GMATRIX: Uma bilioteca matricial para C/C++

Versão 1.0

Geovany A. Borges {gaborges@ene.unb.br}

Departamento de Engenharia Elétrica - ENE
Faculdade de Tecnologia - FT
Universidade de Brasília - UnB
Brasília - DF - Brasil

10 de outubro de 2005

Sumário

1 Introdução

O emprego de vetores e matrizes é bastante comum em computação científica. Apesar de várias linguagens de programação suportarem este tipo de variável, sua manipulação impõe restrições ou requer bastante cuidado por parte do usuário para, por exemplo, não utilizar índices que vão além do tamanho da variável. A biblioteca `GMATRIX` pode ser vista como uma alternativa para a implementação de matrizes em linguagem C, em que o tipo matriz é uma estrutura de dados. Ela vem sendo aperfeiçoada há pelo menos cinco anos, tendo sido usada com sucesso em vários projetos de robótica, em que a precisão e a segurança se fazem necessários. Várias rotinas de cálculo matricial e álgebra linear estão incluídas na biblioteca, além de interface MATLAB e suporte para microcontroladores.

2 Estrutura matricial e inicialização

A estrutura de dados mais importante da bilioteca é `GMATRIX`, cujo ponteiro é definido como `PGMATRIX`. Sua definição é mostrada abaixo:

```
typedef struct{
    int Nr; /* Number of rows of the matrix */
    int Nc; /* Number of columns of the matrix */
    int MaxSize; /* Number of entries of *Data.
                    Used mainly in automatic matrix resizing (if enabled)*/
    double *Data; /* Pointer to entries of GMATRIX,
                     organized sequentially, column-by-column */
} GMATRIX, *PGMATRIX;
```

Em `GMATRIX` , o ponteiro `Data` aponta para um vetor de tamanho `MaxSize*sizeof(double)`. Neste vetor os elementos da matriz são organizados seqüencialmente, coluna por coluna. Os índices de cada elemento são referenciados de forma similar ao MATLAB. Por exemplo, para uma matriz $A = \{a_{ij}\}$, $i = 1, \cdots, N_r$ e $j = 1, \cdots, N_c$, o índice k em ANSI-C do elemento a_{ij} no vetor apontado por `Data` é dado por

$$k = i - 1 + (j - 1)N_r \tag{1}$$

ou seja, `Data[k]` $\Longleftrightarrow a_{ij}$. Para acessar elementos de matrizes, foram definidas as macros

```
#define GMATRIX_DATA(Mat,i,j) (*(&Mat.Data[i-1+(j-1)*Mat.Nr]))
#define PGMATRIX_DATA(pMat,i,j) (*(&pMat->Data[i-1+(j-1)*pMat->Nr]))
```

Estas macros permitem acessar elementos de matrizes (`GMATRIX`) e de ponteiros para matrizes (`PGMATRIX`). Por exemplo, para colocar o valor 1,5 no elemento da linha 2 coluna 1 da variável matricial `Mat`, deve ser usado

```
GMATRIX_DATA(Mat,2,1) = 1.5;
```

Estas macros também permitem ser usadas como operadores à direita. Por exemplo, para colocar na variável `a` o conteúdo do elemento da linha 3 coluna 5 da matriz apontada por `pMat`, deve ser usado

```
a = PGMATRIX_DATA(pMat,3,5);
```

A variável MaxSize contém o número de elementos do vetor apontado por `Data`. Esta variável é útil quando se deseja redimensionar automaticamente uma matriz em algumas funções que usam matrizes auxiliares. Entretanto, nesta versão o dimensionamento automático ainda não está implementado.

A declaração de uma matriz é feita pela macro `GMATRIX_DECLARE`. Assim, para declarar uma matriz `Mat` de dimensão 3×2, é necessário fazer

```
GMATRIX_DECLARE(Mat,3,2);
```

Na verdade, o comando acima cria duas variáveis: o vetor `Mat_Data[]`, de comprimento 3×2, e a estrutura `Mat` de tipo `GMATRIX`, devidamente inicializada, inclusive com `Mat.Data` apontando para o endereço de `Mat_Data[0]`.

Quando a dimensão de uma matriz não for conhecida *a priori* (*e.g.* se depender de variáveis), torna-se conveniente lançar mão da alocação dinâmica. Para tanto, o usuário poderá usar a função `PGMATRIX_ALLOC` que retorna um ponteiro `PGMATRIX` para a matriz alocada. Por exemplo, para alocar uma matriz de dimensão 10×1000 basta fazer

```
PGMATRIX pMat; // Declaracao do ponteiro
pMat = PGMATRIX_ALLOC(10, 1000); // Alocacao
.
.
PGMATRIX_FREE(pMat);
```

No código acima `PGMATRIX_FREE` desaloca o espaço de memória apontado por `pMat`. Esta operação deve ser realizada quando `pMat` não for mais usado. Desta forma evita-se a retenção desnecessária de espaço de memória.

Algumas funções fazem uso de matrizes auxiliares, que devem ser passadas como argumento. Para tanto, o seguinte tipo é definido:

```
typedef struct{
    PGMATRIX pMat1; /* Auxiliary Matrix 1 */
    PGMATRIX pMat2; /* Auxiliary Matrix 2 */
    PGMATRIX pMat3; /* Auxiliary Matrix 3 */
    PGMATRIX pMat4; /* Auxiliary Matrix 4 */
} DUMMY_MATRICES,*PDUMMY_MATRICES;
```

Os elementos desta estrutura devem ser alocados previamente pela função de chamada. As dimensões de cada matriz auxiliar (ou *dummy*, do inglês) são determinadas de acordo com a função que as utiliza, conforme descrito na seção 3.

Sempre que for desejado alterar as dimensões de uma matriz, o usuário poderá fazê-lo através do código

```
Mat.Nr = 3;
Mat.Nc = 6;
```

No entanto, esta alteração resultar em dimensões diferentes com relação à declaração inicial de `Mat`, o acesso ao conteúdo por meio de `GMATRIX_DATA` fica comprometido, devido à organização dos dados ainda estar na forma inicial. Por outro lado, esta alteração não pode resultar em `Mat.Nr*Mat.Nc` > `MaxSize`, sob pena de qualquer acesos futuro usando `GMATRIX_DATA` resulte em endereços que estão fora do vetor `Mat.Data`.

Este tipo de manipulação de dimensões é indicado apenas para adequar uma matriz que receberá o resultado de uma operação, de dimensões possivelmente diferentes às originais. Para tanto, sugere-se o uso da macro `GMATRIX_SETSIZE`, que incorpora proteção contra erro de dimensão:

```
#define GMATRIX_SETSIZE(Mat,Nll,Ncc) PGMATRIX_SETSIZE(&Mat,Nll,Ncc)
void PGMATRIX_SETSIZE(PGMATRIX pMat,int Nll, int Ncc);
```

3 Recursos de GMATRIX

A seguir são apresentadas descrições com exemplos das funções implementadas na biblioteca GMATRIX. Todas as funções foram escritas para trabalhar com ponteiros de matrizes. No entanto, são definidas macros que facilitam o uso das funções por quem não tem o hábito de trabalhar com ponteiros. Por exemplo, considera-se a função `PGMATRIX_SETSIZE`, escrita utilizando argumentos sob a forma de ponteiros, e `GMATRIX_SETSIZE` que é a macro associada. Ambas são apresentadas no texto conforme estão definidas na biblioteca:

```
#define GMATRIX_SETSIZE(Mat,Nll,Ncc) PGMATRIX_SETSIZE(&Mat,Nll,Ncc)
void PGMATRIX_SETSIZE(PGMATRIX pMat,int Nll, int Ncc);
```

