C言語用行列演算ライブラリ
(A matrix operation library for C)

matlib

— matlib(hanahara) —

花原 和之
(Kazuyuki HANAHARA)

hanahara@cs.kobe-u.ac.jp

User's manual: ver.0.401.

目 次

第1章　はじめに

行列演算—理工系で研究活動を行っている多くの人が直面する問題である. いわゆる大型計算機で FORTRAN を使う場合であれば, ひととおりの行列演算パッケージは用意されている場合が多いであろう. しかしながら小型のワークステーション, PC で利用できる言語処理系で当初からこういった機能が付属している場合はほとんどない. その場合, そのためのソフトウェアを購入あるいはダウンロードし, インストールして使用するか, あるいは自作することとなる. また, 最近では各種のプログラミング関係の書物も多く出版されており, 行列演算を扱ったものも多い. これらを手動入力し, あるいは付属の FD や CD-ROM 等の内容をコピーしてコンパイルする場合もあるであろう.

しかしながら, 市販のソフトウェアは巨大で, いわゆる「かゆいところに手が届きにくい」という難点がある. 「ちょっとした演算がしたい」だけの場合には, そこまでの購入費用やインストールの手間をかける気にならない場合もあるであろう. また, 「行列演算パッケージ」として公開されているもののいくつかは基本がインタプリタになっており, 想定されている範囲で利用する場合はともかく, 少し凝ったことをする場合には制御構造等の構文の理解が必要となる場合もあり, 自作のプログラム等に組み込んで利用する際には苦労する場合が少なくない. 各種書物を参考にしたり, あるいはそういった書籍に付属のプログラムを利用する場合には, 別の問題がある. そのようなプログラムにおいては行列が配列を使って実現されていることがほとんどであるが, その場合には「添字は 0 から」「行列の大きさは固定」といった制約がある. 通常の数式での行列の添字が 1 からである点や, 扱う行列の大きさを変える度にコンパイルしなおす必要を考えると, あまり使い勝手がいい, とは言えないだろう. (参考にする書籍によっては, あまり美しくないプログラムや, バグのあるプログラムが掲載されている場合すらある)

この matlib 行列演算ライブラリ群[1] は, 作者である花原が大学院修士課程で知的適応トラスの研究を始めたときに基本部分を作成し, その後必要に応じて修正, 拡張を施してしてきたものである. プログラムの全ては古い (ANSI-C 以前の) 書式の C 言語で書かれている[2] . 関数名等の識別が 8 文字に収まったりしていないほかは, 構造体の代入なども使っておらず, 少々古い処理系でも動作する (はずである). いわゆる「商品となるような」プログラムや, 非常に大規模の演算をするような場合はともかくとして, 大学の研究室等で使う分には十分の能力がある (はずである). 以下にその特長を示す.

古い C でも安心　識別子の長さをのぞけば, 古い処理系でも動作する. もちろん, 新しい処理系でも問題なく動作する (処理系によっては Warning を抑制するコンパイルオプションが必要かも知れないが).

添字は 1 から　行列の扱いを配列ではなく構造体とマクロで実現し, 添字は数式と同様に 1 からとして扱えるようにしている.

大きさも自由自在　領域確保に malloc を使っているので, コンパイルしなおすことなく, 処理に応じて自由な大きさの行列を扱うことができる.

各種演算をサポート　加減乗算, 逆行列, 固有値計算といった基本演算の他に, 一般化逆行列, Newton 法, 最急降下法, 高次元空間での外積等の演算を備えている.

バンドマトリクス　全ての演算ではないが, 巨大行列を扱うためのバンドマトリクスが利用可能.

[1] 実はこの matlib という名前は行列演算パッケージをはじめとして様々な場所で既に使われている (例えば数学 (math) 関係, 材料 (material) 関係等) が, これまでに書いてきたプログラムを書き換える気力もなく, そのままにしてある. 区別する必要がある場合には表紙にもあるように matlib(hanahara) とする.

[2] 現在 ANSI 対応版を準備中 (といっても基本的にプロトタイプを整備するだけなのだが).

ファイル入出力 内部 (バイナリ) 形式, テキスト形式でのファイルの入出力関数を装備.

ただし, 下記の点には改善の余地が残されている.

確保した領域の返還 行列のために確保した領域はシステムに返還する必要がある. これは往々にして忘れられる場合が多く, 注意が必要.

計算効率 現在のところ, 例えば, 線形方程式の解法としては「Gauss の消去法」, 逆行列の場合は「LU 分解法」, 固有値計算は「べき乗法」によるものしかサポートしていない. 他の解法については折を見てサポートしてゆく予定である.

対称行列 行列の対称性を活用して計算量やメモリの利用効率を高めるようなプログラム構造になっていない

特殊関数 制御系の設計に必要なリッカチ方程式や統計処理に必要な各種演算を扱うための関数等は装備されていない (過去の研究の関係で装備されている特殊関数は若干あるが). これらは, 場合によっては将来装備されるかもしれない.

なお, matlib を使用するにあたっては, (当然であるが) ある程度の C 言語の基本知識が必要である. コンパイル・リンク・ライブラリの利用の仕方程度は最低限の知識として必要であり, できればポインタ・構造体・マクロ程度の知識は修得していることが望ましい. また, (最近はほとんどないが) 処理系によってはヘッダ・ファイルの名称等が若干異なる場合がある.

この matlib 関数群は個人的用途に応じて拡張・改変を行ってきたため, 書式等の統一性には留意してきたものの, 様々な意味で若干「くせのある」ものとなっている. 例えば, 非線形計画法のためのシンプレックス法の関数はあるが, よりポピュラーなはずの線形計画法のためのシンプレックス法の関数はない, といったようなこともある. また, 基本的には普段使っているものをそのまま公開することにしたためにデバッグ用に埋め込んだコードがそのままになっている箇所 (たいていは`#ifdef` やコメントアウトで逃げているが) や, 個人的にも修正しようと思いつつ, 過去の因縁のために修正できていない箇所もある. 使用に際してこの点はご理解いただきいたい.

この文書に示した関数群は, ものによっては 10 年以上も使い込んでいるため, その信頼性はかなり高いと言えるが, 個人的に使用していたものであり, バグが潜んでいる可能性は否定できない[3] . バグが発見された場合には, 詳細な情報とともに連絡していただければ, 可能な範囲で対処する所存である. **なお, 作者である花原はこのプログラムの使用による結果について責任は持てないので注意すること.**

[3] 実際, 今回公開に際して見直していたところで発見されたバグもないではなかった.

第2章　コンパイル

2.1　アーカイブの解凍・展開

現在, アーカイブとして matlib.lzh と matlib.zip を用意している. **どちらの内容も基本的に同一となっているが, ディレクトリ (フォルダ) 情報のみが異なる**. matlib.lzh にはディレクトリ情報は含まれていないため, (状況に応じて) 適当なディレクトリを作成してその下で解凍すればよい. また, matlib.zip のほうは, 展開すると matlib というディレクトリの下にファイルを生成する.

アーカイブの解凍・展開は, unix 系の OS の場合, たぶん lha か unzip か少なくともどちらかは利用可能であると思うので,

```
lha x matlib.lzh
```

あるいは

```
unzip matlib.zip
```

とすれば解凍・展開ができる. Windows の場合, lha が利用可能であればそれを用いてもよいし (必要とあればいくつかのサイトからダウンロードできると思う), zip 形式のファイルは特にユーティリティを用意しなくても展開することができるようである.

このあたり (ファイルの解凍・展開) についての詳細な情報については適当な書籍やサイト等を参照されたい.

2.2　ファイル

matlib 関数群を構成するファイルを以下に示す.

matrix.h, matrix.c　最も基本となる関数群. 行列のための領域確保や解放, 入出力, エラー処理, 等々.

operate.h, operate.c　基本演算関数群. 転置, 加減乗算等およびそのバリエーション.

inverse.h, inverse.c　逆行列関係. Gauss の消去法による線形方程式の解, LU 分解法による逆行列, 高次空間における外積等.

ginverse.h, ginverse.c　一般化逆行列関係. Moore-Penrose 一般化逆行列, 疑似逆行列, ノルム最小化型一般化逆行列等.

eigen.h, eigen.c　べき乗法による固有値の計算とその関連関数群.

newton.h, newton.c　Newton 法によるベクトル値非線形方程式の解.

rotate.h, rotate.c　二次元および三次元の回転行列, 微分, 逆変換等.

lsa.h, lsa.c　べき級数関数の最小二乗法.

newmark.h, newmark.c　ニューマーク β 法による数値積分.

steepest.h, steepest.c　最急降下法とその周辺.

udrand.h, udrand.c　乱数関係. 標準ライブラリ関数である rand() を使って (0, 1) や (−1, 1) の実数値乱数や正規乱数を発生する関数群.

simplex.c　非線形計画法におけるシンプレックス法. ヘッダファイルがないのは, 関数 Simplex() が int 型だからである. (C は何も宣言しない関数には暗黙のうちに int 型であると仮定して処理するから. このあたり, 古い書き方のなごりと言えなくもない)

intmat.h, intmat.c　整数値行列を取扱う関数.

matlib.h　ヘッダファイルをまとめたもの. これを#include することにより, 個々の関数に対するヘッダファイルを意識する必要はなくなる.

makefile.uni, makefile.bcc　メイクファイルのサンプル. makefile.uni は unix 系 OS で gcc を想定, makefile.bcc は Borland の処理系 bcc32 を想定したものである. 利用する場合には適切なものを makefile と名前を変更して使う. make コマンドを参照のこと.

testmat.c　テストプログラム.

mat.c　逆ポーランド記法にもとづく簡易行列電卓もどき.

2.3 make によるコンパイル

2.3.1 unix 系 OS の場合

unix 系の OS を使っている場合であれば, 全てのファイルを同一ディレクトリにおき (普通に解凍・展開を行っていればそうなっているはずだが), そのディレクトリで以下のコマンドを実行すればライブラリのアーカイブ matlib.a が生成される.

```
cp makefile.uni makefile
make matlib.a
```

なお, 付属の makefile.uni では, 処理系として gcc を仮定し, ranlib を使用している. また, コンパイラの最適化オプション として-O を用いている. コンパイルができない場合は適宜 makefile を修正する必要がある. 例えば gcc ではなくワークステーションないし OS 付属の cc を使う場合であれば,

```
CC = gcc -O -DERROR_MESSAGE -DBAND
```

の gcc を cc にすればよい.

2.3.2 Windows の場合 (bcc32 利用)

Windows の場合, コンパイルやプログラムの実行環境としてコマンドプロンプト (いわゆる DOS 窓) を想定している.

処理系として, 無償でダウンロードして利用できる Borland 社の C/C++の処理系である Borland C++ 5.5(バージョンは変更されている可能性あり) の場合も make が使える. この場合には bcc32.exe と make.exe のあるディレクトリ (デフォルトの場合はC:\Borland\bcc55\Bin) に PATH が通っている必要がある. unix の場合と同様, 全てのファイルを同一ディレクトリにおき, そのディレクトリで以下のコマンドを実行すればライブラリファイル matlib.lib が生成される.

```
copy makefile.bcc makefile
make matlib.lib
```

2.4 make を用いない場合

makefile.uni や makefile.bcc を参考にして適当なスクリプト (shell スクリプトやバッチファイル) を作成して用いるか, 手動でコンパイルすることも可能. このあたりは環境に大きく依存するのでここでは扱わない. ただし, リンク時やライブラリ作成時にオブジェクトファイル間の参照順序が問題となる場合には注意が必要. この場合には, makefile.uni における

```
OBJS = udrand.o matrix.o operate.o inverse.o ginverse.o eigen.o newton.o \
       rotate.o newmark.o simplex.o steepest.o lsa.o intmat.o
```

を参照するとよい. 先に書かれているオブジェクトファイルが後に書かれているオブジェクトファィルに参照される. (オブジェクトファイルの拡張子は処理系によっては ".obj"(Borland C++) やその他となる場合もある)

また, matlib は (現在のところ) 基本的に古い (ANSI 以前の) 書式のため, 処理系によっては大量に Warning を出す場合がある. このような場合 Warning を抑制するコンパイルオプション等を用いるほうがよい. なお, makefile.bcc ではプロトタイプに関する Warning を抑制するために-w-pro というオプションを用いている.

2.5 コンパイルオプション

コンパイル時に-D オプション (gcc や bcc32 の場合) 等を用いたキーワード定義により, 生成されるコードの下記の点を制御することができる. (makefile.uni や makefile.bcc を参照)

ERROR_MESSAGE　エラーメッセージを標準出力 (stdout) に出力する. ただし, エラーチェックの程度は matlib 関数によって異なる. エラーメッセージを標準出力以外 (標準エラー出力やファイル等) に出力することもできる. (下記の MATERR_VAR を参照)

NO_STDIO　標準入出力関数をリンクしない. このキーワードを定義した場合, ファイル入出力関数は使用できない. また, たとえ ERROR_MESSAGE が定義されていたとしても, エラーメッセージの出力は行われなくなる. 組み込み型のプログラム等の際に使う.

BAND　バンドマトリクスを扱う. このキーワードを定義することにより, バンドマトリクスが使用可能となる. 定義されていない場合はバンドマトリクスを扱うためのコードは生成されない.

GDB_WHERE　キーワード ERROR_MESSAGE が定義されている場合, 通常は, エラーを直接検出した関数がどのように呼び出されたか等をある程度表示した後に停止することになっている. しかし, このキーワードが定義されている場合には, エラーを直接検出した関数内で Segmentation fault を起こして停止する. デバッガとして gdb を使う場合にはコンパイラオプションとして-g とともに-DGDB_WHERE を指定するほうがデバッグには便利である.

MATERR_VAR　キーワード ERROR_MESSAGE が定義されている場合のエラーメッセージの出力先は, 通常, 標準出力に固定されている. しかし, このキーワードの定義により, エラーメッセージの出力先を動的に変更することが可能になる. この場合, matlib ライブラリ関数のヘッダファイル matlib.h のインクルードにより, エラーメッセージの出力先を示すファイルポインタ変数 materr が外部変数として暗黙のうちに宣言されるので, この変数に希望する出力先, 例えば標準エラー出力や特定のファイルのファイルポインタ等を代入すればよい. プログラムで標準出力に大量のデータを出力する場合に matlib 関数のエラーメッセージを切り分ける場合などに利用できる. ただし, この場合にはプログラムの最初で関数 InitMaterr 等により, materr の初期設定を行っておく必要がある.

BAND_ERROR_TEST　このキーワードが定義されている場合, MVAL や BVAL といったマクロを用いたバンドマトリクスのアクセスの際に定められたバンド幅に収まっているかどうかのチェックを行う. ただし, その場合のマクロは多少複雑であり, 処理時間や生成されるコードのサイズといった観点では非効率的である. このキーワードは, matlib ライブラリのコンパイルには用いずに, アプリケーションプログラムのコンパイルにのみ定義しても有効である.

付属の makefile では, 前述したように, コンパイラの-D オプションは以下のようになっている.

```
-O -DERROR_MESSAGE -DBAND
```

すなわち, エラーメッセージを標準出力に出力し, バンドマトリクスを扱うコードを生成する.

第3章　基本的な使い方

3.1　プログラムのコンパイル

付属のサンプルプログラム testmat.c を例に, コンパイルおよび matlib ライブラリとのリンクについて解説する. なお, gcc および bcc32 を使用する場合で前章で述べたように添付の makefile を用いて matlib のコンパイルを行った場合, gcc であれば

```
make testmat
```

bcc32 であれば

```
make testmat.exe
```

とすればコンパイルが行える. 詳細についてはそれぞれの makefile を参照のこと.

3.1.1　unix 系 OS の場合

unix 系の OS でコンパイラとして gcc(あるいは cc) を使用する場合, サンプルプログラム testmat.c が他の matlib ファイルと同じディレクトリにある場合には

```
gcc testmat.c -o testmat matlib.a -lm
```

とすれば実行形式 testmat が生成される. なお, コンパイルオプション-o は出力ファイルの指定, -lm は標準数学関数ライブラリのリンクを意味する. ライブラリファイル matlib.a および matlib ヘッダファイル群が testmat.c とは異なるディレクトリ, たとえば自分のホームディレクトリの下の matlib というディレクトリにある場合であれば, 上記コンパイル命令は

```
gcc -I$home/matlib testmat.c -o testmat $home/matlib/matlib.a -lm
```

となる. ここで, コンパイルオプション-I はヘッダファイルのサーチパスに含めるディレクトリを指定するオプションである. また, $home は自分のホームディレクトリを表す環境変数である (unix の解説書を参照のこと). さらに, バンドマトリクスを扱う場合には (ただしこの場合, matlib.a 自体が-DBAND オプションを付けてコンパイルされていることが前提),

```
gcc -DBAND -I$home/matlib testmat.c -o testmat $home/matlib/matlib.a -lm
```

となる. このような場合, コマンド入力がやや煩雑になるため, コンパイルを頻繁に行うのであれば makefile を作成し, make コマンドを利用することを強く勧める.

