalized conjugate residuals for nonsymmetric systems of linear equations, in Proceedings of the 6th Biennial Conference on Computational Techniques and Applications: CTAC93, D.Stewart, 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.
[48] X.S.Li AND J.W.DEMMEL
A scalable sparse direct solver using static pivoting, in Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing, San Antonio, Texas, 1999, CD-ROM, SIAM, Philadelphia, PA, 1999
[49] Charles Van Loan
Computational Frameworks for the Fast Fourier Transform, SIAM, 1992.
[50] F.T.Luk
Computing the Singular-Value Decomposition on the ILIAC IV, ACM Trans. Math. Softw., 6, 1980, pp.259-273.
[51] F.T.Luk and H.Park
On Parallel Jacobi Orderings, SIAM J.Sci. Comput., 10, 1989, pp.18-26.

[52] N.K.Madsen, G.h.Rodrigue, and J.I.Karush
"Matrix multiplication by diagonals on a vector/parallel processor" , Information Processing Letters, vol.5, 1976, pp.41-45.
[53] G.Marsaglia
A current view of random number genetators, Computer Science and Statistics: The Interface (edited by L.Billard), Elsevier Science Publishers B.V. (North-Holland), 1985, 3-10.
[54] M.Nakanishi, H.Ina, K.Miura
A high performance linear equation solver on the VPP500 parallel supercomputer, Proceedings of Supercomputing' 94, Washington D.C., Nov. 1994.
[55] 中西誠 , 三上次郎
ブロッキング LU 分解法の VP2000 シリーズ向けチューニングについて情報処理学会第 42 回全国大会 , 1991
[56] 中西誠 , 三上次郎
線型方程式の高速化技術 , 情報処理学会研究報告 , 91-OS-54, 情処研報 Vol.92, No.22, pp.33-40, 1992
[57] M.OLSCHOWKA and A.NEUMAIER
A new pivoting strategy for Gaussian elimination, Linear Algebra Appl., 240(1996), pp.131-151
[58] T.Oppe, W.Joubert and D.Kincaid
An overview of NSPCG: a nonsymmetric preconditioned conjugate gradient package, Computer Physics communications 53 p283 (1989).
[59] T.C.Oppe and D.R.Kincaid
"Are there iterative BLAS?", Int. J. Sci. Comput. Modeling (to appear or has appeared).
[60] M.R.Osborne
Solving least squares problems on parallel vector processors, Area 4 working notes no. 17, 1994.
[61] M.R.Osborne
Computing the eigenvalues of tridiagonal matrices on parallel vector processors, Mathematics Research Report No. MRR 044-94, Australian National University, 1994.
[62] J.R.Rice and R.F.Boisvert
Solving Elliptic Problems Using Ellpack, Springer-Verlang, New York, 1985.
[63] D. Ruiz
A scaling algorithm to equilibrate both rows and columns norms in matrices, Tech. rep. RAL-TR-2001-034, Rutherford Appleton Laboratory, Chilton, U.K., 2001
[64] Y.Saad
ILUT: A dual threshold incomplete LU factorization.Research Report UMSI 92/38, University of Minnesota, Supercomputer Institute, 1200 Washington Avenue South, Minneapolis, Minnesota 55415, USA, 1992.
[65] Y.Saad
ILUM: A multi-elimination ILU preconditioner for general sparse 591 matrices. SIAM J. Sci. Comput., 17:830-847, 1996.
[66] Y.Saad
"Iterative methods for sparse linear systems, second edition",
Univ.Minnesota,SIAM, 2003
[67] Y.Saad and M.H.Schultz
"GMRES : a generalized minimal residual algorithm for solving nonsymmetric linear systems" , SIAM J. Sci. Stat. Comput. 7, 1986, p.856-869.
[68] O.Schenk , K.Gärtner
Solving unsymmetric sparse systems of linear equations with PARDISO, Future Genration Computer Systems 20(2004)475-487
[69] J.A.SCOTT
Scaling and Pivoting in an Out-of-Core Sparese Direct Solver
ACM Transactions on Mathematical Software, Vol. 37, No. 2, Article 19, April 2010

[70] 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
[71] H.D.Simon
Bisection is not optimal on vector processors, SISSC 10, 205-209, 1989.
[72] G. Sleijpen, D. Fokkema
BCG for linear equations involving unsymmetric matrices with complex spectrum, Electronic Transactions on Numerical Analysis, 1 p11 1993
[73] Gerard L.G. Sleijpen and Martin B. van Gijzen
"Exploiting BICGSTAB(l) Strategies to Induce Dimension Reduction",
Delft university of technology, Report 09-02, 2009.
[74] Tomohiro Sogabe, Shao-Liang Zhang
"A COCR method for solving complex symmetric linear systems",
Journal of Computational and SIAM Applied Mathematics, 199(2007)297-303.
[75] J.C. Strikwerda
Finite Difference Schemes and Partial Differential Equations. Wadsworth and Brooks/Cole, Pacific Grove, 1989.
[76] Paul N.Swarztrauber
Multiprocessor FFTs. Parallel Comput. 5, 197-210, 1987.
[77] H.A.Van Der Vorst
“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
[78] C.S.Wallace
“Fast Pseudo-Random Generators for Normal and Exponential Variates”, ACM Trans. on Mathematical Software 22 (1996), 119-127.
[79] R.Weiss
Parameter-Free Iterative Linear Solvers. Mathematical Research, vol. 97. Akademie Verlag, Berlin, 1996.
[80] J.H.Wilkinson
The Algebraic Eigenvalue Problem, O.U.P., 1965.
[81] B.B.Zhou and R.P.Brent
A Parallel Ordering Algorithm for Efficient One-Sided Jacobi SVD Computations, to appear in Proc. Sixty IASTED-ISMM International Conference on Parallel and Distributed Computing Systems, 1994.
[82] K. Miura
Full Polynomial Multiple Recursive Generator(MRG) Revisited, MCQMC 2006, Ulm, Germany
[83] Kenta Hongo, Ryo Maezono, and Kenichi Miura
Random Number Generators Tested on Quantum Monte Carlo Simulations, Journal of Computational Chemistry, 31, 2186-2194, 2010
[84] P. L'Ecuyer and R. Simard
TestU01: A C Library for Empirical Testing of Random Number Generators, ACM Transactions on Mathematical Software, Vol. 33, article 22, 2007.