Embora nos dois casos seja a função `PGMATRIX_SETSIZE` quem é executada, `GMATRIX_SETSIZE` é usada quando se dispõe da matriz `Mat` definida no escopo da função de chamada. Por outro lado, `PGMATRIX_SETSIZE` trabalha com ponteiros de matrizes. O exemplo abaixo ilustra os diferentes usos:

```
GMATRIX_DECLARE(MatA,2,4);
PGMATRIX pMatB;

pMatB = PGMATRIX_ALLOC(5,5); // Alocacao de pMatB
GMATRIX_SETSIZE(MatA,1,5);  // Redimensionamento de MatA
PGMATRIX_SETSIZE(pMatB,1,5);  // Redimensionamento de pMatB
```

No mais, quando se deseja passas uma matriz como argumento para uma função, esta passagem é mais eficiente se for feita através de ponteiro. Por exemplo, uma função definida como `void Funcao(PGMATRIX pMatA, int sigma)` pode ser chamada pelas seguintes formas:

```
Funcao(pMatA, 2); // neste caso, pMatA e' um ponteiro definido.
Funcao(&MatB, 2); // neste caso, MatB e' uma matriz.
```

Qualquer especificidade será abordada nas subseções que seguem.

3.1 Constantes e macros

Os números π, 2π e e (neperiano) são representados respectivamente pelas macros

```
#define GMATRIXCONST_PI    3.14159265358979310000
#define GMATRIXCONST_2PI   6.28318530717958620000
#define GMATRIXCONST_E     2.71828182845904550000
```

Algumas operações simples definidas apenas para escalares estão disponíveis sob a forma de macros:

```
#define GMATRIXMACRO_SIGN(a) ((a) >= 0.0 ? 1.0 : -1.0)
#define GMATRIXMACRO_SWAP(a,b) {temp = (a); (a) = (b); (b) = temp;}
#define GMATRIXMACRO_MAX(a,b) ((a) > (b) ? (a) : (b))
#define GMATRIXMACRO_MIN(a,b) ((a) > (b) ? (b) : (a))
#define GMATRIXMACRO_SQR(a) ((a)*(a))
#define GMATRIXMACRO_SIGNCHAR(a) ((a) >= 0.0 ? ' ' : '-')
```

Alguns exemplos de uso e seu significado são mostrados na tabela abaixo:

Código C	**Significado**
`b = GMATRIXMACRO_SIGN(a);`	$\mathtt{b} = \begin{cases} 1 & \text{se } \mathtt{a} \geq 0 \\ -1 & \text{caso contrário} \end{cases}$
`GMATRIXMACRO_SWAP(a,b);`	Os conteúdos de `a` e `b` são trocados. É preciso definir a variável `temp`.
`c = GMATRIXMACRO_MAX(a,b);`	$\mathtt{c} = \begin{cases} \mathtt{a} & \text{se } \mathtt{a} > \mathtt{b} \\ \mathtt{b} & \text{caso contrário} \end{cases}$
`c = GMATRIXMACRO_MIN(a,b);`	$\mathtt{c} = \begin{cases} \mathtt{a} & \text{se } \mathtt{a} \leq \mathtt{b} \\ \mathtt{b} & \text{caso contrário} \end{cases}$
`b = GMATRIXMACRO_SQR(a);`	$\mathtt{c} = \mathtt{a}^2$

Por fim, com `ch = GMATRIXMACRO_SIGNCHAR(a)`, `ch` é uma variável `unsigned char` que recebe um caractere '-' se `a` for negativo ou ' ' em caso contrário. Esta macro é usada por outras funções para imprimir na tela, de forma formatada, o conteúdo de matrizes.

3.2 Inicialização de matrizes especiais

As funções a seguir são usadas para iniciar o conteúdo de matrizes especiais. Elas presumem que a estrutura da matriz foi devidamente declarada/alocada.

- Matriz com 0s em todos os elementos, $A = \{a_{ij}\}$ com $a_{ij} = 0$:

```
#define GMATRIX_ZEROES(Mat) PGMATRIX_ZEROES(&Mat)
void PGMATRIX_ZEROES(PGMATRIX pMat);
```

- Matriz cujos elementos correspondem à soma dos índices linha e coluna, $A = \{a_{ij}\}$ com $a_{ij} = i + j$:

```
#define GMATRIX_SUMLINCOL(Mat) PGMATRIX_SUMLINCOL(&Mat)
void PGMATRIX_SUMLINCOL(PGMATRIX pMat);
```

- Matriz com 1s em todos os elementos, $A = \{a_{ij}\}$ com $a_{ij} = 1$.

```
#define GMATRIX_ONES(Mat) PGMATRIX_ONES(&Mat)
void PGMATRIX_ONES(PGMATRIX pMat);
```

- Matriz de elementos aleatórios e independentes de distribuição uniforme no intervalo $[0, 1]$, $A = \{a_{ij}\}$ com $a_{ij} \sim U(0, 1)$:

```
#define GMATRIX_RAND(Mat) PGMATRIX_RAND(&Mat)
void PGMATRIX_RAND(PGMATRIX pMat);
```

De fato, cada elemento da matriz é obtido por `rand()/RAND_MAX`, de forma que grau de independência entre as v.a.s e sua distribuição dependem de como a função `rand()` foi implementada.

- Matriz de elementos aleatórios de distribuição Gaussiana de média nula e variância unitária, $A = \{a_{ij}\}$ com $a_{ij} \sim N(0, 1)$:

```
#define GMATRIX_RANDN(Mat) PGMATRIX_RANDN(&Mat)
void PGMATRIX_RANDN(PGMATRIX pMat);
```

Cada elemento da matriz é obtido por $a_{ij} = \sum_{k=1}^{12}(\texttt{rand()/RAND\_MAX} - 0,5)$, de forma que o grau de independência entre os elementos e sua distribuição dependem de como a função `rand()` foi implementada. Esta forma de geração de números aleatórios de distribuição Gaussiana é baseada no teorema do limite central [Papoulis e Pillai 2001], e foi utilizada pelo MATLAB em suas primeiras versões. No entanto, a qualidade da distribuição depende fortemente da função `rand()`. Para aplicações que requerem melhores geradores de números aleatórios Gaussianos, sugere-se implementar outros métodos [Press et al. 1992].

- Matriz identidade: $A = \{a_{ij}\}$ com $a_{ij} = 1$ para $i = j$, e $a_{ij} = 0$ para $i \neq j$:

```
#define GMATRIX_IDENTITY(Mat) PGMATRIX_IDENTITY(&Mat)
void PGMATRIX_IDENTITY(PGMATRIX pMat);
```

- Matriz de Wilkinson, simétrica, tridiagonal matriz com número ímpar de colunas. Usada no teste de algoritmos de cálculo de auto-valores:

```
#define GMATRIX_WILKINSON(Mat) PGMATRIX_WILKINSON(&Mat)
void PGMATRIX_WILKINSON(PGMATRIX pMat);
```

- Matrizes ortogonais:

```
#define GMATRIX_ORTHOGONAL(Mat,Type) PGMATRIX_ORTHOGONAL(&Mat,Type)
void PGMATRIX_ORTHOGONAL(PGMATRIX pMat,int Type);
```

De acordo com o valor da variável `Type`, que podem ser pode ser -2, -1, 1 ou 2, diferentes matrizes quadradas ortogonais são geradas:

`Type` = 1: $A = \{a_{ij}\}$ com $a_{ij} = \sqrt{\frac{2}{N+1}} \sin\left(\frac{ij\pi}{N+1}\right)$

`Type` = 2: $A = \{a_{ij}\}$ com $a_{ij} = \frac{2}{\sqrt{N+1}} \sin\left(\frac{2ij\pi}{2N+1}\right)$

`Type` = -1: $A = \{a_{ij}\}$ com $a_{ij} = \cos\left(\frac{(i-1)(j-1)\pi}{N-1}\right)$

`Type` = -2: $A = \{a_{ij}\}$ com $a_{ij} = \cos\left(\frac{(i-1)(j-0{,}5)\pi}{N}\right)$

3.3 Informações sobre matrizes

O conteúdo de uma matriz pode ser impressa na tela utilizando os comandos a seguir. Este comandos fazem uso da macro `GMATRIX_PRINTCOMMAND` que contém o comando de impressão. Por default, esta macro contém `printf`. Se usuário desejar alterar o conteúdo, deve definr a variável no seu arquivo fonte antes de `#include ''gmatrix.h''`. O usuário pode também alterar o valor default de `GMATRIX_PRINTCOMMAND` diretamente em `gmatrix.h`.