3.1.2　コンパイルする上での留意点

unix 系ではない OS を利用する場合についてはここでは書かない. コンパイルする上での一般的な留意点として下記が挙げられる.

ヘッダファイルのサーチパス　matlib ファイルが存在するのとは異なるディレクトリ (フォルダ!?) でコンパイルする場合, ヘッダファイルを`#include` するためのサーチパスに matlib ファイルのあるディレクトリ (フォルダ!?) を含めるようにする.

ライブラリファイルのリンク　matlib ライブラリをリンクするように明示的に指定する. リンクする他のオブジェクトファイルと一緒に指定するようになっていることが多い.

数学関数ライブラリのリンク　処理系によっては特にオプションを指定しなくても数学関数ライブラリをリンクできる場合もあるが, 特に unix 系の処理系では指定するようになっている場合 (前述の`-lm` オプション等) が多い.

コンパイルオプションの整合性　基本的には matlib ライブラリをコンパイルしたものと同等のオプションを指定する. 例えば, バンドマトリクスを使う場合には, 作成したプログラムのコンパイル時だけではなく, リンクする matlib ライブラリを構成するオブジェクトファイルのコンパイル時にも`-DBAND` が指定されていなければならない (ただし, 逆は可能. 作成したプログラムのコンパイル時に`-DBAND` が指定されていなくても`-DBAND` を指定してコンパイルした matlib ライブラリを使用することはできる).

3.2　サンプルプログラム

3.2.1　実行例

コンパイルされたサンプルプログラム `testmat.c` を実行する. このプログラムの起動の書式は

`testmat` 行列の次元

であるので, 例えば

```
testmat 3
```

として実行する. この場合, 次のような出力が得られるであろう (ただし, 個々の数値は異なる. 理由は後述).

```
--- Matrix A ---
    2.0000     4.0000     2.0000
    4.0000     4.0000     8.0000
    2.0000     8.0000     2.0000
--- Vector B ---
    1.0000
    1.0000
    1.0000
*** Solve A X = B ***
--- X ---
    0.7500
       0.0
   -0.2500
(Determinant :        -32)
--- A X ---
    1.0000
    1.0000
    1.0000
--- Inverse of A ---
    1.7500    -0.2500    -0.7500
   -0.2500        0.0     0.2500
   -0.7500     0.2500     0.2500
--- Inverse * Original ---
```

```
    1.0000        0.0        0.0
       0.0     1.0000        0.0
       0.0        0.0     1.0000
```

このプログラムは行列変数 A を生成し, これを用いた線形方程式を解き, 逆行列を求め, それらの結果を積により検証している. 行列変数 A の値は起動時刻に応じて疑似的にランダムであるため, 起動する毎に結果の出力は異なる. もし, 最初に決定される行列変数 A が特異となった場合にはその行列式 (Determinant) の値は 0 となり, 線形方程式の解や逆行列は得られず, 例えば下記のような出力となる (`Inverse * Original` が単位行列になっていないことに注意).

```
--- Matrix A ---
       0.0     1.0000        0.0
    1.0000     6.0000     3.0000
       0.0     3.0000        0.0
--- Vector B ---
    1.0000
    1.0000
    1.0000
*** Solve A X = B ***
(Determinant :          0)
--- A X ---
       0.0
       0.0
       0.0
--- Inverse of A ---
       0.0     1.0000        0.0
    1.0000     6.0000     3.0000
       0.0     3.0000        0.0
--- Inverse * Original ---
    1.0000     6.0000     3.0000
    6.0000    46.0000    18.0000
    3.0000    18.0000     9.0000
```

3.2.2 プログラム解説

`testmat.c` のふるまいについて解説する. このプログラムの動作が理解できれば, matlib を使う際に必要な基本事項は修得できたといってよい. 以下に `testmat.c` を示す (左端の行番号は説明のために付与したものであり, `testmat.c` の内容ではない. 念のため).

```
 1: /*****************************************************************************
 2:         A test program for matlib, the matrix calculation liblaly.
 3: *****************************************************************************/
 4: /*--- Header files for standard library. ---*/
 5: #include <stdio.h>
 6: #include <stdlib.h>
 7: /*--- Collection of header files for matlib. ---*/
 8: #include "matlib.h"
 9:
10: main( argc, argv )
11: int     argc;
12: char    *argv[];
13: {
14:         matrix  *A, *AA, *AAA;  /* Matix pointers. */
15:         vector  *B, *X;         /* Vector pointers. */
16:         long    t;              /* Used for data initialize. */
17:         int     i, j, n;
18:         double  det;
19:
20:         InitMaterr();           /* For error message output to 'stdout'. */
21:
```

```
22:         /*--- Size of matrices and vectors. ---*/
23:         if( argc == 1 ){
24:                 fprintf( stderr, "testmat number\n" );
25:                 exit( 0 );
26:         }
27:         n = atoi( argv[1] );
28:
29:         /*--- Alloc' n-dimensional matrix and vector. ---*/
30:         A = AllocMatrix( n, n );
31:         B = AllocVector( n );
32:
33:         /*--- Data initialize in a pseudo-random manner. ---*/
34:         t = time( 0 );
35:         for( j = 1; j <= CSIZE(A); j++ ){
36:                 for( i = 1; i <= RSIZE(A); i++ )
37:                         MVAL(A,i,j) = (double)(t % (i * j + 3));
38:                 VVAL(B,j) = 1.0;
39:         }
40:
41:         /*--- Display matrix and vector ---*/
42:         printf( "--- Matrix A ---\n" );
43:         PrintMatrix( A );
44:         printf( "--- Vector B ---\n" );
45:         PrintMatrix( B );
46:
47:         /*--- Alloc' same matrix as 'A'. ---*/
48:         AA = AllocSame( A );
49:
50:         /*--- Solve linear equation. ---*/
51:         X = AllocSize( B );     /* 'X' has the same size as 'B'. */
52:         printf( "*** Solve A X = B ***\n" );
53:         if( (det = Gauss( A, X, B )) != 0.0 ){
54:                 printf( "--- X ---\n" );
55:                 PrintMatrix( X );
56:         }
57:         printf( "(Determinant : %10g)\n", det );
58:
59:         /*--- Confirm result by multiplication. ---*/
60:         printf( "--- A X ---\n" );
61:         MulMatrix( B, AA, X );          /* Result is overwritten on B */
62:         PrintMatrix( B );
63:         /*--- 'B' and 'X' are now no use. Free their alloc'ed memory. ---*/
64:         FreeMatrix( B );
65:         FreeMatrix( X );
66:
67:         /*--- Copy matrix : A = AA ---*/
68:         CopyMatrix( A, AA );
69:
70:         /*--- Calculate inverse of 'A'. ---*/
71:         Inverse( A, 0.0 );
72:         printf( "--- Inverse of A ---\n" );
73:         PrintMatrix( A );
74:         /*--- Check inverse by multiplication. ---*/
75:         printf( "--- Inverse * Original ---\n" );
76:         AAA = MulMatrix( NULL, A, AA ); /* Reuslt is written on new matrix. */
77:         PrintMatrix( AAA );
78:
79:         /*--- End of the main function. Free all matrices and vectors. ---*/
80:         FreeMatrix( AAA );
81:         FreeMatrix( AA );
82:         FreeMatrix( A );
83: }
```

いくつかのポイントを行番号を参照しつつ解説する.

5–6 行　標準ライブラリのヘッダファイルの取込み. 一般に, 必要に応じて`#include`行を用いて各種ヘッダファイルの取込みを行う. 例えば, このソースファイル中のプログラムが数学関数を使う場合にはさらに`math.h`を`#include <math.h>`として取込む必要がある. なお, コンパイルオプション等で`NO_STDIO`を定義しない通常の場合, 少なくとも`stdio.h`を`matlib.h`より前に`#include`する必要がある.

8 行　matlib ライブラリ関数全体に対するヘッダファイル`matlib.h`の取込み. matlib ライブラリを使用するプログラムのファイルでは必ずこの行を記述する.

10–12 行　関数宣言 (古い (ANSI 規格以前の) 書式による). ANSI 規格に準拠した新しい書式[1] では, 以下のようになる.

```
main( int argc, char argv*[] )
```

14 行　行列へのポインタ`A`, `AA`, `AAA`の宣言. 行列・ベクトルを扱う変数は全てポインタとして宣言する. したがって, 例えば「行列の配列」を扱う場合には, 行列へのポインタの配列, あるいは行列へのポインタへのポインタを使う.

15 行　ベクトルへのポインタ. ただし, matlib においては, ベクトルは「列数が 1 の行列」(列ベクトル) として実現されているので, この部分を

```
matrix  *B, *X;
```

と書いても全く問題がない. この表記の違いの存在は, 主としてプログラムの可読性の向上のためのものである.

20 行　エラーメッセージの出力先を標準出力に設定する. **デフォルトのコンパイル条件ではこの行は特になくても構わない**が, matlib ライブラリ関数のコンパイルの際にキーワード`MATERR_VAR`を定義している場合には最初にエラーメッセージの出力先の初期設定を行っておく必要がある. 一種のおまじない.

30 行　行列の領域の割り当て. 関数`AllocMatrix`を用いて行列へのポインタを初期化する. ここではポインタ`A`に`n`×`n`行列を割り当てている. 行列にアクセスして値を設定したりする場合には, 必ず領域確保が必要. ただし, 下記にも示すように, 多くの matlib 関数が, 必要な領域を確保した上で演算結果を格納し, そのポインタを返すことができるようになっている (全てではない). その場合には事前に領域を確保する必要はない.

31 行　ベクトルの領域の割り当て. 関数`AllocVector`を用いてベクトルへのポインタを初期化する. 上記でも述べたように, ベクトルは「列数が 1 の行列」であるので, これは

```
B = AllocMatrix( n, 1 );
```

と書くのと全く等価である.

34–39 行　行列およびベクトルの値の初期設定. 詳細は下記のとおり.

34 行　標準ライブラリ関数`time`を使って, 変数`t`に適当な値を設定する. この値を元に, 行列`A`の乱数風の初期化を行う.

[1] とはいっても, これを書いている時点ですでに 10 年以上が経過しているが….

35 行　全ての列をアクセスするための for ループ. CSIZE(A) は行列 A の列数を示すマクロであり, 整数値である.

36 行　全ての行をアクセスするための for ループ. RSIZE(A) は行列 A の行数を示すマクロであり, 整数値である.

37 行　行列 A の i 行 j 列の要素に t と i, j によって決まる適当な値を代入する. MVAL(A,i,j) は行列 A の i 行 j 列の要素をアクセスするためのマクロであり, ひとつの倍精度実数変数と見なすことができる ((double) を用いた型変換を明示しているのはそのため). **行列およびベクトルの個々の要素は倍精度実数変数 (double) である.** printf や scanf 等を使って行列やベクトルの個々の要素の表示・入力等を行う場合には注意が必要.

38 行　ベクトル B の j 番目の要素に 1.0 を代入する. VVAL(B,j) はベクトル B の j 番目の要素をアクセスするためのマクロである. これは, MVAL(B,j,1) と書くのと全く等価であり, やはりひとつの倍精度実数変数と見なすことができる.

43 行　行列 A の表示. 関数 PrintMatrix を用いることにより, 実行例に示したような様式で行列やベクトルを表示することができる.

48 行　関数 AllocSame を用いて大きさおよび内容が等しい新たな行列やベクトルを生成することができる. ここでは, 行列 A と同じ大きさ, 内容を持つ新たな行列 AA を生成している.

51 行　関数 AllocSize を用いて大きさの等しい新たな行列やベクトルを生成することができる. この関数によって生成された (すなわち, 領域を割り当てられた) 行列・ベクトルは初期化されているわけではないことに注意. ここでは, 連立一次方程式の解を格納するための, ベクトル B と同じ次元のベクトル X を準備している.

53 行　関数 Gauss を用い, Gauss の消去法により線形方程式 AX=B を解く. 解はベクトル X に格納される. この関数は行列 A の行列式の値を返す. 行列 A およびベクトル B の内容は保存されないことに注意.

61 行　関数 MulMatrix を用い, 行列・ベクトルの乗算を行う. AA×X の結果が B に格納される. この書式の場合, ポインタ B にはあらかじめ適切な領域が割り当てられていなければならない.

64 行　関数 FreeMatrix を用い, 割り当てられたり演算の結果として生成された行列・ベクトルの領域の解放を行う. matlib では, **行列・ベクトルの領域の解放は基本的に意図的に行う必要がある.**　ある行列・ベクトルのポインタに割り当てた領域を解放しないまま, そのポインタに新たな領域を割り当てるようなプログラムになっている場合には, システムのヒープ領域を食いつぶす可能性があるので注意が必要.

68 行　関数 CopyMatrix を用いた行列の複写 (代入あるいはコピー). この例では, AA の内容を A に代入している. **複写元と複写先が, いわゆる unix における cp 等のコピーコマンドとは異なっていることに注意**[2] . なお, CopyMatrix は, 複写先の行列が複写元よりも大きい場合には, 警告やエラーも発することなく複写を実行する仕様になっている.

71 行　関数 Inverse による逆行列の計算. 第二引数に, ピボットとして許容する値の大きさを与える. これを 0.0 とする場合, 演算の過程で常にピボットが最大となるように行の入れ替えを行いつつ演算する (ただし, かなりの処理時間の増大になるので特に大規模行列の場合には避けるべき). 行列式の値を返す. (ただし, 巨大行列の場合等の場合には, この値の絶対値は 1×10^{-12} から 1×10^{12} となるように調整されていることに注意が必要. この件に関する詳細については関数 LastTeraN を参照)

[2] これは「コピー」あるいは「複写」よりも演算における「代入」という感覚によるものである. また, 他の matlib 関数群との使用感の統一のためでもある.

76 行　関数 MulMatrix を用いて行列の乗算 A×AA を行い, その結果として生成された行列をポインタ AAA に代入している. MulMatrix は, 第一引数として NULL が指定された場合, 第二・第三引数として与えられた行列の大きさによって定まる新たな行列を割り当て, そのポインタを返す. このようにして割り当てられた行列も AllocMatrix 等で割り当てられた行列と同様に扱う.

3.3 基本的な使い方

matlib の基本的な使い方について述べる. 個々の関数の詳細等については, それぞれの関数の説明を参照すること.

3.3.1 行列・ベクトルを扱うための変数宣言

matlib では行列・ベクトルは構造体へのポインタの形式で扱っている. 例えば, 行列変数 A, B, C およびベクトル変数 x, y, z を宣言するのであれば, 以下のようになる.

```
matrix  *A, *B, *C;
vector  *x, *y, *z;
```

なお, ベクトルは列数が 1 の行列と全く等価であり, 実質的には, 上記を

```
matrix  *A, *B, *C, *x, *y, *z;
```

と書くこともできる.

行列やベクトルの配列は, ポインタ変数の配列, という形式で扱うことができる. すなわち,

```
matrix  *A[3];
```

と書けば, 行列 A[0]〜A[2] からなる配列を宣言できる. ただし, 配列の要素である個々の行列を実際に利用するためには, 各々に対し, AllocMatrix 等を用いて領域を割り当てる必要がある. 配列をより効率よく利用する方法については, 3.3.5 節を参照のこと.

3.3.2 領域の割り当てと解放

matlib の行列・ベクトルは (構造体への) ポインタとして宣言されているため, 利用するためには, 領域の割り当てが必要である. 例えば,

```
A = AllocMatrix( 4, 2 );
x = AllocVector( 3 );
```

と書くことにより, A に 4×2 行列, x に 3 次元ベクトルを割り当てることができる.