Os comandos a seguir servem para apresentar o conteúdo de uma matriz `Mat`. Deve ser observado que `#Mat` é uma cadeia de caracteres (string) com o nome da variável (MSVC++ 6.0).

```
#define GMATRIX_PRINT(Mat) PGMATRIX_PRINT_NAMED(#Mat,&Mat)
#define PGMATRIX_PRINT(pMat) PGMATRIX_PRINT_NAMED(#pMat,pMat)
#define GMATRIX_PRINT_MATLABFORM(Mat) PGMATRIX_PRINT_MATLABFORM_NAMED(#Mat,&Mat)
#define PGMATRIX_PRINT_MATLABFORM(pMat) PGMATRIX_PRINT_MATLABFORM_NAMED(#pMat,pMat)
#define GMATRIX_PRINTROW(Mat,i) PGMATRIX_PRINTROW_NAMED(#Mat,&Mat,i)
#define PGMATRIX_PRINTROW(pMat,i) PGMATRIX_PRINTROW_NAMED(#pMat,pMat,i)
#define GMATRIX_PRINT_EXP(Mat) PGMATRIX_PRINT_EXP_NAMED(#Mat,&Mat)
#define PGMATRIX_PRINT_EXP(pMat) PGMATRIX_PRINT_EXP_NAMED(#pMat,pMat)
```

Para visualizar o número de linhas, colunas e `MaxSize` de uma matriz, sugere-se o uso do comando

```
#define GMATRIX_INFO(Mat) PGMATRIX_INFO_NAMED(#Mat,&Mat)
#define PGMATRIX_INFO(pMat) PGMATRIX_INFO_NAMED(#pMat,pMat)
```

3.4 Operações elementares

3.4.1 Cópia de matrizes

Para cópia de uma matriz `Mat` em `MatResult`, a seguite função pode ser usada:

```
#define GMATRIX_COPY(MatResult, Mat) PGMATRIX_COPY(&MatResult, &Mat)
void PGMATRIX_COPY(PGMATRIX pMatResult, PGMATRIX pMat);
```

O cuidado que deve-se ter com relação a `MatResult` é que suas dimensões são ajustadas para serem as mesmas de `Mat`. No mais, `MatResult` deve ser alocada previamente, e `pMatResult->MaxSize` deve ser grande o suficiente para que a matriz comporte todos os dados.

Para copiar uma coluna inteira de matriz `Mat` em uma outra coluna de `MatResult`, a função abaixo pode ser usada:

```
#define GMATRIX_COPY_COLUMN(MatResult, ColDest, Mat, ColOrigin) \
          PGMATRIX_COPY_COLUMN(&MatResult, ColDest, &Mat, ColOrigin)
void PGMATRIX_COPY_COLUMN(PGMATRIX pMatResult, int ColDest, PGMATRIX pMat, int ColOrigin);
```

Nesta função, a coluna de índice `ColOrigin` da matrix `Mat` é copiada na coluna `ColDest` de `MatResult`. As dimensões das matrizes devem ser apropriadas. De forma similar, `GMATRIX_COPY_ROW` pode ser usada para copiar linhas:

```
#define GMATRIX_COPY_ROW(MatResult, RowDest, Mat, RowOrigin) \
          PGMATRIX_COPY_ROW(&MatResult, RowDest, &Mat, RowOrigin)
void PGMATRIX_COPY_ROW(PGMATRIX pMatResult, int RowDest, PGMATRIX pMat, int RowOrigin);
```

3.4.2 Transposta

Através da função abaixo, obtem-se `MatTranspose` = $\texttt{Mat}^T$.

```
#define GMATRIX_TRANSPOSE_COPY(MatTranspose, Mat) \
          PGMATRIX_TRANSPOSE_COPY(&MatTranspose, &Mat)
void PGMATRIX_TRANSPOSE_COPY(PGMATRIX pMatTranspose, PGMATRIX pMat);
```

3.4.3 Comutação de linhas e colunas

Para a comutação de linhas e colunas de matrizes, o usuário poderá faszer uso de

```
#define GMATRIX_SWAP_ROW(Mat,i,j) PGMATRIX_SWAP_ROW(&Mat,i,j)
void PGMATRIX_SWAP_ROW(PGMATRIX pMat,int i, int j);
#define GMATRIX_SWAP_COLUMN(Mat,i,j) PGMATRIX_SWAP_COLUMN(&Mat,i,j)
void PGMATRIX_SWAP_COLUMN(PGMATRIX pMat,int i, int j);
```

Por exemplo, considere o seginte código:

```
GMATRIX_SWAP_ROW(MatA,2,4);
GMATRIX_SWAP_COLUMN(MatB,1,3);
```

Após sua execução as linhas 2 e 4 de `MatA` estarão trocadas. O mesmo acontece com a colunas 1 e 3 de `MatB`.

3.4.4 Operações entre matrizes e constantes

Para realizar as operações `Mat = Mat+Value` (que no estilo Matlab significa somar o escalar `Value` a todos os elementos da matriz `Mat`) e `Mat = Mat*Value`, pode-se lançar mão respectivamente das funções

```
#define GMATRIX_ADD_CONST(Mat,Value) PGMATRIX_ADD_CONST(&Mat,Value)
void PGMATRIX_ADD_CONST(PGMATRIX pMat,double Value);
#define GMATRIX_MULTIPLY_CONST(Mat,Value) PGMATRIX_MULTIPLY_CONST(&Mat,Value)
void PGMATRIX_MULTIPLY_CONST(PGMATRIX pMat,double Value);
```

Por outro lado, obtem-se `MatResult = Mat*Value` e `MatResult = MatResult + (Mat*Value)`, respectivamente, por meio de

```
#define GMATRIX_MULTIPLY_CONST_COPY(MatResult, Mat,Value) \
          PGMATRIX_MULTIPLY_CONST_COPY(&MatResult, &Mat,Value)
void PGMATRIX_MULTIPLY_CONST_COPY(PGMATRIX pMatResult, PGMATRIX pMat,double Value);
#define GMATRIX_MULTIPLY_CONST_ADD(MatResult, Mat,Value) \
          PGMATRIX_MULTIPLY_CONST_ADD(&MatResult, &Mat,Value)
void PGMATRIX_MULTIPLY_CONST_ADD(PGMATRIX pMatResult, PGMATRIX pMat,double Value);
```