演算の結果, 新たな行列やベクトルが生成される場合もある. 例えば,

```
C = MulMatrix( NULL, A, B );
```

は, 行列 A および B の乗算の結果が格納された, 適切な大きさの行列を自動的に生成し, C に代入している. これは, 次のように書くこともできる.

```
C = AllocMatrix( RSIZE(A), CSIZE(B) );
MulMatrix( C, A, B );
```

ここで, RSIZE(A) は行列 A の行数, CSIZE(B) は行列 B の列数を示すマクロである.

行列やベクトルに割り当てられた領域を解放するには関数 FreeMatrix を用いる (FreeVector もあるが, 全く等価). 例えば,

```
FreeMatrix( A );
FreeVector( x );
```

とすれば, 行列 A およびベクトル x に用いられていた領域を解放することができる. AllocMatrix や AllocVector だけでなく, MulMatrix 等によって演算の結果として生成された行列・ベクトルの場合にも同様の手続きによる領域の解放が必要である. **これらは自動ではなされない. 全て明示的に領域を解放する必要がある.** すなわち, 中間的な演算結果を格納するために関数の内部で宣言された局所行列変数の場合, 関数から出る前に必ず領域解放をしなければならない.

繰り返し呼ばれる関数やループの内部で領域割り当てを行っているプログラムの場合, プログラムの先頭で, 以下のように記述することにより, ある程度のチェックができる.

```
DispMaxAllocLevel( TRUE );
```

この場合, 領域の割り当てが行われた回数が, 解放された回数より 10 多くなる毎に下記のようなメッセージを materr(通常は標準出力に設定) に出力する.

```
*** MaxAllocLevel : 10 ***
*** MaxAllocLevel : 20 ***
```

正しいプログラムでは, ある一定のところでこのメッセージが表示されなくなるが, この表示が常に出力されている場合, 領域の割り当てと解放が対応していない可能性が高い.

3.3.3　要素等をアクセスするためのマクロ

matlib では, 行列やベクトルの要素のアクセスを記述する際に, その実体である構造体のメンバを明示的に書かない. 全て, 本節に示すマクロを介して行う. もちろん, 構造体の実体の構成を理解すれば, それらへの直接のアクセスを記述したプログラムを書くことは可能であるが, その場合プログラムの可読性は低下することになり, また, 将来の拡張等[3] の際の動作は保証されない. 以下に, よく使われるマクロの説明を示す.

MVAL, VVAL　行列・ベクトルの要素をアクセスする. 例えば

```
MVAL(A,3,4) = 3.14;
VVAL(x,3) = 2.718;
```

は, 行列 A の 3 行 4 列の要素への 3.14 の代入とベクトル x の第 3 要素への 2.718 の代入を記述している. なお, N 次元の行列やベクトルに対する MVAL および VVAL における行, 列, 要素の指定は, 数式との整合性を高めるために 1 から N となっており, 通常の C における**配列とは異なる**ことに注意が必要. **行列・ベクトルの個々の要素は倍精度実数変数と等価である.** したがって, 行列 A の 1 行 2 列への標準入力からの値の読み込み:

```
scanf( "%lf", MVAL(A,1,2) );
```

は誤りである. 正しくは,

```
scanf( "%lf", &MVAL(A,1,2) );
```

としなければならない (&に注意).

[3] 可能ならテンソルへの一般化を試みてみたい, と思ったことはある.

RSIZE, CSIZE, VSIZE　行列の行数, 列数およびベクトルの次元を表す. 例えば, 以下は, 行列 A に「行 $\times 10+$ 列」, ベクトル x に要素の番号を代入するプログラムである.

```
for( j = 1; j <= CSIZE(A); j++ ){
        for( i = 1; i <= RSIZE(A); i++ )
                MVAL(A,i,j) = (double)(i * 10 + j);
}
for( i = 1; i <= VSIZE(x); i++ )
        VVAL(x,i) = (double)i;
```

ただし, i および j は整数変数である. **これらのマクロによってアクセスされる行列の行数・列数およびベクトルの次元は書き換えてはならない**[4] .

NELEM　行列やベクトルが持っている要素の総数. バンドマトリクスでない行列に対しては, 行数 $\times$ 列数に等しい.

3.3.4 基本的な関数の紹介

頻繁に使用される関数の一覧 (名称, 機能) を示す. 書式等については, 第 4 章「関数の説明」を参照せよ.

AllocMatrix, FreeMatrix　任意の大きさの行列の領域の確保と解放.

DiagonalMatrix, UnitMatrix　対角行列, 単位行列.

FillMatrix, ZeroMatrix　行列の全要素に対するある値の設定, 零行列.

FillPart, ZeroPart　行列の部分要素にある値や零を設定.

CopyMatrix　行列のコピー (代入).

CopyPart　行列の部分要素の, (別の) 行列の部分要素へのコピー.

AllocSize, AllocSame　ある行列と同じ大きさの (初期化されない) 行列や, 全く同じ内容の行列の生成.

PrintMatrix, PrintTrans　行列をそのままあるいは転置して表示.

ReadMatrix　テキストファイルからの行列の読み込みと生成.

Transpose　転置行列.

AddMatrix, SubMatrix　行列の加算・減算.

MulScalar　スカラ倍.

MulMatrix　行列の乗算.

Trace　行列のトレース.

ScalarP　ベクトルのスカラ積 (内積).

VectorP　3 次元ベクトルのベクトル積 (外積).

L2Norm　ベクトルの L^2 ノルム.

Gauss　Gauss の消去法による線形方程式の解.

Inverse　逆行列.

[4] すなわち, RSIZE(A) や VSIZE(x) に値を代入してはならない.

3.3.5　行列・ベクトルの配列

行列およびベクトルの配列を準備する方法のひとつに, 関数 AllocMatrixArray および AllocVectorArray の利用がある. 例えば, 以下のプログラム:

```
matrix  **A;
vector  **x;

A = AllocMatrixArray( 10, 4, 2 );
x = AllocVectorArray( 20, 3 );
```

は, それぞれ 10 個の 4×2 行列 A[0]〜A[9] および 20 個の 3 次元ベクトル x[0]〜x[19] の配列を利用するためのものである. 行列・ベクトルの配列の領域の解放には

```
FreeVectorArray( x );
FreeMatrixArray( A );
```

を用いる. この場合, **配列の要素である行列やベクトルの領域を, FreeMatrix 等を用いて個別に解放することはできない.**

より高次の配列を構築するための関数も用意されているが, ここでは述べない. Alloc2dMatrixArray, Alloc3dMatrixArray 等の解説を参照せよ.

3.3.6　バンドマトリクスについて

バンドマトリクス (対角付近にのみ非零要素が存在する行列) を利用することにより, 巨大な行列を効率的に扱うことができる. バンドマトリクスの領域の割り当ては, 次のように行う.

```
matrix  *A;

A = AllocBandMatrix( 1000, 5 );
```

ここでは, 関数 AllocBandMatrix を用いて, 行列へのポインタ A に 1000×1000 の 5 重対角行列を割り当てている. AllocBandMatrix の第二引数は奇数でなければならないことに注意する必要がある. このようにして領域を割り当てられた行列 A は, 基本的に他の一般の行列と同様に取扱うことができ, マクロ MVAL 等を用いた要素へのアクセスも可能であるが, **定められたバンド領域外にアクセスした場合の結果については保証されていない**. こういったマクロによるバンド領域外へのアクセスのエラーチェックを行うには, コンパイルオプション BAND_ERROR_TEST を用いればよい (2.5 節を参照). また, バンドマトリクスの領域の解放には関数 FreeBandMatrix を以下のように用いる.

```
FreeBandMatrix( A );
```

なお, バンドマトリクスに関する以下のマクロが準備されている.

BSIZE　バンド幅を与える. 例えば, BSIZE(A) は行列 A のバンド幅である.

ISBND　ある行列がバンドマトリクスとして領域を割り当てられているかどうかをチェックする条件式. 例えば, 以下のように用いる.

```
if( ISBND(A) )
        printf( "This is a band matrix\n" );
else
        printf( "This is a non-band matrix\n" );
```

このマクロは**形式的に**その行列がバンドマトリクスとして割り当てられているかどうかをチェックするものであり, **対角領域以外に非零要素があるかどうかを調べているわけではない**ことに注意が必要である. そういった目的のためには関数 TestBandWidth を用いる.

そのバンドマトリクスに対して**許される演算であるかぎり**は matlib 関数群も対応している (はずである. 若干の制限がある場合もある). 例えば, 関数 `Gauss` によって `A` を用いて構成される線形方程式を解くことは一般的には可能であるが, 関数 `Inverse` を用いて `A` の逆行列をそのまま求めることはできない[5] . バンドマトリクスの利用による制限等の詳細については, 個々の関数についての解説を参照せよ.

[5] 関数 `Inverse` は `A` の逆行列を `A` に上書きするため. これでわからない場合はよく理由を考えること

第4章　関数の説明

以下に書いているように，ほぼ全ての関数の説明を含める予定ではあるが，量が多いため，現在のところ使用頻度の高そうなものから随時書いている状態である．matlib の関数群は (現在のところ) 全て作者個人の手によるものなので，ある程度パターンが把握できれば，ソースコードの個々の関数のコメントを参照して使うことができると思う．(コメントは全て英語だし，学生時代に書いた表現には誤りも散見されたりするが)

この章では matlib に含まれるほぼ全ての関数とそれに付随するデータ型やマクロ[1] についてソースファイル単位で解説する．なお, 節のタイトルの「〜.[hc]」という表記は, ヘッダ「〜.h」およびプログラム「〜.c」をまとめたものである[2] ．ただし, ヘッダファイルに関しては, 個々を意識することなく, matlib.h を#include すればよい.

なお, 説明に用いられている関数の引数は基本的にプログラムリスト中のものと同一としている．そのために同じような役割の引数であっても関数によって異なる変数名となっていることがある．また, 「使用例」として挙げてあるプログラム例の多くはあくまで参考であり, 中には (リストの範囲では) 宣言や初期化がなされていない変数が存在していることもあるのでこれを見てプログラムを作成する場合には注意すること.

4.1　matrix.[hc]──基本操作・領域確保・表示・入出力

matlib 関数群で利用される基本となるデータ型の定義が行われ, 行列操作における基本となる関数およびマクロが納められている.

4.1.1　bool，matrix，vector──データ型の定義

ここで定義されるデータ型 bool は実質的には int である．matlib 関数群において, 引数や関数の型が bool と宣言されている場合, その値が 4.1.8 節に示す TRUE あるいは FALSE のみに限定されることを示す[3] .

構造体 matrix は, 行列を扱うのに必要となる情報, すなわち行および列の大きさ, バンド幅, 行列の実体としての個々の要素の値を格納するものとして定義されている．また, vector は実質的には matrix と等価である．すなわち, matlib 関数群では, ベクトルは, 列の大きさが 1 である行列として扱われている．したがって, 変数宣言の際に matrix と vector の何れを用いるかは, 純粋にプログラムの可読性のみの問題である．これらの構造体は, 通常, 構造体へのポインタとして宣言することにより利用する (3.3.1 節を参照せよ)．また, matrix および vector の利用は, 所定の matlib 関数およびマクロを用いて行うべきであり, **構造体のメンバを直接アクセスすることは行うべきではない**. なお, matlib では, 行列の個々の要素は倍精度実数 (double) である.

---

[1] ソースコード中に存在していても利用することが稀であったり, 利用のために非常な注意が必要なものは書かれていない場合がある.

[2] unix 系コマンドの解説書等の正規表現に関する説明を参照のこと.

[3] このような場合, 型の定義は, 本来, 列挙型を用いて行うべきであるが, このプログラムが書かれた当時は列挙型を扱えない処理系も少なくなかったため, このようにしてある.

使用例

ブール型変数, ベクトル, 行列の変数宣言例.

```
bool    flag;           /* A TRUE or FALSE flag. */
vector  *p, *q, *r;     /* Vectors. */
matrix  *A, *M[3];      /* A matrix and a matrix array */
```

4.1.2 MVAL, VVAL, NVAL, BVAL, TVAL —行列・ベクトルの個々の要素のアクセス

これらは行列およびベクトルの個々の要素をアクセスするためのマクロである. 基本的には MVAL および VVAL を利用すればよいが, 厳しいループの中でのアクセスの場合には NVAL や BVAL を利用することにより, 若干の処理速度の向上が見込める場合もある (NVAL と BVAL は対になっている). また, バンドマトリクスを利用する場合には, エラーチェックのために TVAL を用いることもできる.

これらのマクロを用いてアクセスした結果は, 一つの倍精度実数と全く等価である (3.3.3 節を参照).

MVAL(mat,i,j)　行列変数 mat の i 行 j 列の要素へのアクセスを表す (1≤i≤RSIZE(mat), 1≤j≤CSIZE(mat)). 行および列の大きさに関するチェック等はなされていない. MVAL を用いることにより, mat がバンドマトリクスである場合にも全く同様にアクセスできるが, バンド幅の外をアクセスした場合の結果は保証されない.

VVAL(vec,i)　ベクトル変数 vec の i 番目の要素へのアクセスを表す. これは MVAL(vec,i,1) と書くことと全く等価である (前節を参照).

NVAL(mat,i,j)　行列変数 mat の i 行 j 列の要素へのアクセスを表す. MVAL とは異なり, NVAL は, mat がバンドマトリクスではないことを前提としている. 処理は若干速くなるが, mat がバンドマトリクスであった場合の結果は保証されない.

BVAL(bmat,i,j)　バンドマトリクス bmat の i 行 j 列の要素へのアクセスを表す. MVAL とは異なり, BVAL は, bmat がバンドマトリクスであることを前提としている. 処理は若干速くなるが, bmat がバンドマトリクスでなかった場合の結果は保証されない. また, バンド幅の外をアクセスした場合の結果も保証されない. キーワード BAND が定義されている場合のみ有効.

TVAL(bmat,i,j)　バンドマトリクス bmat の i 行 j 列の要素へのアクセスを表す. TVAL も bmat がバンドマトリクスであることを前提としているが, BVAL とは異なり, バンド幅の外をアクセスした場合には materr にエラーメッセージを出力する. なお, コンパイル時に, オプションを用いてキーワード BAND_ERROR_TEST を定義すれば, MVAL や BVAL によるアクセスの際にも同様のエラーチェックを行うようにすることができる (2.5 節を参照). キーワード BAND が定義されている場合のみ有効.

使用例

行列・ベクトルの要素への値の代入・表示・入力の例.

```
/* Assigment of values. */
MVAL(A,2,2) = 3.0;
VVAL(x,3) = 2.5;
/* Write elements of matrix and vector with 'printf'. */
printf( "a22 = %lf, x3 = %lf\n", MVAL(A,2,2), VVAL(x,3) );
/* Read an element of matrix with 'scanf'. */
scanf( "%lf", &MVAL(A,2,2) );
```

標準入出力関数 scanf を使った読み込み例における&に注意.

4.1.3 MPEL, VPEL, NPEL, BPEL —行列・ベクトルの個々の要素へのポインタ

これらは行列およびベクトルの個々の**要素へのポインタ**を参照するためのマクロであり, それぞれ&MVAL, &VVAL, &NVAL, &BVAL (前節参照) と等価である.

使用例

行列の要素への値の入力の例.

```
/* Read an element of matrix with 'scanf'. */
scanf( "%lf", MPEL(A,2,2) );
```

前節における scanf を使った読み込み例と比較せよ.

4.1.4 RSIZE, CSIZE, VSIZE, BSIZE, NELEM —行列やベクトルのサイズに関するマクロ

行列における行数・列数, ベクトルの次元, バンド幅および行列・ベクトルの要素数を与えるマクロである. **当該情報の参照にのみ用いること. これらのマクロを用いて値の代入等により当該情報を書き換えてはならない.**

RSIZE(mat), CSIZE(mat)　行列変数 mat の行数 (RSIZE) あるいは列数 (CSIZE) を与える. なお, matlib ではベクトルは列数が 1 の行列と等価であるので, 引数としてベクトルを指定した場合には (その変数がベクトルとして正しく扱われているならば) RSIZE(vec) はそのベクトルの次元 (=VSIZE(vec)), CSIZE(vec) は 1 となる.

VSIZE(vec)　ベクトル変数 vec の次元 (サイズ・要素数) を与える. このマクロは RSIZE と全く等価であり, どちらを用いるかは純粋に可読性の問題である. したがって, 引数として行列を指定した場合にはその行数を与える.

BSIZE(bmat)　バンドマトリクスのバンド幅を与える. 引数として与えられたのがバンドマトリクスでない場合は, その行数を与える. キーワード BAND が定義されている場合のみ有効.

NELEM(mat)　引数として与えられたのがバンドマトリクスでない場合, 要素の総数 (行数 × 列数に等しい) を返す. バンドマトリクスであった場合には, バンド幅に含まれる要素数の**目安** (列数 × バンド幅) を返す. 行列に含まれるデータに関する領域確保等に使用される.

使用例

行列変数 mat の全ての要素を二乗する.

```
for( j = 1; j <= CSIZE(mat); j++ )
        for( i = 1; i <= RSIZE(mat); i++ )
                MVAL(mat,i,j) *= MVAL(mat,i,j);
```

4.1.5 B_IN, ISBND—バンドマトリクスに関する条件式

指定された要素がバンド幅に含まれているか, あるいは指定された行列がバンドマトリクスであるか, を判別するためのマクロである.

B_IN(bmat,r,c)　引数として与えられた bmat の r 行 c 列の要素がバンド幅に含まれているかどうかを判別する. bmat がバンドマトリクスでない場合には常に真である. ただし, r および c が正であるか, あるいは行数や列数の制約を満たしているか等はチェックしないので注意が必要.

ISBND(mat)　引数 mat がバンドマトリクスであるかどうかを判別する.

使用例

行列変数 mat の全ての要素を二乗する (バンドマトリクスの可能性を考慮).

```
for( j = 1; j <= CSIZE(mat); j++ ){
        for( i = 1; i <= RSIZE(mat); i++ ){
                if( B_IN(mat,i,j) )
                        MVAL(mat,i,j) *= MVAL(mat,i,j);
        }
}
```

4.1.6　UPR, LWR—大きい値・小さい値

二つの値の大小関係を比較し, その大きいほうの値 (UPR), 小さいほうの値 (LWR) を与える. 関数ではなくマクロであることから, **比較する値の型 (int か double か等) に依存することなく利用可能**である[4] .

UPR(x, y)　引数 x と y を比較し, 大きいほうの値を返す.

LWR(x, y)　引数 x と y を比較し, 小さいほうの値を返す.

使用例

二つの整数変数 a, b の大きいほうの回数のループの記述例.

```
for( i = 0; i < UPR(a, b); i++ ){
        ...
}
```

4.1.7　LRI, URI, LCI, UCI, LBI, UBI—行および列の下限および上限

バンドマトリクスを含め, 行列において実際に要素が存在する行および列の上限および下限を与える. バンドマトリクスを扱わない場合には下限は 1, 上限は RSIZE および CSIZE に等しいので特にこれらのマクロを利用する必要はない.

LRI(mat, i), URI(mat, i)　行列 mat の第 i 列の行の下限および上限.

LCI(mat, i), UCI(mat, i)　行列 mat の第 i 行の列の下限および上限.

LBI(bmat, i), UBI(bmat, i)　バンドマトリクス bmat の第 i 行/列における列/行の下限および上限. このマクロは bmat がバンドマトリクスであることを前提としている.

[4] このことが理解できない場合には, 例えば「プログラミング言語 C・第 2 版」の 4.11.2 節「マクロの置換」等を復習するとよい.

使用例

行列 mat の n 行目を 2 倍にし, その後 n 列目を 2 倍にする. これはバンドマトリクスにも対応した記述である. 同様の (バンドマトリクスに対応した) プログラムは前述の B_IN を用いた方法でもできるが, 行列の規模が大きい場合 (特にバンド幅が狭い場合) にはこちら記述のほうが処理を軽減できる.

```
for( j = LCI(mat, n); j <= UCI(mat, n); j++ )
        MVAL(mat,n,j) *= 2.0;
for( i = LRI(mat, n); i <= URI(mat, n); i++ )
        MVAL(mat,i,n) *= 2.0;
```

4.1.8 TRUE, FALSE, iX, iY, iZ—頻繁に使われる整数定数

特定の意図で利用される整数定数を表す.

TRUE, FALSE　主として bool 型[5] で宣言された変数や関数の値の表記に用いられる. TRUE は「真」, FALSE は「偽」に対応し, 値としてそれぞれ 1 および 0 が設定されている[6] .

iX, iY, iZ　二次元および三次元の座標空間等を扱う際に利用される. iX, iY および iZ には値としてそれぞれ 1, 2 および 3 が設定されている.

使用例

平面位置ベクトル $\boldsymbol{p} = [p_x, p_y]^T = [1, 2]^T$ に対応する二次元ベクトル変数 p を準備する.

```
vector  *p;

p = AllocVector( 2 );
VVAL(p,iX) = 1.0;
VVAL(p,iY) = 2.0;
```

4.1.9 EPSILON, INFINIT—実数値の条件判断等に利用される定数

整数値を用いた計算の場合と異なり, 実数値による計算の場合には「ある値と等しいかどうか」という条件判断はほとんどの場合不適切である[7] . したがってある実数変数が別の実数変数や実数定数と等しいかどうかを判断する場合にはマージンをとって比較することが必要となる. 例えば, ある行列が正則かどうかは「行列式の値が 0 と等しいかどうか」で判定することになっているが, 行列式の値がちょうど 0 になるのは整数値行列を整数演算した場合くらいで, 実際には「その値がどれだけ 0 に近ければ, 0 であるとみなすか」という判定をすることになる.

また, ある種の数値計算において初期値に無限大の値を設定したい場合がある. このような場合, 適当な巨大な (と見なせる) 値を初期値として用いることができる.

EPSILON　matlib の関数群においては, 特に明示されていない場合に matrix.h 内で EPSILON として定義されている値 (1×10^{-12} としてある) を用いて実数値の等号判定を行う. 主として, この値は 0 か否かの判定に用いられる.

INFINIT　数値計算の初期値として適当な「巨大な値」を設定する場合などに使用する. matrix.h 内では 1×10^{12} として定義されている. これはほとんど用いられていない.

[5] matlib では bool の実体は int であることに注意.

[6] ここで, TRUE が 1 であることに大きな意味はない (すなわち他の値であってもほぼ問題ない**はず**である) が, FALSE は 0 である必要がある. このことが理解できない場合には, 例えば「プログラミング言語 C・第 2 版」の第 3 章「制御の流れ」等を復習するとよい.

[7] 整数型の 1 を 10 回足せば 10 になるが, 実数型の 1.0 を 10 回足しても 10.0 ちょうどにはまずならない. 10.0 よりわずかに小さいか大きい値となる (処理系等に依存). したがって, 整数変数の場合と同様の for ループを実数変数で構成しても意図したとおりに動作しない場合がある.

4.1.10　Kdelta—クロネッカのデルタ

いわゆるクロネッカのデルタ (Kronecker delta) を与えるマクロである.

`Kdelta(i, j)` 引数の `i` と `j` は整数を想定している (実数でも構わないが, 多くの場合, 意図した結果とはならない). `i` と `j` が等しい場合には 1.0, 異なる場合には 0.0 を返す. 実数演算の係数として用いることを想定したものであるため, **返す値が実数型となっている**ことに注意.

使用例

対角要素のみに行番号に対応する値を持つ行列変数 `A` を準備する.

```
matrix  *A;
int     i, j;

A = AllocMatrix( 9, 9 );
for( i = 1; i <= 9; i++ )
        for( j = 1; j <= 9; j++ )
                MVAL(A,i,j) = Kdelta(i,j) * (double)i;
```

4.1.11　InitMaterr—エラー出力の初期化

matlib 関数群の全てのエラー出力は `materr` に対して行われる. `materr` は `matrix.h` 内で標準出力 (`stdout`) として定義されており, 例えばこれを標準エラー出力 (`stderr`) に定義しなおすことも可能である.

コンパイルオプション等で `MATERR_VAR` が定義されている場合には, `materr` はファイルポインタ型の変数となり, 状況に応じてエラー出力先を変更することが可能となるが, 初期化が必要となる. `InitMaterr` は, `materr` が変数である場合にこれを標準出力 (`stdout`) に初期化する. `materr` が変数でない場合 (`MATERR_VAR` が定義されていない場合) には何もしない.

通常の使用では使わなくても特に問題はない.

`void InitMaterr()` コンパイルオプション等で `MATERR_VAR` が定義されている場合に matlib 関数群のエラー出力 `materr` を標準出力に初期化する.

4.1.12　MatStatus, MatError—エラー状態のチェック

matlib 関数群で想定している検証可能なエラー (例えば, 正方行列にしか許されない演算処理を長方形行列に対して行おうとした等) が発生していないかどうかをチェックする. 基本的には `MatError` は `MatStatus` の否定だが, エラーが発生している場合のふるまいが異なる.

`bool MatStatus()` 過去の matlib 関数群の処理が正常に行われていれば `TRUE`, そうでなければ `FALSE` を返す[8] .

`bool MatError()` 過去の matlib 関数群の処理でエラーが発生しているときに `TRUE`, そうでなければ `FALSE` を返す. ただし, (`NO_STDIO` が定義されていない場合には) エラーが発生している状態で 10 回この関数が呼び出されたらプログラムを停止する点が `MatStatus` と異なる. これは, エラーが発生している状態ではある関数から戻る際にはその関数名を出力するようにし, 適当な所でプログラムを停止することにより, エラーを発生している関数がどこから呼び出されたかを把握しやすくするためである.

[8] ここでいう「正常」とは計算結果の妥当性に関するものではなく, 不能な演算 (サイズの異なる行列の加算等) の類が行われていないかどうか, である.

使用例

とある関数の最後を以下のようにすることにより, その関数内で呼び出した matlib 関数でエラーが発生したことが把握しやすくなる.

```
        ...

        if( MatError() ){
                fprintf( materr, "in 'ThisFunction()'.\n" );
                return
        }
}
```

ただし, このような処理はあくまで補助的なものであり, あまり役に立たない場合も少なくない.

4.1.13 MatErrorSet, MatErrorReset—エラー状態の設定と解除

matlib 関数群のエラー状態を意図的に操作する.

`void MatErrorSet()` matlib 関数群にエラー状態を設定する. プログラムのエラー状態の設定や参照を matlib 関数群と合わせて統一的に行う場合等に使用する. 数値計算を行う場合, エラーが発生したかどうかはプログラム全体の問題であるため, このような状態を示すものは単一でよく, 時々利用される.

`void MatErrorReset()` matlib 関数群のエラー状態を解除する. 数値計算を行うプログラムの場合, エラーが起きた時点で計算結果が信用できないものとなってしまうことが多いので基本的にはあまり使う必要はない. ただ, そのエラーが想定の範囲内であるときにそれをリセットして計算を続行したりするような場合に使う.

4.1.14 SetBandErrorLevel—バンド外要素を 0 と見なす大きさ

対角部のみに非ゼロ要素が存在すると考えられる通常マトリクスを `TestBandWidth` や `CopyMatrix` 等を用いてバンドマトリクスに変換する際などにどの程度の値であれば 0 と見なしてバンド外で問題ないとするかを指定する.

```
void SetBandErrorLevel( double level )
```

引数 `level` 新たに指定する 0 と見なす大きさ

この関数が用いられない場合, デフォルトの値は `matrix.h` で定義されている `EPSILON` である.

4.1.15 DispMaxAllocLevel—行列に割り当てられたヒープ領域数の表示

matlib ではマトリクスやベクトルに割り当てるメモリ領域を `AllocMatrix` 等を用いてヒープ領域から確保し, 不要となった時点で `FreeMatrix` 等を用いて解放することとなっている. したがって, 例えばループの中で領域の確保と解放のアンバランスがある場合には, ヒープ領域を食いつぶしてプログラムがハングアップし, 脆弱なシステムの場合, まれにシステムダウンを起こす可能性がある. このようなことが起こる可能性があるかどうかをチェックする際に用いる.

```
int DispMaxAllocLevel( bool set )
```

引数 set TRUE 領域確保数を表示
FALSE 領域確保数を表示しない (デフォルト)

返す値 現在の領域確保数 (最大値ではないことに注意)

領域確保数を表示する場合には, 領域の割り当てが行われた回数が解放された回数より, 10 多くなる毎に下記のようなメッセージを materr(通常は標準出力に設定) に出力する.

```
*** MaxAllocLevel : 10 ***
*** MaxAllocLevel : 20 ***
```

正しいプログラムでは, ある一定のところでこのメッセージが表示されなくなるが, この表示が常に出力されている場合, 領域の割り当てと解放が対応していない可能性が高い.

4.1.16 AllocMatrix, AllocVector, AllocBandMatrix, AllocSize, AllocSame, FreeMatrix, FreeVector —行列・ベクトルの領域の割り当てと解放

指定された大きさの行列やベクトルの領域を割り当て, matrix 構造体に必要な値を設定し, そのポインタを返す. また, そのようにして確保された領域を解放する.

```
matrix *AllocMatrix( int nr, int nc )
```

引数 nr, nc 行および列のサイズ

返す値 領域を割り当てた matrix 構造体へのポインタ. 領域不足等により割り当てに失敗した場合は NULL を返す.

```
vector *AllocVector( int n )
```

引数 n ベクトルのサイズ

返す値 領域を割り当てた vector 構造体へのポインタ. 領域不足等により割り当てに失敗した場合は NULL を返す.

ここで, vector 構造体は matrix 構造体に別の名前を付けただけのもので, 両者は全く等価であり, プログラム上で「これはベクトル」ということを明確にする以上の意味はない. 実際, この関数は AllocMatrix を用いて n×1 の行列を準備するマクロとして matrix.h 内で定義されている.

```
matrix *AllocBandMatrix( int nrc, int bsize )
```

引数 nrc 行列のサイズ (正方行列なので一つでよい)
bsize バンド幅. **奇数でなければならない**

返す値 領域を割り当てた matrix 構造体へのポインタ. 領域不足等により割り当てに失敗した場合は NULL を返す.

バンドマトリクスとして正方行列を準備し, 領域を割り当てる.

```
matrix *AllocSize( matrix *mat )
```

引数 mat サイズを参照する行列

返す値 領域を割り当てた matrix 構造体へのポインタ. 領域不足等により割り当てに失敗した場合は NULL を返す.

指定された行列を参照し, 同じサイズの行列を準備する. このとき mat がバンドマトリクスであれば準備される行列もバンドマトリクスとなる.

```
matrix *AllocSame( matrix *mat )
```

引数 mat 参照する行列

返す値 領域を割り当て, 値を設定した matrix 構造体へのポインタ. 領域不足等により割り当てに失敗した場合は NULL を返す.

指定された行列を参照し, 同一サイズ・同一内容の行列を準備する. このとき mat がバンドマトリクスであれば準備される行列もバンドマトリクスとなる.

```
void FreeMatrix( matrix *mat )
```

引数 mat 領域を解放する行列.

指定された行列のために確保されていた領域を解放する. ただし, mat が NULL である場合には何もしない.

```
void FreeVector( vector *v )
```

引数 v 領域を解放するベクトル.

指定されたベクトルのために確保されていた領域を解放する. ただし, v が NULL である場合には何もしない. この関数は FreeMatrix に別の名前を付けただけのもので, 両者は全くの等価である.

4.1.17 ChangeMatrixSize, ChangeRowSize, ChangeColumnSize, ChangeVectorSize—行列・ベクトルのサイズの変更

4.1.18 AllocNdArray, FreeNdArray —N 次元配列の動的領域割り当てと解放

4.1.19 AllocNdMatrixArray, FreeNdMatrixArray —行列を要素とする N 次元配列

4.1.20 AllocMatrixArray, AllocVectorArray, FreeMatrixArray, FreeVectorArray —行列・ベクトルを要素とする配列

行列やベクトルを要素とする配列を用いる最も単純な方法は, matrix や vector のポインタの配列を宣言し, AllocMatrix や AllocVector を用いて個々の要素に個別に領域を割り当てることである. しかしながら, 配列の要素が多数である場合には, システムコールである malloc を何度も呼び出すことになり, (処理系によっては) メモリの利用効率や時間といった点が問題になる場合がある. また, この場合には配列の最大の大きさが宣言によって決められてしまうというある意味で不自由な側面もある.

これらの関数は, システムコールである malloc の呼び出し回数を低減し, 行列やベクトルの配列の動的な割り当てを可能とするものである.

```
matrix **AllocMatrixArray( int nm, int nr, int nc )
```

引数 nm 配列の大きさ (行列の個数)
nr, nc 配列要素である行列の行および列のサイズ

返す値 行列の配列へのポインタ (matrix 構造体へのポインタのポインタ). 領域不足等により割り当てに失敗した場合は NULL を返す.