3.4.5 Soma/Subtração de matrizes

Para lidar com soma e subtração de matrizes (e casos especiais), a biblioteca `GMATRIX` dispõe das seguintes funções:

```
#define GMATRIX_ADD_COPY(MatResult, MatA, MatB) \
          PGMATRIX_ADD_COPY(&MatResult, &MatA, &MatB)
void PGMATRIX_ADD_COPY(PGMATRIX pMatResult, PGMATRIX pMatA, PGMATRIX pMatB);
#define GMATRIX_ADD(MatA, MatB) PGMATRIX_ADD(&MatA, &MatB)
void PGMATRIX_ADD(PGMATRIX pMatA, PGMATRIX pMatB);
#define GMATRIX_SUBSTRACT_COPY(MatResult, MatA, MatB) \
          PGMATRIX_SUBSTRACT_COPY(&MatResult, &MatA, &MatB)
void PGMATRIX_SUBSTRACT_COPY(PGMATRIX pMatResult, PGMATRIX pMatA, PGMATRIX pMatB);
#define GMATRIX_SUBSTRACT_IDENTITY_COPY(MatResult, Mat) \
          PGMATRIX_SUBSTRACT_IDENTITY_COPY(&MatResult, &Mat)
void PGMATRIX_SUBSTRACT_IDENTITY_COPY(PGMATRIX pMatResult, PGMATRIX pMat);
#define GMATRIX_SUBSTRACT_IDENTITY(Mat) PGMATRIX_SUBSTRACT_IDENTITY(&Mat)
void PGMATRIX_SUBSTRACT_IDENTITY(PGMATRIX pMat);
#define GMATRIX_SUBSTRACT(MatA, MatB) PGMATRIX_SUBSTRACT(&MatA, &MatB)
void PGMATRIX_SUBSTRACT(PGMATRIX pMatA, PGMATRIX pMatB);
```

Os exemplos a seguir são auto-explicativos, com `I` sendo a matriz identidade:

```
GMATRIX_ADD_COPY(MatC, MatA, MatB); // MarC = MatA + MatB.
GMATRIX_ADD(MatA, MatB); // MatA = MatA + MatB.
GMATRIX_SUBSTRACT_COPY(MatC, MatA, MatB); // MatC = MatA - MatB.
GMATRIX_SUBSTRACT_IDENTITY_COPY(MatC, MatA); // MatC = MatA - I.
GMATRIX_SUBSTRACT_IDENTITY(MatA); // MatA = MatA - I.
GMATRIX_SUBSTRACT(MatA, MatB); // MatA = MatA - MatB.
```

3.4.6 Submatrizes

`GMATRIX` possui funções para manipular partes de matrizes (ou submatrizes). Seus protótiposd são apresentados abaixo:

```
#define GMATRIX_SUBMATRIX_COPY(Mat,nl,nc,MatOrigin) \
          PGMATRIX_SUBMATRIX_COPY(&Mat,nl,nc,&MatOrigin);
void PGMATRIX_SUBMATRIX_COPY(PGMATRIX pMat, int nl, int nc, PGMATRIX pMatOrigin);
#define GMATRIX_SUBMATRIX_ADD(Mat,nl,nc,MatOrigin) \
          PGMATRIX_SUBMATRIX_ADD(&Mat,nl,nc,&MatOrigin);
void PGMATRIX_SUBMATRIX_ADD(PGMATRIX pMat, int nl, int nc, PGMATRIX pMatOrigin);
#define GMATRIX_SUBMATRIX_COPY_EXTENDED(Mat,nl,nc,MatOrigin,FlagTransposeOrigin) \
          PGMATRIX_SUBMATRIX_COPY_EXTENDED(&Mat,nl,nc,&MatOrigin,FlagTransposeOrigin);
void PGMATRIX_SUBMATRIX_COPY_EXTENDED(PGMATRIX pMat,
          int nl, int nc, PGMATRIX pMatOrigin, int FlagTransposeOrigin);
#define GMATRIX_SUBMATRIX_FILL(Mat,nlb,nle,ncb,nce,Value) \
          PGMATRIX_SUBMATRIX_FILL(&Mat,nlb,nle,ncb,nce,Value)
void PGMATRIX_SUBMATRIX_FILL(PGMATRIX pMat,int nlb,int nle,int ncb,int nce,double Value);
#define GMATRIX_SUBMATRIX_MULTIPLY_CONST(Mat,nlb,nle,ncb,nce,Value) \
          PGMATRIX_SUBMATRIX_MULTIPLY_CONST(&Mat,nlb,nle,ncb,nce,Value)
void PGMATRIX_SUBMATRIX_MULTIPLY_CONST(PGMATRIX pMat,
          int nlb,int nle,int ncb,int nce,double Value);
```

A tabela seguinte apresenta códigos em C e seu correspondente MATLAB.

Código C	Significado (MATLAB)
`GMATRIX_SUBMATRIX_COPY(MatA,r,c,MatB);`	`MatA(r:MatB.Nr,c:MatB.Nc) = MatB`
`GMATRIX_SUBMATRIX_ADD(MatA,r,c,MatB);`	`MatA(r:MatB.Nr,c:MatB.Nc) = MatA(2:MatB.Nr,3:MatB.Nc) + MatB`
`GMATRIX_SUBMATRIX_COPY_EXTENDED(MatA,r,c,MatB,0)`	`MatA(r:MatB.Nr,c:MatB.Nc) = MatB`
`GMATRIX_SUBMATRIX_COPY_EXTENDED(MatA,r,c,MatB,1)`	`MatA(r:MatB.Nr,c:MatB.Nc) = MatB'`
`MATRIX_SUBMATRIX_FILL(Mat,a,b,c,d,12.5);`	`MatA(a:b,c:d) = 12.5`
`GMATRIX_SUBMATRIX_MULTIPLY_CONST(Mat,a,b,c,d,12.5);`	`MatA(a:b,c:d) = MatA(a:b,c:d)*12.5`

3.5 Álgebra linear

3.5.1 Traço de matrizes quadradas e soma de elementos

```
#define GMATRIX_TRACE(Mat) PGMATRIX_TRACE(&Mat)
double PGMATRIX_TRACE(PGMATRIX pMat);
#define GMATRIX_SUMENTRIES(Mat) PGMATRIX_SUMENTRIES(&Mat)
double PGMATRIX_SUMENTRIES(PGMATRIX pMat);
#define GMATRIX_SUMABSOLUTEENTRIES(Mat) PGMATRIX_SUMABSOLUTEENTRIES(&Mat)
double PGMATRIX_SUMABSOLUTEENTRIES(PGMATRIX pMat);
```

Os exemplos a seguir são auto-explicativos:

```
a = GMATRIX_TRACE(MatA); // a = soma dos elementos da diagonal de MatA.
b = GMATRIX_SUMENTRIES(MatB); // b = soma de todos os elementos de MatB.
c = GMATRIX_SUMABSOLUTEENTRIES(MatC); // c = soma dos valores absolutos dos elementos de MatC.
```

3.5.2 Decomposição LU

Na decomposição LU, a partir de uma matriz **A** obtem-se matrizes **L** e **U** tais que $\mathbf{A} = \mathbf{L} \cdot \mathbf{U}$. **L** é triangular inferior e **U** triangular superior. Decomposição LU pode ser usada na resolução mais eficiente de sistemas de equações [Press et al. 1992]. O protótipo da função de `GMATRIX` que realiza decomposição LU é

```
#define GMATRIX_LUDCMP(MatLU, Mat) PGMATRIX_LUDCMP(&MatLU, &Mat)
double PGMATRIX_LUDCMP(PGMATRIX pMatLU, PGMATRIX pMat);
```

A implementação é praticamente a mesma de [Press et al. 1992], com algumas adaptações. O resultado da decomposição da matriz `Mat` é armazenado em `MatLU`, com a seguinte observação: se **L** e **U** são dadas por

$$\mathbf{L} = \begin{bmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} \end{bmatrix} \qquad \mathbf{U} = \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{bmatrix} \tag{2}$$

os elementos da diagonal de L não são copiados em `MatLU`:

$$\texttt{MatLU} = \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ l_{21} & u_{22} & u_{23} & u_{24} \\ l_{31} & l_{32} & u_{33} & u_{34} \\ l_{41} & l_{42} & l_{43} & u_{44} \end{bmatrix} \tag{3}$$

Conforme [Press et al. 1992], decomposição LU também pode ser usada na inversão eficiente de matrizes.