行列の配列を指定された数, サイズで準備する. 配列の要素である個々の行列の領域確保も行われる.

```
vector **AllocVectorArray( int nv, int n )
```

引数 nv 配列の大きさ (ベクトルの個数)
n 配列要素であるベクトルのサイズ

返す値 ベクトルの配列へのポインタ (vector 構造体へのポインタのポインタ). 領域不足等により割り当てに失敗した場合は NULL を返す.

ベクトルの配列を指定された数, サイズで準備する. 配列の要素である個々のベクトルの領域確保も行われる.

```
void FreeMatrixArray( matrix **m )
```

引数 m 領域を解放する行列配列へのポインタ

AllocMatrixArray で割り当てられた行列配列の領域を解放する.

```
void FreeVectorArray( vector **v )
```

引数 v 領域を解放するベクトル配列へのポインタ

AllocVectorArray で割り当てられた行列配列の領域を解放する. なお, AllocVector のところで述べたように, vector と matrix は, 処理上は全く等価であるため, ベクトル配列を FreeMatrixArray を用いて解放することやその逆も処理上は全く問題はない.

【注意】 AllocMatrixArray や AllocVectorArray を用いて領域確保した配列の個々の要素である行列やベクトルを FreeMatrix や FreeVector を用いて領域解放することはできない.

使用例

任意の自由度 (dofs) の直列リンクロボットアームに対する 4×4 同次座標変換行列を自由度の数だけ準備して単位行列として初期化する.

```
matrix  **T;
int     n;
...

T = AllocMatrixArray( dofs, 4, 4 );
for( n = 0; n < dofs; n++ )
        UnitMatrix( T[n] );
...
```

4.1.21 Alloc1dMatrixArray, Free1dMatrixArray, Alloc2dMatrixArray, Free2dMatrixArray, Alloc3dMatrixArray, Free3dMatrixArray —行列を要素とする 1 次元, 2 次元, 3 次元の配列

4.1.22 AliasMatrix, AliasVector, AliasPartVector —行列もしくはその一部を別の行列やベクトルとする

行列やベクトルの一部を, あたかも独立の行列やベクトルのように扱う演算を行うことを考える. この場合, 該当部分を別に準備した相当サイズの行列やベクトルに (CopyPart 等を用いて) コピーして演算を行

い, (その値が変化するような演算であった場合には) 元の行列に再びコピーするという手法が考えられる. これは確かに単純な手法ではあるが, コピーの手間等もかかり, 処理時間の点で不利であるし, やや面倒な側面がある.

このような場合, 行列やベクトルの一部に別の名前を付け, 内容を共有する形であたかも独立した別の行列やベクトルとして扱えれば, それらに対する修正が直ちに元の行列の該当部分にも反映され, 便利である. これらの関数はそういった処理を取扱う. ただし, これらの処理は matrix の内部表現に依存している関係で, 行列の連続する列か, ベクトルの連続する一部に限られる. 行列の一部の行に対してこの処理を行うことはできない.

```
matrix *AliasMatrix( matrix *amat, matrix *smat, int c, int n )
```

引数 amat 別の名前のための行列.
smat 元の行列.
c, n 別名を付ける列とその数の指定.

返す値 通常は amat を返す. 列の指定のエラー等があれば NULL を返す.

この関数は行列 smat の c 列目から始まる n 列を amat として独立の行列であるかのように扱うことを可能とする. ただし, amat は AllocMatrix(0, 0) を用いて初期化されている必要があり, 使用後は FreeMatrix を用いて領域を解放する必要がある. また, c が 0 である場合は n は参照されず, smat 全体が amat という別名の対象となる[9] . なお, smat がバンドマトリクスである場合には c は 0 でなければならない.

```
vector *AliasVector( vector *vec, matrix *mat, int c )
```

引数 vec 別の名前のためのベクトル.
mat 元の行列.
c 別名を付けてベクトルとする列.

返す値 通常は vec を返す. 列の指定のエラー等があれば NULL を返す.

この関数は行列 mat の c 列目を vec として独立のベクトルであるかのように扱うことを可能とする. ただし, vec は AllocVector(0)(AllocMatrix(0, 0) と等価) を用いて初期化されている必要があり, 使用後は FreeVector を用いて領域を解放する必要がある. この関数はバンドマトリクスである mat には使用できない.

```
vector *AliasPartVector( vector *vec, matrix *mat, int r, int c, int size )
```

引数 vec 別の名前のためのベクトル.
mat 元の行列.
r, c, size 別名を付けてベクトルとする行, 列, サイズ.

返す値 通常は vec を返す. 列の指定のエラー等があれば NULL を返す.

この関数は行列 mat の r 行 c 列から size 行を vec として独立のベクトルであるかのように扱うことを可能とする. ただし, vec は AllocVector(0) を用いて初期化されている必要があり, 使用後は FreeVector を用いて領域を解放する必要がある. 複数の列に及ぶ指定はできない. また, この関数はバンドマトリクスである mat には使用できない.

これらの関数において別の名前を担う行列やベクトルは, FreeMatrix や FreeVector を用いて解放することなく, 次々に異なる行列やその異なる部分を取扱うことができる. 逆に, 別の名前を付けた行列やベクトルの個々の要素の値は元の行列の存在に完全に依存している. したがって, (別名のための行列やベクトルを解放することなく) 元の行列を解放する場合には注意が必要である.

[9] こういう使い方をすることはないと思うが….

使用例

三次元構造解析計算における全節点位置ベクトルを $\boldsymbol{X} = [\boldsymbol{x}_1^T, \cdots, \boldsymbol{x}_N^T]^T$ とし, これに対応するベクトル変数 X を準備する. このときに個々の節点位置 $\boldsymbol{x}_i$ に対する処理を該当部分に Xi という別名を付けて行う.

```
matrix  *X, *Xi;
int     i;

X = AllocVector( 3 * N );
Xi = AllocVector( 0 );
for( i = 1; i <= N; i++ ){
        AliasPartVector( Xi, X, (i-1)*3+1, 1, 3 );
        /* Now, Xi can be dealt with an independent 3D vector. */
        ...
}
FreeVector( Xi );
FreeVector( X );
```

この処理は ConvertToMatrix と AliasVector を用いて下記のように書くこともできる.

```
matrix  *X, *Xi;
int     i;

X = AllocVector( 3 * N );
Xi = AllocVector( 0 );
ConvertToMatrix( X, 3 );
for( i = 1; i <= N; i++ ){
        AliasVector( Xi, X, i );
        /* Now, Xi can be dealt with an independent 3D vector. */
        ...
}
ConvertToVector( X );
FreeVector( Xi );
FreeVector( X );
```

4.1.23　DiagonalMatrix, UnitMatrix—対角行列, 単位行列の設定

既に領域が確保された正方行列を, 特定の値を対角要素に持つ行列, あるいは単位行列として初期化する.

```
matrix *DiagonalMatrix( matrix *mat, double d )
```

引数　mat　初期化する正方行列.
　　d　対角要素を初期化する値.

返す値　通常は mat を返す. 何らかのエラーがあれば NULL を返す.

対角要素を d, 他の要素は 0 として行列 mat を初期化する.

```
matrix *UnitMatrix( matrix *mat )
```

引数　mat　初期化する正方行列.

返す値　通常は mat を返す. 何らかのエラーがあれば NULL を返す.

対角要素を 1, 他の要素は 0 として行列 mat を初期化する.

使用例

$N \times N$ 単位行列を変数 E として準備する.

```
matrix  *E;

E = UnitMatrix( AllocMatrix( N, N ) );
...
```

4.1.24 FillMatrix, FillPart —行列もしくはその一部を特定の値で満たす

特定の値を行列の全体もしくは部分領域 (矩形領域) に書き込む.

```
matrix *FillMatrix( matrix *mat, double d )
```

引数 mat 既に領域が確保された行列.
d 初期化する値.

返す値 通常は mat を返す. 何らかのエラーがあれば NULL を返す.

行列 mat の全ての要素を d で初期化する. mat がバンドマトリクスである場合, **バンドの領域内の**要素が d で初期化される.

```
matrix *FillPart( matrix *mat, double d, int i, int j, int m, int n )
```

引数 mat 既に領域が確保された行列.
d 初期化する値.
i, j 初期化する領域の左上隅の行と列.
m, n 初期化する領域の大きさ.

返す値 通常は mat を返す. 何らかのエラーがあれば NULL を返す.

行列 mat の i 行 j 列から i+m-1 行 j+n-1 列の大きさ m 行 n 列の矩形領域にある要素を d で初期化する. mat がバンドマトリクスで, d の絶対値が SetBandErrorLevel で設定したレベル (デフォルトは EPSILON) よりも大きく, バンドの領域外が初期化の対象となった場合にはエラーとなる.

4.1.25 ZeroMatrix, ZeroPart, ZeroRow, ZeroColumn —行列の全体もしくは指定された部分を 0 で満たす

行列の全体あるいは指定された領域を全て 0 とする. 演算の初期化で頻繁に用いられる.

```
matrix *ZeroMatrix( matrix *mat )
```

引数 mat 既に領域が確保された行列.

返す値 通常は mat を返す. 何らかのエラーがあれば NULL を返す.

行列 mat の全ての要素を 0 に初期化する.

```
matrix *ZeroPart( matrix *mat, int i, int j, int m, int n )
```

引数 mat 既に領域が確保された行列.
i, j 初期化する領域の左上隅の行と列.
m, n 初期化する領域の大きさ.

返す値 通常は mat を返す. 何らかのエラーがあれば NULL を返す.

行列 mat の i 行 j 列から i+m-1 行 j+n-1 列の大きさ m 行 n 列の矩形領域にある要素を 0 に初期化する.

```
matrix *ZeroRow( matrix *mat, int r )
```

引数 mat 既に領域が確保された行列.
r 初期化する行.

返す値 通常は mat を返す. 何らかのエラーがあれば NULL を返す.

行列 mat の r 行を 0 に初期化する.

```
matrix *ZeroColumn( matrix *mat, int c )
```

引数 mat 既に領域が確保された行列.
c 初期化する列.

返す値 通常は mat を返す. 何らかのエラーがあれば NULL を返す.

行列 mat の c 列を 0 に初期化する.

4.1.26 TestZeroRow, TestZeroColumn —特定の行もしくは列が全て 0 かどうかのテスト

4.1.27 TestBandWidth—行列のバンド幅を調べる

ある行列が対角付近にしか非ゼロ要素を持たない場合, (その行列に適用しようとする演算処理の種類にもよるが) バンドマトリクスとして取扱えば, 記憶領域や計算の効率の点で有利になる. この関数は, 与えられた行列をバンドマトリクスとして扱う場合のバンド幅を調べる.

```
int TestBandWidth( matrix *mat )
```

引数 mat バンド幅を調べる正方行列.

返す値 バンド幅. 何らかのエラーがあれば -1 を返す.

バンド幅の判定のための閾値 (どの程度小さい値なら 0 と見なすか) は SetBandErrorLevel で設定した値 (デフォルトは EPSILON) である. 行列 mat は既にバンドマトリクスであってもかまわない.

使用例

ある正方行列 mat をチェックしてバンド幅が行列のサイズの 1/10 より小さければバンドマトリクスに変換する.

```
matrix  *mat, *tmp;
int     w;
...
w = TestBand( mat );
if( 10 * w < RSIZE(mat) ){
        tmp = AllocSame( mat );
        FreeMatrix( mat );
        mat = AllocBandMatrix( RSIZE(tmp), w );
        CopyMatrix( mat, tmp );
        FreeMatrix( tmp );
}
...
```

ただし, このような手法による変換は mat が AllocMatrixArray 等を用いて領域確保を行われた配列の要素である行列変数の場合は用いることができない. また, mat の全体あるいは一部に対して AliasVector 等で別名を付けている場合にも行うべきではない.

4.1.28 CopyMatrix, CopyPart, CopyPartVec —行列・ベクトルもしくはその一部のコピー

行列やベクトルの全部もしくは一部を別の行列やベクトルにコピーする. 数学的な代入の意味から, **引数の順序が unix の** cp **や MS-DOS の** copy **といったいわゆるコピー命令とは逆になっている**ことに注意が必要.

```
matrix *CopyMatrix( matrix *dmat, matrix *smat )
```

引数 dmat コピー先行列.
smat コピー元行列.

返す値 通常は dmat を返す. 何らかのエラーがあれば NULL を返す.

smat の値を dmat にコピーする. dmat の行や列のサイズが smat よりも大きい場合, エラーとはならず, dmat の smat に該当する部分のみが書き換えられ, 他の部分は変化しない (ゼロクリアされたりしない).

```
matrix *CopyPart( matrix *dmat, int dr, int dc, matrix *smat, int sr, int sc,
                  int rsiz, int csiz )
```

引数

dmat	コピー先行列.
dr, dc	コピーを貼り付ける場所の左上隅の行と列.
smat	コピー元行列.
sr, sc	コピーする部分の左上隅の行と列.
rsize, csiz	コピーする部分の行および列のサイズ.

返す値 通常は dmat を返す. 何らかのエラーがあれば NULL を返す.

smat の指定された部分 (sr 行 sc 列から sr+rsiz-1 行 sc+csiz-1 列の大きさ rsiz 行 csiz 列の矩形領域) を dmat の dr 行 dc 列を左上隅としてコピーする.

```
vector *CopyPartVec( vector *d, int i, vector *s, int j, int n )
```

引数 d コピー先ベクトル.
i コピーを貼り付ける場所.
s コピー元ベクトル.
j コピーする部分の先頭位置.
n コピーする部分のサイズ.

返す値 通常は d を返す. 何らかのエラーがあれば NULL を返す.

ベクトル s の指定された部分 (i 番目から i+n-1 番目の n 個の要素) を d の j 番目の要素を先頭としてコピーする. なお, この関数は CopyPart のマクロとして実現されており, d や s が実際にベクトル (列数 1) かどうかといったチェックは行われていない.

これらの関数のコピー先がバンドマトリクスであり, バンド幅の外もコピー先の対象となっている場合, SetBandErrorLevel で設定した値 (デフォルトは EPSILON) より絶対値の大きな値が対応するコピー元にある場合にはエラーとなる.

4.1.29 CopyMatrixWithMask—行列のコピー (行および列のマスク付き)

4.1.30 ExRow, ExColumn—行または列の置換

4.1.31 AbsMax, MaxVal, MinVal—最大・最小の要素の値を調べる

行列やベクトルの要素の最大値や最小値を調べる. 要素の最大の絶対値で正規化する場合などに用いる.

```
double AbsMax( matrix *mat )
```

引数 mat 要素の値を調べるマトリクス.

返す値 行列 mat に含まれる全ての要素で最大の絶対値を持つものの値.

```
double MaxVal( matrix *mat )
```

引数 mat 要素の値を調べるマトリクス.

返す値 行列 mat に含まれる全ての要素で最大の値を持つものの値.

mat がバンドマトリクスの場合, 対象となるのはバンド幅の中にある要素のみ.

```
double MinVal( matrix *mat )
```

引数 mat 要素の値を調べるマトリクス.

返す値 行列 mat に含まれる全ての要素で最小の値を持つものの値.

mat がバンドマトリクスの場合, 対象となるのはバンド幅の中にある要素のみ.

4.1.32 ConvertToVector, ConvertToMatrix —行列のベクトル化・ベクトルの行列化

行列の各列を一次元状に並べてベクトルとする. あるいは逆にベクトルを適当な大きさの小ベクトルの集まりであると見なして, これらの小ベクトルを列とする行列とする.

```
vector *ConvertToVector( matrix *mat )
```

引数 mat ベクトル化する行列.

返す値 通常はベクトルに変換された mat を返す. 何らかのエラーがあれば NULL を返す.

行列変数 mat($m \times n$ 行列とする) を $mn \times 1$ 行列 (すなわち要素数 mn のベクトル) に変換する. 具体的には下記のような変換を行う.

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}_{(m\times n)} \Rightarrow \boldsymbol{A} = [a_{11}, a_{21}, \cdots, a_{m1}, a_{12}, a_{22}, \cdots, a_{m2}, \cdots, a_{1n}, a_{2n}, \cdots, a_{mn}]^T$$

```
matrix *ConvertToMatrix( vector *vm, int r )
```

引数 vm 行列化するベクトル.