3.5.3 Eliminação de Gauss-Jordan

```
#define GMATRIX_GAUSSJORDAN(AInv, X, A, B) PGMATRIX_GAUSSJORDAN(&AInv, &X, &A, &B)
void PGMATRIX_GAUSSJORDAN(PGMATRIX pAInverse, PGMATRIX pX, PGMATRIX pA, PGMATRIX pB);
```

Considerando matrizes $\mathbf{A}$, $\mathbf{X}$ e $\mathbf{B}$ tais que

$$\mathbf{A} \cdot \mathbf{X} = \mathbf{B} \tag{4}$$

então `GMATRIX_GAUSSJORDAN(AInv, X, A, B)` determina `Ainv` = $\mathbf{A}^{-1}$ e `X` = $\mathbf{A}^{-1}\mathbf{B}$. Esta função pode ser usada paa determinar a inversa de uma matriz $\mathbf{A}$ (neste caso $\mathbf{B}$ seria identidade) ou resolver um conjunto de equações [Press et al. 1992]. Esta função realiza inversão matricial se for usado `GMATRIX_GAUSSJORDAN(AInv, NULL, A, NULL)`. No entanto, apesar de existirem soluções mais eficientes para inversão matricial, eliminação de Gauss-Jordam é mais estável numericamente [Press et al. 1992]. Sua implementação usa pivotação, o que o torna mais estável.

3.5.4 Decomposição de Cholesky

```
#define GMATRIX_CHOLESKY(L, A) PGMATRIX_CHOLESKY(&L, &A)
void PGMATRIX_CHOLESKY(PGMATRIX pL, PGMATRIX pA);
```

Na decomposição de Cholesky, a partir de uma matriz $\mathbf{A}$ determina-se uma matriz $\mathbf{L}$ (triangular inferior) tal que $\mathbf{A} = \mathbf{L} \cdot \mathbf{L}^T$. Não deve ser confundido com raiz quadrada de matrizes.

3.5.5 Determinante

As seguintes funções retornam um `double` que é o determinante de uma matriz `Mat`:

```
#define GMATRIX_DETERMINANT2(Mat) PGMATRIX_DETERMINANT2(&Mat)
double PGMATRIX_DETERMINANT2(PGMATRIX pMat);
#define GMATRIX_DETERMINANT3(Mat) PGMATRIX_DETERMINANT3(&Mat)
double PGMATRIX_DETERMINANT3(PGMATRIX pMat);
#define GMATRIX_DETERMINANT(Mat, MatDummy) PGMATRIX_DETERMINANT(&Mat, &MatDummy)
double PGMATRIX_DETERMINANT(PGMATRIX pMat, PGMATRIX pMatDummy);
```

As funções `GMATRIX_DETERMINANT2` e `GMATRIX_DETERMINANT3` são específicas para matrizes quadradas de dimensões 2 e 3, respectivamente. Estas funções realizam o cálculo direto do determinante.

Para matrizes de maior dimensão, deve-se usar `GMATRIX_DETERMINANT`. Esta função faz uso de `GMATRIX_LUDCMP`, de decomposição LU, cujo produto dos elementos da diagonal de (3) é o determinante desejado.

3.5.6 Inversão matricial

```
#define GMATRIX_INVERSE2_COPY(MatInverse, Mat) PGMATRIX_INVERSE2_COPY(&MatInverse, &Mat)
void PGMATRIX_INVERSE2_COPY(PGMATRIX pMatInverse, PGMATRIX pMat);
#define GMATRIX_INVERSE3_COPY(MatInverse, Mat) PGMATRIX_INVERSE3_COPY(&MatInverse, &Mat)
void PGMATRIX_INVERSE3_COPY(PGMATRIX pMatInverse, PGMATRIX pMat);
#define GMATRIX_INVERSE_COPY(MatInverse, Mat) PGMATRIX_INVERSE_COPY(&MatInverse, &Mat)
void PGMATRIX_INVERSE_COPY(PGMATRIX pMatInverse, PGMATRIX pMat);
#define GMATRIX_INVERSE(Mat) PGMATRIX_INVERSE(&Mat)
void PGMATRIX_INVERSE(PGMATRIX pMat);
```

As funções `GMATRIX_INVERSE2_COPY` e `GMATRIX_INVERSE3_COPY` são específicas para matrizes quadradas de dimensões 2 e 3, respectivamente. Estas funções realizam o cálculo direto da inversa matricial. Elas podem ser mais rápidas do que soluções iterativas pois usam a relação

$$\mathbf{A}^{-1} = \frac{adj(\mathbf{A})}{\det(\mathbf{A})} \tag{5}$$

Se det($\mathbf{A}$) for muito pequeno, então estas funções utilizam uma solução mais estável numericamente (`GMATRIX_GAUSSJORDAN(AInv, NULL, A, NULL)`).

Para matrizes de maior dimensão, deve-se usar `GMATRIX_INVERSE_COPY` ou `GMATRIX_INVERSE`. Esta função faz uso de `GMATRIX_GAUSSJORDAN(AInv, NULL, A, NULL)`, que é mais estável numericamente (seção 3.5.3).

Alguns exemplos são mostrados na tabela seguinte.

Código C	**Significado (MATLAB)**

`GMATRIX_INVERSE2_COPY(MatB, MatA);`
`GMATRIX_INVERSE3_COPY(MatB, MatA);`
`GMATRIX_INVERSE_COPY(MatB, MatA);`
	`MatB = inv(MatA)`

`GMATRIX_INVERSE(MatA);`	`MatA = inv(MatA)`

3.5.7 Decomposição em valores singulares

```
#define GMATRIX_SVD(U,S,V,Mat,FlagSorted) PGMATRIX_SVD(&U,&S,&V,&Mat,FlagSorted)
void PGMATRIX_SVD(PGMATRIX pU,
          PGMATRIX pS,PGMATRIX pV,PGMATRIX pMat, unsigned char FlagSorted);
```

A decomposição em valores singulares de uma matriz $\mathbf{A}$ é dada por

$$\mathbf{A} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T \tag{6}$$

em que $\mathbf{U}$ é ortogonal-coluna, $\mathbf{V}$ é ortogonal e $\mathbf{S}$ é diagonal. Os elementos na diagonal de $\mathbf{S}$ são chamados de valores singulares da matriz $\mathbf{A}$, e são valores maiores ou iguais a zero. No caso de `GMATRIX_SVD`, `Mat` é a matriz que será decomposta e se `FlagSorted` $\neq 0$, então os elementos s_{ii} da diagonal de $\mathbf{S}$ estarão organizados em ordem descrescente: $s_{11} \geq s_{22} \geq \cdots \geq s_{cc}$, com c = `Mat.Nc`. Decomposição em valores singulares encontra inúmeras aplicações em teoria da estimação e álgebra matricial [Press et al. 1992].

3.5.8 Pseudo-Inversa

```
#define GMATRIX_PSEUDOINVERSE(Apinv,A,MatDummy) \
          PGMATRIX_PSEUDOINVERSE(&Apinv,&A,&MatDummy)
void PGMATRIX_PSEUDOINVERSE(PGMATRIX pApinv, PGMATRIX pA, PGMATRIX pMatDummy);
#define GMATRIX_LEFT_PSEUDOINVERSE_COPY(Apinv,A,MatDummy) \
          PGMATRIX_LEFT_PSEUDOINVERSE_COPY(&Apinv,&A,&MatDummy)
void PGMATRIX_LEFT_PSEUDOINVERSE_COPY(PGMATRIX pApinv, PGMATRIX pA, PGMATRIX pMatDummy);
#define GMATRIX_DERIVATIVE_LEFT_PSEUDOINVERSE(dApinvdx,dAdx,A,Apinv,MatDummy1,MatDummy2) \
          PGMATRIX_DERIVATIVE_LEFT_PSEUDOINVERSE(&dApinvdx,
          &dAdx,&A,&Apinv,&MatDummy1,&MatDummy2)
void PGMATRIX_DERIVATIVE_LEFT_PSEUDOINVERSE(PGMATRIX pdApinvdx,
          PGMATRIX pdAdx, PGMATRIX pA, PGMATRIX pApinv,
          PGMATRIX pMatDummy1, PGMATRIX pMatDummy2);
```

Dado o seguinte conjunto de equações

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \tag{7}$$

sua solução é dada por $\mathbf{x} = \mathbf{A}^{\dagger} \cdot \mathbf{b}$ em que $\mathbf{A}^{\dagger} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$ é chamada de pseudo-inversa (à esquerda) de Moore-Penrose da matriz $\mathbf{A}$. Se $\mathbf{A}$ não for de posto completo, $\mathbf{A}^T\mathbf{A}$ é singular. No entanto, uma aproximação no sentido de mínimos quadrados pode ser obtida por meio de decomposição em valores

singulares. De fato, este método é usado pela função `GMATRIX_PSEUDOINVERSE`. Desta forma, sendo $\mathbf{A} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T$, então `Apinv` $= \mathbf{A}^\dagger = \mathbf{V} \cdot \mathbf{D} \cdot \mathbf{U}^T$, em que $\mathbf{D} = \{d_{ij}\}$ é uma matriz diagonal com

$$d_{ii} = \begin{cases} s_{ii}^{-1} & s_{ii} > \eta \\ 0 & s_{ii} \leq \eta \end{cases} \tag{8}$$

em que $\eta = \max(N_r, N_c) \cdot \max(s_{11}, s_{22}, s_{33}, \cdots) \cdot EPS$, com $\dim(\mathbf{A}) = \{N_r, N_c\}$ e EPS sendo a precisão relativa para ponto flutuante (*i.e.*, mínimo valor EPS tal que $1 + EPS = 1$ na representação numérica em uso). Na implementação atual, $EPS = 10^{-12}$. Se a matriz $\mathbf{A}$ for quadrada e não singular, então `Apinv` é sua inversa. Se for usado `PGMATRIX_PSEUDOINVERSE` com `pApinv = NULL`, então a pseudo-inversa será copiada em `pA`, que terá suas dimensões devidamente alteradas. `MatDummy` é uma matriz auxiliar, que deve estar devidamente alocada antes da chamada de `GMATRIX_PSEUDOINVERSE` ou `PGMATRIX_PSEUDOINVERSE`. Suas dimensões devem ser as mesmas da matriz $\mathbf{A}$.

O cálculo de pseudo-inversas usando `GMATRIX_PSEUDOINVERSE` envolve um custo computacional considerável quando comparado ao cálculo direto de $\mathbf{A}^\dagger = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$. De fato, houver certeza de que $\mathbf{A}^T\mathbf{A}$ é não singular, então a função `GMATRIX_LEFT_PSEUDOINVERSE_COPY` pode ser usada.

Se for desejado determinar a derivada de $\mathbf{A}^\dagger$ com relação a uma variável escalar x, esta é dada por

$$\frac{d(\mathbf{A}^\dagger)}{dx} = -(\mathbf{A}^T\mathbf{A})^{-1} \cdot \left(\frac{d\mathbf{A}}{dx}\right)^T \cdot \left(\mathbf{A}\mathbf{A}^\dagger - \mathbf{I}\right) - \mathbf{A}^\dagger \cdot \frac{d\mathbf{A}}{dx} \cdot \mathbf{A}^\dagger \tag{9}$$

A implementação de $d(\mathbf{A}^\dagger)/dx$ é feita pela função `GMATRIX_DERIVATIVE_LEFT_PSEUDOINVERSE`. Como argumentos, além de `Apinv` $= \mathbf{A}^\dagger$, o usuário precisa passar `dAdx`$= \frac{d\mathbf{A}}{dx}$ e as matrizes auxiliares `MatDummy1` e `MatDummy2`, ambas com as mesmas dimensões da matriz $\mathbf{A}$.

3.5.9 Posto de matrizes

```
#define GMATRIX_RANK_FROMSVD(U,S,V) PGMATRIX_RANK_FROMSVD(&U,&S,&V)
int PGMATRIX_RANK_FROMSVD(PGMATRIX pU, PGMATRIX pS, PGMATRIX pV);
#define GMATRIX_RANK(Mat) PGMATRIX_RANK(&Mat)
int PGMATRIX_RANK(PGMATRIX pMat);
```

As funções `GMATRIX_RANK` e `GMATRIX_RANK_FROMSVD` são equivalentes no sentido que usam decomposição em valores singulares para determinar o posto de uma matrix `Mat`. Ambas retornam um número inteiro, que é o posto. Nestas implementações, o posto é determinado como sendo o número de valores singulares da matriz `Mat` que são maiores que η (ver seção 3.5.8). Se em uma determinada implementação as decomposição em valores singulares da matriz `Mat` estiver disponível (e.g., se ela for indispensável para o cálculo em questão), então o usuário pode fazer uso de `GMATRIX_RANK_FROMSVD`, que já parte das matrizes da decomposição. Por outro lado, `GMATRIX_RANK` realiza a decomposição para então retornar o posto.

3.6 Estatística

```
#define GMATRIX_MEAN(Mat) PGMATRIX_MEAN(&Mat)
double PGMATRIX_MEAN(PGMATRIX pMat);
#define GMATRIX_VARIANCE(Mat) PGMATRIX_VARIANCE(&Mat)
double GMATRIX_VARIANCE(PGMATRIX pMat);
```

Estas funções determinam simplesmente o valor médio μ e a variância σ^2 dos elementos da matriz `Mat`. Estas medidas são calculadas como

$$\mu = \frac{1}{(\texttt{Mat.Nr}) \cdot (\texttt{Mat.Nc})} \sum_{i=1}^{\texttt{Mat.Nr}} \sum_{j=1}^{\texttt{Mat.Nj}} a_{ij} \tag{10}$$

$$\sigma^2 = \frac{1}{(\texttt{Mat.Nr}) \cdot (\texttt{Mat.Nc})} \sum_{i=1}^{\texttt{Mat.Nr}} \sum_{j=1}^{\texttt{Mat.Nj}} (a_{ij} - \mu)^2 \tag{11}$$

em que a_{ij} são elementos da matriz `Mat`. A medida de variância é polarizada [Papoulis e Pillai 2001]. Para obter a variância não polarizada, basta multiplicar o resultado por uma constante [Papoulis e Pillai 2001].