返す値 通常は行列に変換された vm を返す. 何らかのエラー (vm のサイズが r で割り切れない等) があれば NULL を返す.

ベクトル変数 vm(rc 個の要素を持つとする) を $r \times c$ 行列に変換する. 具体的には下記のような変換を行う.

$$\boldsymbol{A} = [a_{11}, a_{21}, \cdots, a_{r1}, a_{12}, a_{22}, \cdots, a_{r2}, \cdots, a_{1c}, a_{2c}, \cdots, a_{rc}]^T \;\Rightarrow\; \boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1c} \\ a_{21} & a_{22} & \cdots & a_{2c} \\ \vdots & \vdots & \ddots & \vdots \\ a_{r1} & a_{r2} & \cdots & a_{rc} \end{bmatrix}_{(r\times c)}$$

これらの関数はバンドマトリクスには適用できない.

使用例

AliasVector の使用例 (ConvertToMatrix と共に用いた例) を参照のこと.

4.1.33 PutMatrix, GetMatrix, PutMatrixBody, GetMatrixBody —行列のファイル入出力

行列やベクトルのデータのファイルへの読み書きを内部表現 (バイナリイメージ) のまま行う. PutMatrix および GetMatrix は行列やベクトルの数値データに加えてサイズに関する情報も一緒に扱うが, PutMatrixBody および GetMatrixBody では行列やベクトルの中身の数値データのみを扱う点が異なる. これらの関数はバンドマトリクスも使用可能だが, 数値データの書き込み順序等に (バンドでないマトリクスの場合との間の) 互換性は特にない.

```
matrix *PutMatrix( matrix *mat, FILE *fp )
```

引数 mat ファイルに書き込む行列.
fp ファイルポインタ.

返す値 通常は mat を返す. 何らかのエラーがあれば NULL を返す.

行列 mat のデータをサイズに関する情報と共にファイル fp に書き込む.

```
matrix *GetMatrix( FILE *fp )
```

引数 fp 行列を読み出すファイルポインタ.

返す値 読み出されたデータを書き込んだ, 新たに領域を確保した行列へのポインタを返す. 何らかのエラー (ファイルのフォーマット等) があれば NULL を返す.

PutMatrix を用いて書き込まれた行列をファイル fp から読み出す. 行列の領域は自動的に確保されてポインタが返される. 生成される行列がバンドマトリクスか否かは, ファイルに書き込まれた行列に依存する.

```
matrix *PutMatrixBody( matrix *mat, FILE *fp )
```

引数 mat ファイルに書き込む行列.
fp ファイルポインタ.

返す値 通常は mat を返す. 何らかのエラーがあれば NULL を返す.

行列 mat の要素の数値データのみをファイル fp に書き込む. すなわち, $m \times n$ の (バンドマトリクスでない) 通常行列であれば, mn 個の倍精度実数のバイナリイメージがファイルに書き込まれる.

```
matrix *GetMatrixBody( matrix *mat, FILE *fp )
```

引数 mat　ファイルから読み出したデータを格納する行列.
fp　ファイルポインタ.

返す値 通常は mat を返す. 何らかのエラーがあれば NULL を返す.

ファイル fp から行列 mat に数値データを読み込む. 行列変数 mat には AllocMatrix 等を用いて**事前に適切な領域が割り当てられていなければならない**. 行や列のサイズに関するチェックは行われない. PutMatrixBody で書き込んだものと (バンド幅も含めて) 同じサイズの行列変数を準備する.

【注意】 これらの関数でアクセスされるファイルは必ず**バイナリモード**でオープンされている必要がある.

使用例

サイズの情報を伴うファイルへの書き込み.

```
matrix  *A, *B;
FILE    *fp;
...
if( (fp = fopen( "data", "wb" )) == NULL ){
        ... /* Error process */
}
PutMatrix( A, fp ); /* Write matrix A */
PutMatrix( B, fp ); /* Write matrix B */
fclose( fp );
...
```

上記で書き込んだ行列の読み込み.

```
matrix  *A, *B;
FILE    *fp;
...
if( (fp = fopen( "data", "rb" )) == NULL ){
        ... /* Error process */
}
A = GetMatrix( fp ); /* Read matrix A */
B = GetMatrix( fp ); /* Read matrix B */
fclose( fp );
...
```

この場合, 読み込みに用いる行列変数に事前に領域等を割り当てる必要がない (もし割り当てられていた場合にはいったん解放する必要がある) ことに注意.

行列の中身の数値データのみのファイルへの書き込み. 書式等は上記の PutMatrix の場合と同様である.

```
matrix  *A, *B;
FILE    *fp;

A = AllocMatrix( 100, 100 );
B = AllocSize( A );
...
if( (fp = fopen( "data", "wb" )) == NULL ){
        ... /* Error process */
}
PutMatrixBody( A, fp ); /* Write matrix A without size */
PutMatrixBody( B, fp ); /* Write matrix B without size */
fclose( fp );
...
```

上記で書き込んだ行列を読み込んで処理を行う.

```
matrix  *C;
FILE    *fp;

C = AllocMatrix( 100, 100 );
if( (fp = fopen( "data", "rb" )) == NULL ){
        ... /* Error process */
}
GetMatrixBody( C, fp ); /* Read matrix 'A' into C */
... /* Process with matrix in C. */
GetMatrixBody( C, fp ); /* Read matrix 'B' into C */
... /* Process with matrix in C. */
fclose( fp );
...
```

この場合, 読み込みに用いる行列変数には予め領域が割り当てられている必要がある. 逆に言えば, (サイズ等の点で矛盾が生じなければ) このプログラム例のようにいったん領域を割り当てた行列変数に繰り返し読み込んで処理を行うこともできる.

4.1.34 fprintfMatrix, PrintMatrix, PrintTrans, fPrintMatrix, fPrintTrans, PrintFMatrix, PrintFTrans, fPrintFMatrix, fPrintFTrans —行列の表示

行列やベクトルの表示あるいはテキストとしてのファイルへの出力を行う[10] .

```
matrix *fprintfMatrix( FILE *fp, char *form, matrix *mat, bool trans, bool line )
```

引数 `fp` 出力するファイルのファイルポインタ.
`form` 個々の行列要素の出力のための書式指定文字列.
`NULL` の場合は matlib 標準の書式が用いられる.
`mat` 表示・出力する行列やベクトル.
`trans` `FALSE` でなければ転置して表示する.
`line` `FALSE` でなければ全ての要素を (改行せずに) 表示する.

返す値 通常は `mat` を返す. 何らかのエラーがあれば `NULL` を返す.

行列変数 `mat` のデータを指定された書式でファイル `fp` に出力する. 書式指定文字列 `form` は基本的に標準ライブラリ関数である `printf` のものに従うが, ひとつの倍精度実数に対応するものでなければならない (例えば`"%10.3lg"`や`" %6.1le "`等). なお, `form` を `NULL` とした場合に用いられる matlib 標準の書式は一つの要素の表示に 11 桁を用いており, 多くの場合見やすく適切であるが, 絶対値が `EPSILON` より小さい値は `0.0` と**表示**されるようにしてあるので若干注意が必要である. なお, ここで述べる他の関数は全てこの `fprintfMatrix` を用いたマクロとして `matrix.h` で定義されている.

他の関数については簡単に述べる.

```
matrix *PrintMatrix( matrix *mat )
```
matlib 標準書式で表示.

```
matrix *PrintTrans( matrix *mat )
```
matlib 標準書式で転置して表示.

```
matrix *fPrintMatrix( FILE *fp, matrix *mat )
```
matlib 標準書式でファイルに出力.

[10] ここで言う表示は出力ファイルを標準出力 (`stdout`) とすることに他ならないので, 本質的には何れもファイルへのテキスト出力なのだが….

`matrix *fPrintTrans( FILE *fp, matrix *mat )`
matlib 標準書式で転置してファイルに出力.

`matrix *PrintFMatrix( char *form, matrix *mat )`
指定された書式で表示.

`matrix *PrintFTrans( char *form, matrix *mat )`
指定された書式で転置して表示.

`matrix *fPrintFMatrix( FILE *fp, char *form, matrix *mat )`
指定された書式でファイルに出力.

`matrix *fPrintFTrans( FILE *fp, char *form, matrix *mat )`
指定された書式で転置してファイルに出力.

なお, ファイルに出力する場合は, `fp` はテキストモードでオープンされている必要がある.

4.1.35 ReadMatrix—テキストファイルからの行列の読み込み

プログラムに手入力でデータ (行列やベクトルの値) を与える場合, 最も単純なのは `scanf` 等の標準入力関数を用いて読み込ませることであろう. しかしながら, データの量が多くなると入力ミスからのリカバリ等のために使い勝手のよいユーザインターフェースが必要となる. そういった (ある意味で本題から離れる) プログラムを書くのは苦痛であることも少なくない. こういう場合には使い慣れたエディタで行列やベクトルのデータをテキストファイルとして作成し, それをプログラムに読み込ませるほうが便利である. また, テキストファイルからのデータの読み込みは, 他のプログラムによる計算結果を読み込んで処理をする場合にも利用できる.

`matrix *ReadMatrix( char *filename, int max_elem )`

引数 `filename` 行列やベクトルが格納されているテキストファイル
`max_elem` 読み込むファイルに予想される最大の要素数

返す値 ファイルに格納されているデータに相当する領域を確保して値を読み込んだ行列やベクトルへのポインタを返す. フォーマットやファイルが巨大すぎる等のエラーがあれば `NULL` を返す.

`ReadMatrix` は, 指定された**テキストファイルが全体として一つの行列あるいはベクトル**のデータを格納しているとして行や列のサイズを調べ, 領域を割り当てた行列・ベクトル変数を生成する. 具体的には, 最初の行で列の数 (すなわち各行の要素数) を確定し, それに従って残りの行の読み込みを行う. 各行は改行コードによって区別され, 行あたりの要素数の過剰・不足については特に考慮されていない. 個々の行列要素は空白もしくはタブによって区切られ, その他の区切り記号 (コンマ (,) やコロン (:) 等) は想定されていないので他のプログラムの出力を読み込む際には注意が必要である[11] .

この関数は `fPrintMatrix` 等によって生成されたファイルを (単一の行列やベクトルのデータ以外のものが含まれていない場合には) 直接読み込むことができる. ただし, **バンドマトリクスには対応していない**. 返される行列変数は (たとえ対角成分にしか非零要素がなかったとしても) 必ず非バンドのマトリクスとして領域確保が行われている.

[11] このような場合は `awk` 等のスクリプト言語を用いて適当な書式変換を行えばよい.

使用例

ファイル A, B に格納されている行列の加算の結果をファイル C に格納するプログラム.

```
#include <stdio.h>
#include "matlib.h"

void main(){
        matrix  *A, *B, *C;
        FILE    *fp;

        A = ReadMatrix( "A", 100 );
        B = ReadMatrix( "B", 100 );
        C = AddMatrix( NULL, A, B );
        fp = fopen( "C", "w" );
        fPrintMatrix( fp, C );
        fclose( fp );
        FreeMatrix( C );
        FreeMatrix( B );
        FreeMatrix( A );
}
```

例えばファイル A, B の中身が

A
1 2 3
4 5 6
7 8 9

B
9 8 7
6 5 4
3 2 1

である場合には, このプログラムの実行によって得られるファイル C の中身は以下のようになる.

C		
10.0000	10.0000	10.0000
10.0000	10.0000	10.0000
10.0000	10.0000	10.0000

4.2 operate.[hc]—基本演算

行列やベクトルの加減算・乗算等の基本的な演算のための関数およびマクロが収められている.

以下の多くの関数では, 返す値は演算結果の格納された行列やベクトルへのポインタとなっている. また, 基本的には演算結果を格納する変数を, 適切なサイズで領域を確保した上で指定することになっているが, その代わりに NULL を指定することのできるものもある. その場合には, 適切な大きさの行列やベクトルの領域を確保して結果を格納した行列やベクトルへのポインタが返される.

特に断りがない限り, 以下の演算にはバンドマトリクスを含めてもかまわない. この場合, (演算の種類にもよるが) 非演算数である行列がバンドマトリクスであれば, 結果格納に NULL を指定した場合に返されるのもバンドマトリクスとなる. また, 結果を格納する変数として指定する場合には, 非バンド領域への書き込みに制約があることに注意が必要である.

4.2.1 Transpose—転置行列

行列の転置を行う. matlib ではベクトルは列ベクトル $(n \times 1)$ として扱われているため, ベクトルが指定された場合には $1 \times n$ 行列が得られる.

```
matrix *Transpose( matrix *matT, matrix *mat )
```

引数 matT 結果を格納する行列変数. NULL も指定可能.

mat 転置する行列.

返す値 通常は matT を返す. matT として NULL が指定された場合, 領域を確保した上で mat の転置が格納された行列へのポインタが返される. 何らかのエラーがあれば NULL を返す.

結果格納場所である matT として mat をそのまま指定することもできる. これは正方行列でない場合でも可能である. この場合, (当然)mat の行と列のサイズは入れ替わることになる. また, mat がバンドマトリクスの場合, matT として NULL を指定した場合にはバンドマトリクスとして領域確保が行われる.

4.2.2 MinusMatrix—要素の符号反転

行列の全ての要素の符号を反転する. 基本的には −1 を乗ずるのと同じことになるが, この関数の内部では負号を付して代入する処理を行っている点が異なる.

```
matrix *MinusMatrix( matrix *matm, matrix *mat )
```

引数 matm 結果を格納する行列変数. NULL も指定可能.

mat 符号を反転する行列.

返す値 通常は matm を返す. matm として NULL が指定された場合, 領域を確保した上で結果が格納された行列へのポインタが返される. 何らかのエラーがあれば NULL を返す.

結果格納場所である matm として mat をそのまま指定することもできる. また, mat がバンドマトリクスの場合, matm として NULL を指定した場合にはバンドマトリクスとして領域確保が行われる.

4.2.3 AddMatrix, SubMatrix, AddMatrixWithCoeff—行列の加減算

行列・ベクトルの加算・減算を行う.

```
matrix *AddMatrix( matrix *mat, matrix *mat1, matrix *mat2 )
```

引数 mat 結果を格納する行列変数. NULL も指定可能.

mat1, mat2 加算を計算する行列.

返す値 通常は mat を返す. mat として NULL が指定された場合, 領域を確保した上で結果が格納された行列へのポインタが返される. 何らかのエラーがあれば NULL を返す.

行列 mat1 と mat2 の和が mat に返される. mat として mat1 あるいは mat2 を指定することも可能.

```
matrix *SubMatrix( matrix *mat, matrix *mat1, matrix *mat2 )
```

引数 mat 結果を格納する行列変数. NULL も指定可能.

mat1, mat2 減算を計算する行列.

返す値 通常は mat を返す. mat として NULL が指定された場合, 領域を確保した上で結果が格納された行列へのポインタが返される. 何らかのエラーがあれば NULL を返す.

行列 mat1 と mat2 の差 (mat1−mat2) が mat に返される. mat として mat1 あるいは mat2 を指定することも可能.

```
matrix *AddMatrixWithCoeff( matrix *mat, matrix *mat1, matrix *mat2,
                            double coeff1, double coeff2 )
```

引数 mat 結果を格納する行列変数. NULL も指定可能.

mat1, mat2 係数つきの加算を計算する行列.

coeff1, coeff2 mat1, mat2 に乗ずる係数.

返す値 通常は mat を返す. mat として NULL が指定された場合, 領域を確保した上で結果が格納された行列へのポインタが返される. 何らかのエラーがあれば NULL を返す.

二つの行列 mat1, mat2 にそれぞれ係数 coeff1, coeff2 を乗じて加算する. mat として mat1 あるいは mat2 を指定することも可能.

これらの関数において, 結果格納場所の mat としてバンドマトリクスを指定する場合には, そのバンド幅が mat1 および mat2 のそれよりも狭くないことが必要である (すなわちその両者がバンドマトリクスである場合に限られる). また, mat として NULL が指定された場合には, mat1 と mat2 のうちのバンド幅の広いほうを参照して領域が確保される.

使用例

行列 A に行列 B, C をそれぞれ 2 倍, 3 倍して加えたものを R に格納する.

```
matrix  *A, *B, *C, *R;
...
R = AddMatrixWithCoeff( NULL, A, B, 1.0, 2.0 );
AddMatrixWithCoeff( R, R, C, 1.0, 3.0 );
...
```

このような計算の場合, A, B および C の値は変化しない.

4.2.4 AddPart, SubPart, AddPartWithCoeff —行列の部分に対する加減算

行列・ベクトルの一部を取り出して加算・減算を行う. 構造解析における要素剛性マトリクスからの全体剛性マトリクスの組み立ての場合等に使用される.

```
matrix *AddPart( matrix *mat, int r, int c, matrix *mat1, int r1, int c1,
                 matrix *mat2, int r2, int c2, int rsiz, int csiz )
```

引数

mat, r, c	結果を格納する行列と格納位置.
mat1, r1, c1, mat2, r2, c2	加算を行う行列の部分の指定.
rsiz, csiz	演算を行う部分のサイズ.

返す値 通常は mat を返す. 何らかのエラーがあれば NULL を返す.

mat1 の r1 行 c1 列から r1+rsiz-1 行 c1+csiz-1 列の大きさ rsiz 行 csiz 列の部分行列と mat2 の r2 行 c2 列からの同様の部分行列を加えて mat の r 行 c 列からの矩形領域に格納する. 格納場所 mat として mat1 あるいは mat2 を指定することも可能だが, 例えば mat1 を指定していて r, c と r1, c1 が異なる場合には注意が必要である[12] .

```
matrix *SubPart( matrix *mat, int r, int c, matrix *mat1, int r1, int c1,
                 matrix *mat2, int r2, int c2, int rsiz, int csiz )
```

引数

mat, r, c	結果を格納する行列と格納位置.
mat1, r1, c1, mat2, r2, c2	減算を行う行列の部分の指定.
rsiz, csiz	演算を行う部分のサイズ.

返す値 通常は mat を返す. 何らかのエラーがあれば NULL を返す.

mat1 と mat2 の指定された部分行列の差を計算する. 部分行列や結果の格納場所の指定については AddPart と同様である.

```
matrix *AddPartWithCoeff( matrix *mat, int r, int c,
                          matrix *mat1, int r1, int c1, matrix *mat2, int r2, int c2,
                          int rsiz, int csiz, double coeff1, double coeff2 )
```

引数

mat, r, c	結果を格納する行列と格納位置.
mat1, r1, c1, mat2, r2, c2	演算対象となる部分行列の指定.
rsiz, csiz	演算を行う部分のサイズ.
coeff1, coeff2	mat1, mat2 にの部分行列に乗ずる係数.

返す値 通常は mat を返す. 何らかのエラーがあれば NULL を返す.

mat1 と mat2 の指定された部分行列にそれぞれ係数 coeff1 および coeff2 を乗じて加えたものを mat の指定された場所に格納する. 部分行列や結果の格納場所の指定については AddPart と同様である.

[12] 特に演算対象となる部分行列と結果を格納する部分行列がずれて重なっている場合, 予期せぬ副作用が起こる可能性がある.

4.2.5 MulScalar—行列・ベクトルのスカラー倍

行列のスカラー倍を計算する.

```
matrix *MulScalar( matrix *mat, matrix *mat0, double scalar )
```

引数 mat 結果を格納する行列変数. NULL も指定可能.
mat0 スカラー倍を計算する行列.
scalar 行列 mat0 の各要素に乗ずるスカラー値.

返す値 通常は mat を返す. mat として NULL が指定された場合, 領域を確保した上で結果が格納された行列へのポインタが返される. 何らかのエラーがあれば NULL を返す.

結果格納場所である mat として mat0 をそのまま指定することもできる. また, mat0 がバンドマトリクスの場合, mat として NULL を指定した場合にはバンドマトリクスとして領域確保が行われる.

4.2.6 MulMatrix, AddMulMatrix, SubMulMatrix—行列の乗算

行列・ベクトルの乗算を行う. また, 複数の行列の積の和を求める場合等に用いることのできる関数もある.

```
matrix *MulMatrix( matrix *mat, matrix *mat1, matrix *mat2 )
```

引数 mat 結果を格納する行列変数. NULL も指定可能.
mat1, mat2 積を計算する行列.

返す値 通常は mat を返す. mat として NULL が指定された場合, 領域を確保した上で結果が格納された行列へのポインタが返される. 何らかのエラーがあれば NULL を返す.

行列 mat1 と mat2 の積 (mat1×mat2) が mat に返される. 例えサイズが等しくても, mat として mat1 あるいは mat2 を指定することはできない. AliasVector や AliasMatrix を用いて別名をつけている行列やベクトルを mat として指定する場合にはその本体と重複しないように注意が必要である.

```
matrix *AddMulMatrix( matrix *mat, matrix *mat1, matrix *mat2 )
```

引数 mat mat1 と mat2 の積を加える行列. NULL も指定可能.
mat1, mat2 積を計算する行列.

返す値 通常は mat を返す. mat として NULL が指定された場合, 領域を確保した上で結果が格納された行列へのポインタが返される. 何らかのエラーがあれば NULL を返す.

行列 mat1 と mat2 の積が mat に加えられる. すなわち, mat+ =mat1×mat2 という処理が行われる. mat に NULL が指定された場合は, 適当な領域を割り当てて mat1×mat2 の計算結果を格納して返す. MulMatrix と同様, 例えサイズが等しくても, mat として mat1 あるいは mat2 を指定することはできない.

```
matrix *SubMulMatrix( matrix *mat, matrix *mat1, matrix *mat2 )
```

引数 mat mat1 と mat2 の積を減ずる行列. NULL も指定可能.
mat1, mat2 積を計算する行列.

返す値 通常は mat を返す. mat として NULL が指定された場合, 領域を確保した上で結果が格納された行列へのポインタが返される. 何らかのエラーがあれば NULL を返す.

行列 mat1 と mat2 の積が mat から引かれる．すなわち, mat− =mat1×mat2 という処理が行われる．他の点については AddMulMatrix と同様.

これらの関数の引数としてバンドマトリクスも指定可能である．ただし, 結果格納場所 mat としてバンドマトリクスを指定する場合には, BSIZE(mat1)+BSIZE(mat2)-1 以上のバンド幅を持っている必要がある．また, mat として NULL が指定され, mat1 および mat2 が共にバンドマトリクスである場合に確保されるマトリクスのバンド幅もこの値となる．

使用例

10 個のベクトルに対応する係数行列を乗じて和を求める．

```
matrix  **A;
vector  **x, *y;

A = AllocMatrixArray( 10, 3, 3 );
x = AllocVectorArray( 10, 3 );
... /* Data setting for 'A' and 'x' */
for( y = NULL, n = 0; n < 10; n++ )
        y = AddMulMatrix( y, A[n], x[n] );
... /* Use calculated result 'y' */
FreeVector( y );
FreeVectorArray( x );
FreeMatrixArray( A );
```

このような for ループを構成すれば, y の初期値が NULL であるため, 自動的に最初 (n=0 のとき) のみ領域が割り当てられ, そのための条件分岐等は不要となる．

4.2.7 Trace—行列のトレース

行列のトレース (対角要素の和) を計算する．

```
double Trace( matrix *mat )
```

引数 mat トレースを計算する行列.

返す値 mat のトレースの値.

本来, トレースは正方行列に対して定義されているが, この関数では行列が正方かどうかのチェックはしていない．正方でない行列の場合には, 存在する全ての対角成分の和を計算する．

4.2.8 Distance, Distance2—二つのベクトルの間の距離

二つのベクトル間のユークリッド距離あるいはその二乗を計算する．

```
double Distance( vector *v1, vector *v2 )
```

引数 v1, v2 距離を計算するベクトル.

返す値 v1 と v2 の間のユークリッド距離．エラー (サイズが合わない等) の場合は負の数 (−1.0) を返す.

```
double Distance2( vector *v1, vector *v2 )
```

引数 v1, v2 距離の二乗を計算するベクトル.

返す値 v1 と v2 の間のユークリッド距離の二乗 (ベクトルの要素の差の二乗和). エラー (サイズが合わない等) の場合は負の数 (−1.0) を返す.

4.2.9 ScalarP, VectorP, MakeMVP— ベクトルのスカラー積, ベクトル積

二つのベクトルの間のスカラー積 (内積) およびベクトル積 (外積) を計算する.

```
double ScalarP( vector *a, vector *b )
```

引数 a, b スカラー積を計算するベクトル.

返す値 ベクトル a, b のスカラー積.

引数の a, b は行列でもかまわない. その場合は行列の全要素がベクトル化されているとして計算を行う. ただし, バンドマトリクスは不可.

```
vector *VectorP( vector *v, vector *a, vector *b )
```

引数 v 結果を格納する 3 次元ベクトル. NULL も指定可能.
a, b ベクトル積を計算する 3 次元ベクトル.

返す値 通常は v を返す. v として NULL が指定された場合, 領域を確保した上で結果が格納されたベクトルへのポインタが返される. 何らかのエラーがあれば NULL を返す.

引数の v, a, b は 3 次元ベクトルに限定される (v は NULL も指定できるが).

```
matrix *MakeMVP( matrix *MVPA, vector A )
```

引数 MVPA 結果を格納する 3×3 行列. NULL も指定可能.
A 3 次元ベクトル.

返す値 通常は MVPA を返す. MVPA として NULL が指定された場合, 領域を確保した上で結果が格納された 3×3 行列へのポインタが返される. 何らかのエラーがあれば NULL を返す.

ベクトル積を「行列×ベクトル」の形で計算することを可能とする. すなわち, 3 次元ベクトル $\boldsymbol{A}, \boldsymbol{B}$ のベクトル積の計算を

$$\boldsymbol{A} \times \boldsymbol{B} \Longrightarrow \mathrm{MVP}(\boldsymbol{A}) \cdot \boldsymbol{B}$$

と変形する場合の 3×3 行列 $\mathrm{MVP}(\boldsymbol{A})$ を求める. ベクトル積が用いられた関数のヤコビ行列の計算等に用いる.

4.2.10 CoordinateTransform—座標変換

ベクトルに対する一般的な座標変換, すなわち回転変換と並進変換を計算する. (構造解析計算等で用いられる) 部分ベクトルの集合である全体ベクトルに対する座標変換を一括で処理する.

```
vector *CoordinateTransform( vector *X1, vector *X0, matrix *R, vector *T )
```

引数 X1 座標変換結果. NULL が指定可能.
X0 座標変換を計算する (集合) ベクトル.
R 回転行列.
T 並進ベクトル

返す値 通常は X1 を返す. X1 として NULL が指定された場合, 領域を確保した上で結果が格納されたベクトルへのポインタが返される. 何らかのエラーがあれば NULL を返す.

座標系 0 を参照したベクトル ${}^0\boldsymbol{x}$ と座標系 1 を参照したベクトル ${}^1\boldsymbol{x}$ の間の座標変換が回転行列 $\boldsymbol{R}$ と並進ベクトル $\boldsymbol{T}$ を用いて

$$ {}^1\boldsymbol{x} = \boldsymbol{R} \cdot {}^0\boldsymbol{x} + \boldsymbol{T} $$

と表されるときに, これらのベクトル N 個で構成される集合ベクトル ${}^0\boldsymbol{X} = [{}^0\boldsymbol{x}_1^T, \cdots, {}^0\boldsymbol{x}_N^T]^T$ と ${}^1\boldsymbol{X} = [{}^1\boldsymbol{x}_1^T, \cdots, {}^1\boldsymbol{x}_N^T]^T$ の間の座標変換を一括して行う. 結果を格納する変数である X1 として X0 を指定することもできるが, その場合には明確に (実体が) 同一の変数であることを示す必要があるため, AliasVector 等を用いている場合には注意が必要である.

4.2.11 L1Norm, L2Norm, L2Norm2—行列・ベクトルのノルム

行列やベクトルに対するいくつかのノルム (大きさを測る尺度) を計算する. ただし, L1 ノルムは全要素の絶対値の合計, L2 ノルムは全要素の二乗の合計の平方根 (1/2 乗) である.

```
double L1Norm( matrix *mat )
```

引数 mat L1 ノルムを計算する行列やベクトル.

返す値 計算されたノルム.

```
double L2Norm( matrix *mat )
```

引数 mat L2 ノルムを計算する行列やベクトル.

返す値 計算されたノルム.

```
double L2Norm2( matrix *mat )
```

引数 mat L2 ノルムの二乗を計算する行列やベクトル.

返す値 計算された値.

行列やベクトルの全要素の二乗和を計算する.

4.3 inverse.[hc]—逆行列・線形方程式

逆行列の計算や連立方程式を解くための関数が収められている.

4.3.1 LastDeterminant, LastTeraN—直前に計算された行列式の値

連立方程式の解や逆行列の計算の際の行列式の値を返す. 行列の内容や規模によっては行列式の値が非常に大きくなることから, 以下の二つの関数による構成となっている.

```
double LastDeterminant()
```

```
int LastTeraN()
```

これらの関数は直前の Gauss や Inverse で扱った行列の行列式の値を返す. 実際の行列式の値 D は, LastDeterminant が返す値を d, LastTeraN が返す値を N として, $D = d \times 10^{12N}$ で与えられる.

4.3.2 Gauss—ガウスの消去法による連立方程式の解

連立方程式 $\boldsymbol{ax} = \boldsymbol{b}$ の解 $\boldsymbol{x}$ をガウスの消去法を用いて求める.

```
double Gauss( matrix *a, vector *x, vector *b )
```

引数 a 正方係数行列.
x 未知ベクトル.
b 定数ベクトル.

返す値 正方行列 a の行列式の値を返す. ただし, 行列式の正しい値については LastTeraN を参照する必要がある. 行列 a が特異であった場合や何らかのエラーが発生した場合は 0.0 を返す.

線形方程式 a×x=b の解を計算する. この関数においては x は予め適切な次元のベクトルとして準備されている必要がある (NULL は不可). また, 処理の過程で行列 a およびベクトル b の内容は破壊される. 行列 a としてバンドマトリクスも指定可能だが, この場合はピボット選択の処理が行われないため, 実際には行列 a が正則であるにもかかわらず特異と判断して 0.0 を返す可能性がある[13] .

4.3.3 Inverse—逆行列の計算

LU 分解法を用いて逆行列を計算する.

```
double Inverse( matrix *A, double MinPiv )
```

引数 A 逆行列を求める正方行列. 結果もここに格納されるため, バンドマトリクスは不可.
MinPiv 許容できる最小のピボットの大きさ.

返す値 正方行列 A の行列式の値を返す. ただし, 行列式の正しい値については LastTeraN を参照する必要がある. 行列 A が特異であった場合や何らかのエラーが発生した場合は 0.0 を返す.

行列 A の逆行列を計算する. 結果はそのまま行列 A に上書きされる. 計算の過程でピボットの大きさが MinPiv を下回った場合にはピボット選択が行われるが, MinPiv 以上の大きさのピボットが発見できない場合にはその時点で行列 A が特異であると判断する. しかしながら, この場合, 実際には行列 A が特異ではない可能性もある. MinPiv として 0.0 が与えられている場合には, 常に最大の大きさのピボットを探索しつつ計算が行われるためこのようなことがなく, 計算の精度も向上するが, 関連する処理の影響で計算時間は格段に長くなる (行列の内容にもよるが, 比較的小規模の行列 (10×10 程度) で 2〜3 倍, 中規模の行列 (100×100 程度) で 5〜7 倍, 大規模の行列 (1000×1000 程度) では 20〜30 倍にもなる). 通常は, MinPiv として行列の要素の値の平均的な大きさの千分の一程度の値を用いれば特に問題はないようである.

4.3.4 IndependentVectors—一次独立なベクトルの探索

4.3.5 ExteriorProduct—一般次元空間における外積の計算

4.4 ginverse.[hc]—一般化逆行列

一般化逆行列[14] . には様々なものがあるが, matlib では現在のところ疑似逆行列とノルム最小化型一般化逆行列を準備している. 一般化逆行列の詳細については適当な書籍[15] を参照のこと.

[13] 実用的には, バンドマトリクスを用いているような場合にこのようなこと (ピボットの問題) が起こることはあまりないようだ.
[14] 「一般逆行列」と「一般**化**逆行列」の二種類の用語があるようだが, 英語では “generalized inverse” なので, どちらかといえば「一般化逆行列」のほうが適切だと著者は考える.
[15] 例えば, ラオ, ミトラ (渋谷, 田辺訳)「一般逆行列とその応用」東京図書 (1973) (この本は絶版のようだが大学図書館等にはたぶんあるはず) や半谷, 川口「形態解析 (一般逆行列とその応用)」倍風館 (計算力学と CAE シリーズ 5) (1991) が挙げられる.