3.7 Operações especiais

3.7.1 Multiplicação de três matrizes

```
#define GMATRIX_TRIPLEMULTIPLY_COPY(MatResult, MatA, MatB, MatC, MatDummy) \
          PGMATRIX_TRIPLEMULTIPLY_COPY(&MatResult, &MatA, &MatB, &MatC, &MatDummy)
void PGMATRIX_TRIPLEMULTIPLY_COPY(PGMATRIX pMatResult,
          PGMATRIX pMatA, PGMATRIX pMatB, PGMATRIX pMatC, PGMATRIX pMatDummy) ;
#define GMATRIX_TRIPLEMULTIPLY_ADD(MatResult, MatA, MatB, MatC, MatDummy) \
          PGMATRIX_TRIPLEMULTIPLY_ADD(&MatResult, &MatA, &MatB, &MatC, &MatDummy)
void PGMATRIX_TRIPLEMULTIPLY_ADD(PGMATRIX pMatResult,
          PGMATRIX pMatA, PGMATRIX pMatB, PGMATRIX pMatC, PGMATRIX pMatDummy) ;
#define GMATRIX_TRIPLEMULTIPLY_COPY_EXTENDED(MatResult,
          MatA, FlagTransposeA,
          MatB, FlagTransposeB,
          MatC, FlagTransposeC, MatDummy) \
          PGMATRIX_TRIPLEMULTIPLY_COPY_EXTENDED(&MatResult,
          &MatA, FlagTransposeA,
          &MatB, FlagTransposeB,
          &MatC, FlagTransposeC, &MatDummy)
void PGMATRIX_TRIPLEMULTIPLY_COPY_EXTENDED(PGMATRIX pMatResult,
          PGMATRIX pMatA, BOOL FlagTransposeA, PGMATRIX pMatB, BOOL FlagTransposeB,
          PGMATRIX pMatC, BOOL FlagTransposeC, PGMATRIX pMatDummy) ;
```

`GMATRIX_TRIPLEMULTIPLY_COPY` e `GMATRIX_TRIPLEMULTIPLY_ADD` realizam a multiplicação de três matrizes, que podem estar repetidas, uma vez que as matrizes originais não são alteradas. Com `GMATRIX_TRIPLEMULTIPLY_COPY_EXTENDED` uma variável associada a cada matriz indica se ela está na multiplicação como transposta ou não. A matriz resultante da operação não pode ser nenhuma das matrizes que compõem a multiplicação. Alguns exemplos são mostrados na tabela seguinte.

Código C	**Significado (MATLAB)**
`GMATRIX_TRIPLEMULTIPLY_COPY(D, A, B, C, MatDummy);`	`D = A*B*C`
`GMATRIX_TRIPLEMULTIPLY_COPY_EXTENDED(D,A,0,B,1,C,1,MatDummy);`	`D = A*(B')*(C')`
`GMATRIX_TRIPLEMULTIPLY_COPY_EXTENDED(D,A,0,P,0,A,1,MatDummy);`	`D = A*P*(A')`
`GMATRIX_TRIPLEMULTIPLY_ADD(D, A, B, C, MatDummy);`	`D = D + A*B*C`

3.7.2 Propagação de matriz de covariâncias

```
#define GMATRIX_COVARIANCE_PROPAGATION_COPY(MatCovResult, MatGain, MatCov, MatDummy) \
          PGMATRIX_COVARIANCE_PROPAGATION_COPY(&MatCovResult, &MatGain, &MatCov, &MatDummy)
void PGMATRIX_COVARIANCE_PROPAGATION_COPY(PGMATRIX pMatCovResult, PGMATRIX pMatGain,
          PGMATRIX pMatA, PGMATRIX pMatB, PGMATRIX pMatC, PGMATRIX pMatDummy) ;
          PGMATRIX pMatCov, PGMATRIX pMatDummy);
#define GMATRIX_COVARIANCE_PROPAGATION_ADD(MatCovResult, MatGain, MatCov, MatDummy) \
          PGMATRIX_COVARIANCE_PROPAGATION_ADD(&MatCovResult, &MatGain, &MatCov, &MatDummy)
void PGMATRIX_COVARIANCE_PROPAGATION_ADD(PGMATRIX pMatCovResult, PGMATRIX pMatGain,
          PGMATRIX pMatCov, PGMATRIX pMatDummy);
```

Se X é um vetor de variáveis aleatórias, e $Y = \mathbf{A} \cdot X + \mathbf{b}$ é uma transformação afim com $\mathbf{A}$ e $\mathbf{b}$ sendo matriz e vetor não aleatórios, então dado

$$\mathbf{P}_X = E\{(X - E\{X\})(X - E\{X\})^T\}, \tag{12}$$

a matriz de covariâncias de X, então a matriz de covariâncias de Y é dada por [Papoulis e Pillai 2001]

$$\mathbf{P}_Y = \mathbf{A} \cdot \mathbf{P}_X \cdot \mathbf{A}^T. \tag{13}$$

A função `GMATRIX_COVARIANCE_PROPAGATION_COPY` realiza o cálculo (13). Mais especificamente, `MatGain` $= \mathbf{A}$, `MatCov` $= \mathbf{P}_X$ e o resultado `MatCovResult` recebe $\mathbf{P}_Y$.

Por outro lado, `GMATRIX_COVARIANCE_PROPAGATION_ADD` adciona $\mathbf{P}_Y$ ao que conteúdo existente de `MatCovResult`. `MatDummy` deve ter dimensão suficientemente grande para armazenar $\mathbf{P}_X \cdot \mathbf{A}^T$.

Alguns exemplos são mostrados na tabela seguinte.

Código C	**Significado (MATLAB)**
`GMATRIX_COVARIANCE_PROPAGATION_COPY(Py, A, Px, MatDummy);`	`Py = A*Px*(A')`
`GMATRIX_COVARIANCE_PROPAGATION_ADD(Py, A, Px, MatDummy);`	`Py = Py + A*Px*(A')`

3.7.3 Distância de Mahalanobis

```
#define GMATRIX_MAHALANOBIS_DISTANCE(MatResidual, MatCovResidual, MatDummy1, MatDummy2) \
          PGMATRIX_MAHALANOBIS_DISTANCE(&MatResidual, &MatCovResidual, &MatDummy1, &MatDummy2)
double PGMATRIX_MAHALANOBIS_DISTANCE(PGMATRIX pMatResidual,
          PGMATRIX pMatCovResidual, PGMATRIX pMatDummy1, PGMATRIX pMatDummy2);
```

Sendo $X \sim N(\mathbf{x}, \mathbf{P}_X)$ e $Y \sim N(\mathbf{y}, \mathbf{P}_Y)$ vetores de variáveis aleatórias Gaussianas independentes, a distância de Mahalanobis é uma métrica comumente usada para avaliar a distância estatística entre elas. Assim, considerando $\boldsymbol{\varepsilon} = \mathbf{x} - \mathbf{y}$ o resíduo, sua matriz de covariâncias é dada por

$$\mathbf{P}_\varepsilon = \mathbf{P}_X + \mathbf{P}_Y, \tag{14}$$

e a distância de Mahalanobis entre X e Y é

$$q = \boldsymbol{\varepsilon}^T \cdot \mathbf{P}_\varepsilon^{-1} \cdot \boldsymbol{\varepsilon}. \tag{15}$$

Considerando este resultado, `GMATRIX_MAHALANOBIS_DISTANCE` retorna um `double` que é a distância q a partir de `MatResidual` $= \boldsymbol{\varepsilon}$ e `MatCovResidual` $= \mathbf{P}_\varepsilon$. `MatDummy1` e `MatDummy2` devem ter no mínimo as dimensões de $\mathbf{P}_\varepsilon$.

3.8 Gerenciamento de erros

A macro `GMATRIX_ABORTCOMMAND` é usada quando uma operação inválida é executada por funções da `GMATRIX`. De fato, as funções de `GMATRIX` utilizam uma macro `GMATRIX_ASSERT` para verificar incoerências nos parâmetros de entrada das funções (e.g., dimensões inapropriadas). Então, em caso de incoerência `GMATRIX_ASSERT` imprime uma mensagem indicando a incoerência e realiza o comando `GMATRIX_ABORTCOMMAND`. Por default, este comando é `exit(1)`, se `GMATRIX_ABORTCOMMAND` não for definido antes da inclusão de `gmatrix.h`. Se o usuário quiser um usar outro comando, basta definir `GMATRIX_ABORTCOMMAND` com o comando desejado.