4.4.1 LastRank—直前の (一般化逆行列の計算に用いた行列の) 階数

フルランクでない一般化逆行列を計算した場合の行列の階数.

int LastRank() 直前に一般化逆行列を計算した行列の階数 (ランク) を返す. ただし, 計算の過程における正方行列の逆行列を求める手法として MOORE_INV を指定した場合に限られる.

4.4.2 PseudoInverse—疑似逆行列 (ムーア・ペンローズ一般化逆行列)

疑似逆行列とは, 線形方程式 $\boldsymbol{Ax} = \boldsymbol{y}$ において $\boldsymbol{A}$ の行数が列数より少なくかつ行フルランク ($\boldsymbol{A}$ の階数が行数と等しい) である場合に, $\boldsymbol{x} = \boldsymbol{A}^+\boldsymbol{y}$ として $\boldsymbol{x}^T\boldsymbol{x}$ が最小となる解を与える行列 $\boldsymbol{A}^+$ を指す[16]. このような $\boldsymbol{A}^+$ をフルランクでない場合にまで一般化したものがムーア・ペンローズ一般化逆行列である[17]. この関数は疑似逆行列あるいはムーア・ペンローズ一般化逆行列を計算するものである.

```
matrix *PseudoInverse( matrix *InvAmr, matrix *A, double MinPiv, int SelInv )
```

引数 InvAmr 計算された一般化逆行列を格納. NULL も指定可能.
A 一般化逆行列を計算する行列.
MinPiv 許容される最小のピボット.
SelInv 計算過程での正方行列の逆行列の計算手法.
NORMAL_INV: 通常の逆行列計算 (Inverse) を使用.
MOORE_INV: ムーア・ペンローズ一般化逆行列を計算.

返す値 通常は InvAmr を返す. InvAmr として NULL が指定された場合, 適当な領域を割り当てて計算した一般化逆行列を格納して返す. 何らかのエラーがあれば NULL を返す. なお, SelInv として NORMAL_INV を指定したにもかかわらず行列 A がフルランクではなかった場合 (下記参照) にも NULL を返す.

与えられた行列 A の疑似逆行列あるいはムーアペンローズ一般化逆行列を計算する. 行列 A が行または列のフルランクであることが明らかな場合には SelInv として NORMAL_INV を指定する. この場合, 計算過程での正方行列の逆行列の計算には関数 Inverse を用いるので, MinPiv として 0.0 も使用可能である (Inverse を参照). 行列 A がフルランクでない場合あるいは不明な場合には SelInv として MOORE_INV を指定する. この場合には MinPiv として正の適当な値を指定する.

許容最小ピボット MinPiv は行列 A の階数の決定に直接関わるため, SelInv として MOORE_INV を指定した場合, MinPiv の大小によって得られる一般化逆行列が異なる可能性がある.

4.4.3 MinNormGinv—ノルム最小化型一般化逆行列

二次形式の最小化問題

$$\text{Minimize} \quad \boldsymbol{x}^T\boldsymbol{N}\boldsymbol{x} \quad \text{with respect to} \quad \boldsymbol{x} \quad \text{subject to} \quad \boldsymbol{Ax} = \boldsymbol{y}$$

の解を $\boldsymbol{x} = \boldsymbol{A}^+\boldsymbol{y}$ として求める場合の行列 $\boldsymbol{A}^+$ を求める.

```
matrix *MinNormGinv( matrix *InvAmr, matrix *A, matrix *N,
                     double MinPiv, double SelInv )
```

[16] 疑似逆行列の定式化は Lagrange 乗数法を用いれば簡単に行えるので, 是非一度やってみることを勧める.
[17] ここでの説明はかなりいいかげんなものなので, 詳細については書籍等を参照すること.

引数 InvAmr 計算された一般化逆行列を格納. NULL も指定可能.
A 一般化逆行列を計算する行列.
N ノルム指定行列. 正定でなければならない.
MinPiv 許容される最小のピボット.
SelInv 計算過程での正方行列の逆行列の計算手法.
NORMAL_INV: 通常の逆行列計算 (Inverse) を使用.
MOORE_INV: ムーア・ペンローズ一般化逆行列を計算.

返す値 通常は InvAmr を返す. InvAmr として NULL が指定された場合, 適当な領域を割り当てて計算した一般化逆行列を格納して返す. 何らかのエラーがあれば NULL を返す. なお, SelInv として NORMAL_INV を指定したにもかかわらず行列 A がフルランクではなかった場合にも NULL を返す.

SelInv および MinPiv については PseudoInverse の説明を参照のこと.

4.5 eigen.[hc]—行列の固有値

行列の固有値の計算を行う. 元来が構造解析における固有振動数の計算を目的としたものであるため, 複素固有値は想定していない. また, べき乗法を用いているため, 固有値計算が可能な行列には若干の制約がある. ただし, 構造解析の動力学計算における固有振動数や振動モードの計算には特に支障はないようだ.

4.5.1 EigenPower—べき乗法による固有値・固有ベクトルの計算

べき乗法を用いて与えられた線形方程式の固有値および固有ベクトルを計算する関数である. 具体的には, 行列 $\boldsymbol{A}$, $\boldsymbol{B}$ に対し, $\boldsymbol{Ax} = \lambda \boldsymbol{Bx}$ が成り立つ最大の固有値 $\boldsymbol{\lambda}$ および対応する固有ベクトル $\boldsymbol{x}$ を求める. いわゆる行列の固有値および固有ベクトルを求める場合には, $\boldsymbol{B}$ として単位行列を指定すれば, 行列 $\boldsymbol{A}$ の固有値が得られる.

```
double EigenPower( matrix *A, matrix *B, vector *eivec,
                   int maxiter, double limit, bool *convflg )
```

引数 A, B 固有値を求める線形方程式を構成する行列 $\boldsymbol{A}$, $\boldsymbol{B}$.
B は正則である必要がある. B が NULL の場合には単位行列と見なす.
eivec 固有ベクトルの初期値. 求められた固有ベクトルもここに返される.
maxiter 繰り返し回数. 0 の場合は収束するまで繰り返す.
limit 収束誤差. 固有ベクトルの L2 ノルムで評価.
convflg 収束した (TRUE) か否 (FALSE) かが返される.

返す値 求められた固有値を返す. 収束条件を満たさずに終了した場合 (*convflg が FALSE) でも, その時点で得られている固有値を返す. 何らかのエラーがあれば 0.0 を返す.

べき乗法を用いて最大固有値および対応する固有ベクトルを計算する. 固有ベクトルの初期値を eivec に与える必要があるが, 特に意図がなければ, 全ての要素を 1.0 とするのが概ね適当であろう. この作業には FillMatrix 等を用いればよい. なお, 得られた eivec はその最も大きい要素の値で正規化されている.

EigenPower は最大固有値を求めるものであるが, A と B を入れ替えて EigenPower を使用し, 得られた固有値の逆数を計算すれば, 最小固有値が得られる (使用例を参照). ただし, この場合, 行列 A は正則でなければならない.

使用例

行列 A の最大固有値 lambda を求める.

```
matrix  *A;
vector  *eivec;
double  lambda;
bool    convflg;
...
FillMatrix( eivec, 1.0 );
lambda = EigenPower( A, NULL, eivec, 1000, 1.0e-6, &convflg );
if( convflg == FALSE )
        printf( "Fail to calculate\n" );
...
```

行列 A の最小固有値 lambda を求める.

```
matrix  *A, *B;
vector  *eivec;
double  adbmal, lambda;
bool    convflg;
...
B = UnitMatrix( AllocSize( A ) );
FillMatrix( eivec, 1.0 );
adbmal = EigenPower( B, A, eivec, 1000, 1.0e-6, &convflg );
if( convflg == FALSE )
        printf( "Fail to calculate\n" );
lambda = 1.0 / adbmal;
...
```

4.5.2 OrthogonalEigens—複数の固有値・固有ベクトルの計算

対称行列の異なる固有値に対する固有ベクトルは互いに直交する. このことを用い, べき乗法により, 対称行列 $\boldsymbol{A}$, $\boldsymbol{B}$ により構成される線形方程式 $\boldsymbol{Ax} = \lambda\boldsymbol{Bx}$ の複数の固有値・固有ベクトルを求める[18] .

```
int OrthogonalEigens( matrix *A, matrix *B, vector *eivals, matrix *eivecs,
                      int maxiter, double limit, bool contflg )
```

引数 A, B 固有値を求める線形方程式を構成する**対称行列** $\boldsymbol{A}$, $\boldsymbol{B}$.
B は正則である必要がある. B が NULL の場合には単位行列と見なす.

eivals 得られた固有値が格納されるベクトル. この次元により計算する固有値の数を指定.

eivecs 得られた固有ベクトルを列ベクトルとして格納する行列. 特に初期値は必要ない.

maxiter 個々の固有値に対する繰り返し回数. 0 の場合は収束するまで繰り返す.

limit 許容誤差. 固有ベクトルの L2 ノルムで評価.

contflg 固有値計算が途中で収束しない場合の継続 (TRUE) か中止 (FALSE) かの指定.

返す値 計算された固有値および固有ベクトルの数を返す. 何らかのエラーがあれば -1 を返す.

対称行列の固有ベクトルの直交性を用いてべき乗法により複数の固有値および固有ベクトルを計算する. 求められた固有値は大きいものから順に eivals に格納される. また, 対応する固有ベクトルは列ベクトルとして eivecs に格納される. 計算する固有値の数はベクトル eivals によって決まる. ある固有値の計算が収束しなかった場合, contflg として中止 (FALSE) が指定されていれば, (それが最初の固有値でなければ) 未収束の固有値は破棄され, 残りの部分には 0.0 が格納されるが, contflg として継続 (TRUE) が指定されていた場合には固有値および固有ベクトルを未収束のまま eivals お

[18] 行列 $\boldsymbol{A}$ および $\boldsymbol{B}$ の双方が共に単位行列ではない場合の固有ベクトル $\boldsymbol{x}_i, \boldsymbol{x}_j (i \neq j)$ の「直交」とは $\boldsymbol{x}_i^T\boldsymbol{A}\boldsymbol{x}_j = 0$ あるいは $\boldsymbol{x}_i^T\boldsymbol{B}\boldsymbol{x}_j = 0$ であることに注意.

よび eivecs に格納して次の固有値の計算に移る．なお, 得られた eivecs 内の固有ベクトルは, 各々その最も大きい要素の値で正規化されている．

OrthogonalEigens は固有値を大きいほうから求めるものであるが, EigenPower と同様に A と B を入れ替えて使用し, 得られた固有値の逆数を計算することにより, 小さいほうからの固有値を求めることもできる．

使用例

慣性マトリクス $\boldsymbol{M}$, 剛性マトリクス $\boldsymbol{K}$ が与えられているときに, 固有振動数を低いほうから (最大)3 次まで計算し, 対応する固有振動モードを求める．この場合の固有方程式は振動モードベクトルを $\boldsymbol{u}$, 角振動数を ω とすると

$$\omega^2 \boldsymbol{M}\boldsymbol{u} = \boldsymbol{K}\boldsymbol{u}$$

である．OrthogonalEigens は固有値を大きいほうから求めるものであるため, この式を下記のように変形して用いる．

$$\boldsymbol{M}\boldsymbol{u} = \frac{1}{\omega^2}\boldsymbol{K}\boldsymbol{u}$$

すなわち, 固有値として $1/\omega^2$ を大きいほうから求め, これを用いて固有振動数を低いほうから求める．

```
matrix  *M, *K, *Modes;
vector  *Freqs;
int     n;
...
Freqs = AllocVector( 3 );
Modes = AllocMatrix( RSIZE(M), 3 );
n = OrthogonalEigens( M, K, Freqs, Modes, 1000, 1.0e-6, FALSE );
printf( "\n... %d mode(s) are calculated.\n", n );
for( m = 1; m <= n; m++ )
        VVAL(Freqs,m) = 1.0 / (2.0 * 3.1416 * sqrt( VVAL(Freqs,m) ));
...
```

4.6 newton.[hc]—ニュートン法による非線形方程式の解

4.7 rotate.[hc]—回転行列

4.8 lsa.[hc]—最小二乗法

4.9 newmark.[hc]—ニューマーク β 法による数値積分

4.10 steepest.[hc]—最急降下法

4.11 udrand.[hc]—実数値乱数

4.12 simplex.c—非線形計画問題のシンプレックス法

4.13 intmat.[hc]—整数値行列

第5章　プログラム例—簡易行列電卓mat

5.1　はじめに

matlib のサンプルプログラムとして簡易行列電卓 mat を紹介する．これは適当なフォーマット (ReadMatrix を参照) で数値データが格納されたテキストファイルを各々行列やベクトルと見なして演算処理を行うものである．演算順序の指定は逆ポーランド記法[1] に従う．入出力がそれほど桁数の多くないテキストファイルであるため精度はそれほど高くないが, ちょっとした行列計算には便利である．

5.2　プログラムのコンパイル

基本的には unix の場合であれば

```
make mat
```

bcc32 であれば

```
make mat.exe
```

とすればコンパイルが行える．上記の方法が適用できない場合, 他のプログラムと同様に, mat.c をコンパイルして matlib のライブラリとリンクすればよい．詳しくは 3.1 節を参照のこと．

コンパイル・リンクが完了後,

```
mat
```

と起動すれば, 下記のように表示される．

```
** Simple matrix calculator for reverse Polish notaion **
   Binary operator : '+' Add., '-' Sub., 'x' Mul.
   Unary operator  : '-1' Inv., '^' Scalar times, '=' Assign.

 Example: D = Inv[(-2.5 A + B) - C] --> mat A ^-2.5 B + C - -1 =D
```

すなわち, mat において用意されている二項演算子は+(加算), -(減算), x(乗算), 単項演算子は-1(逆行列), ^数値 (スカラー倍), =ファイル名 (代入) である．mat は最後に結果 (スタックの中身) を表示して終了する．

5.3　使用例

以下の線形方程式を解いて未知数 $\boldsymbol{U}$ を求める．

$$(\boldsymbol{A}+\boldsymbol{B})\boldsymbol{C}\boldsymbol{U}=\boldsymbol{Y}$$

[1] これがわからない場合は構文解析のテキスト等によく登場するのでそういった書籍や適当なサイトを参照のこと．簡単にいうと $a+b$ を $ab+$, $(a+b)\times c$ を $ab+c\times$, $c\times(a+b)$ を $cab+\times$ とするような表記法．いわゆる計算順序を指定するための括弧 () が不要になる．

まず, 行列 $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ およびベクトル (実は行列でもかまわないが)$\boldsymbol{Y}$ をそれぞれ適当なファイルとして準備する. たとえば, A, B, C, Y というファイル名でテキストエディタ等を用いて作成する. これらのファイルのあるディレクトリで mat を以下のように起動すれば (mat と異なるディレクトリで作業を行う場合は, mat に PATH を通しておく必要がある),

```
mat A B + C x -1 Y x
```

計算結果 ($[(\boldsymbol{A}+\boldsymbol{B})\boldsymbol{C}]^{-1}\boldsymbol{Y}$) を表示して終了する. また, 下記のように単項演算子=U を追加すれば, 結果をファイル U に格納した後に表示して終了する.

```
mat A B + C x -1 Y x =U
```

また, 例えば $(\boldsymbol{A}+\boldsymbol{B})\boldsymbol{C}$ が正則でない場合には下記のようなエラー表示がなされる.

```
Singular matrix.
Operation error.
>> A B + C x !~1! Y x =U
```

第6章　おわりに

十年以上にわたって書き溜めてきたプログラムのうち, 汎用性の高いものを公開しようという考えは数年前から持っていたのだが, マニュアルの整備を真面目にやろうとすると思いのほか時間がかかり, 不十分なまま公開することにした (4 章を見ればよくわかる…). 残りの関数群 (まだかなりたくさんある) に関する記述はおいおい時間のあるときにでも行ってゆく予定である.

現状のマニュアルの範囲でもそれなりに使い物にはなるのではないかと考えているが, 利用された方はバグ情報と共に感想や要望等を連絡いただければ幸いである. やはり反響があれば, マニュアルの未完成な部分の補完や, 記述の ANSI 化を進めるのにも熱が入ろうというものである.

ここまで書くだけでも思いのほか時間を取られてしまったので, とりあえず今のところはここまでとする.

2005/10/04 記

索 引