3.9 Interface com MATLAB

3.9.1 Acesso a arquivos de dados MATLAB (.mat) (não concluído)

ATENÇÃO*: O texto abaixo é provisório, sendo que as funções ainda não estão completamente testadas.*

A biblioteca GMATRIX possui funções para acesso de escrita/leitura de arquivos .mat do MATLAB. Pode-se assim, por exemplo, salvar uma ou mais matrizes em um único arquivo `file.mat`, que pode ser lido pelo MATLAB. Também é possível ler variáveis a partir destes arquivos. Para tanto, é definida a seguinte estrutura auxiliar:

```
typedef struct{
    long type; /* Data type */
    long mrows; /* Number of rows */
    long ncols; /* Number of columns */
    long imagf; /* Imaginary flag */
    long namlen; /* Length of variable name */
} GMATRIX_MATLAB_DATAHEAD;
```

O formato em que os dados são estruturados em arquivos .mat segue a especificação do MATLAB 4.0, cuja compatibilidade vem sendo mantida pelas mais recentes versões do MATLAB. A estrutura `GMATRIX_MATLAB_DATAHEAD`

3.9.2 C-MEX

Para utilizar a biblioteca `GMATRIX` em uma função C-MEX, a macro `GMATRIX_CMEX` deve ser definida (ver topo de `gmatrix.h`). No mais, as seguintes funções são úteis para transferência de dados na `mexFunction`:

```
PGMATRIX PGMATRIX_ALLOC_FROM_MXARRAY(const mxArray *pmxarray);
mxArray *PGMATRIX_COPY_TO_MXARRAY(PGMATRIX pMat);
```

A primeira função, `PGMATRIX_ALLOC_FROM_MXARRAY`, aloca uma `GMATRIX` com o mesmo conteúdo da `mxarray` passada como argumento. Já com `PGMATRIX_COPY_TO_MXARRAY`, a `GMATRIX` apontada por `pMat` é copiada para uma `mxarray`. Neste caso não se faze necessário alocar a `mxarray` de saída, pois isto já é feito em `PGMATRIX_COPY_TO_MXARRAY`.

4 Ambientes de desenvolvimento

4.1 Microsoft Visual C++ 6.0

GMATRIX foi desenvolvida em programas criados no ambiente Microsoft Visual C++ 6.0. Portanto, seu uso neste ambiente é simples. Independente do tipo de projeto, sugere-se que o arquivo `gmatrix.c` seja incluído no mesmo. Cada módulo do projeto que fizer uso de uma função da biblioteca GMATRIX deverá incluir o cabeçalho `gmatrix.h`. Se o usuário desejar definir alguma macro que controla o modo de funcionamento de GMATRIX, deverá fazê-lo antes de incluir o cabeçalho `gmatrix.h`. Por exemplo,

```
#define GMATRIX_PRINTCOMMAND MeuPrintf
#include ''lib\gmatrix.h''
```

4.2 WinAVR - Microcontroladores ATMEGA

Por enquanto, foi verificada compatibilidade de trechos da biblioteca com o compilador GCC do ambiente WinAVR para microcontroladores ATMEGA8 da Atmel. No entanto, sendo estes dispositivos de memória bastante limitada quando comparado a microcomputadores compatíveis IBM-PC,

algumas macros permitem habilitar a inclusão apenas de trechos da biblioteca GMATRIX durante a compilação. Isto permite reduzir bastante o tamanho do código final, uma vez que o usuário pode escolher incluir na compilação apenas as funções necessárias.

Uma destas macros, `GMATRIX_DEFINE_ALL`, quando em `1`, resulta na inclusão de todas as funções da biblioteca. Com `GMATRIX_DEFINE_ALL` em `0`, outras macros fazem a seleção de trechos específicos, conforme definido pelo seguinte trecho de código de `gmatrix.h` :

```
#define GMATRIX_DEFINE_ALL 1
#define GMATRIX_DEFINE_ALLOC (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_SETSIZE (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_PRINT (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_PRINT_MATLABFORM (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_PRINTROW (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_PRINT_EXP (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_INFO (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_ZEROES (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_SUMLINCOL (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_ONES (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_RAND (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_RANDN (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_IDENTITY (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_WILKINSON (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_ORTHOGONAL (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_COPY (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_COPY_COLUMN (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_COPY_ROW (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_TRANSPOSE_COPY (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_TRACE (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_SUMENTRIES (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_SUMABSOLUTEENTRIES (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_LUDCMP (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_GAUSSJORDAN (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_CHOLESKY (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_DETERMINANT2 (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_DETERMINANT3 (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_DETERMINANT (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_INVERSE2_COPY (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_INVERSE3_COPY (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_INVERSE_COPY (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_INVERSE (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_SWAP_ROW (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_SWAP_COLUMN (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_SVD (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_ADD_CONST (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_MULTIPLY_CONST (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_MULTIPLY_CONST_COPY (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_MULTIPLY_CONST_ADD (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_ADD_COPY (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_ADD (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_SUBSTRACT_COPY (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_SUBSTRACT_IDENTITY_COPY (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_SUBSTRACT_IDENTITY (0 + GMATRIX_DEFINE_ALL)
```

```
#define GMATRIX_DEFINE_SUBSTRACT (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_MULTIPLY_COPY (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_MULTIPLY_COPY_EXTENDED (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_MULTIPLY_ADD (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_MULTIPLY_SUBSTRACT (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_TRIPLEMULTIPLY_COPY (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_TRIPLEMULTIPLY_COPY_EXTENDED (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_TRIPLEMULTIPLY_ADD (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_COVARIANCE_PROPAGATION_COPY (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_COVARIANCE_PROPAGATION_ADD (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_MAHALANOBIS_DISTANCE (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_SUBMATRIX_COPY (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_SUBMATRIX_ADD (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_SUBMATRIX_COPY_EXTENDED (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_SUBMATRIX_FILL (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_SUBMATRIX_MULTIPLY_CONST (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_PSEUDOINVERSE (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_LEFT_PSEUDOINVERSE_COPY (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_DERIVATIVE_LEFT_PSEUDOINVERSE (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_RANK_FROMSVD (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_RANK (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_MEAN (0 + GMATRIX_DEFINE_ALL)
#define GMATRIX_DEFINE_VARIANCE (0 + GMATRIX_DEFINE_ALL)
```

Assim, se for necessário fazer uso das funções `PGMATRIX_ALLOC` e `PGMATRIX_FREE` relacionadas à alocação dinâmica de memória, o usuário deve mudar o `#define` associado à macro `GMATRIX_DEFINE_ALLOC`, conforme indicado abaixo:

```
#define GMATRIX_DEFINE_ALLOC (1 + GMATRIX_DEFINE_ALL)
```

Para identificar que funções cada macro habilita, o usuário deverá consultar o código do arquivo `gmatrix.c`.

Um código exemplo do uso de GMATRIX com um ATMEGA é mostrado abaixo:

```
#include <avr/io.h>
#include <math.h>
#include <stdio.h>
#include ''gmatrix.h''

int main (void)
{
   // Testar inversa de matriz 3x3
   GMATRIX_DECLARE(Matrix, 3, 3);
   GMATRIX_DECLARE(MatrixInverse, 3, 3);

   // coloca PORTB como saida
   DDRB =0xFF;
   while (1){
      // Inicia a marix de forma aleatoria
      GMATRIX_RANDN(Matrix);
      // PORTB.2 = 1
      PORTB |=1<<2;
      // Calcula sua inversa
```

```
      // GMATRIX_INVERSE_COPY(MatrixInverse,Matrix);
      GMATRIX_INVERSE3_COPY(MatrixInverse,Matrix);
      // PORTB.2 = 0
      PORTB &=~(1<<2);
    }
}
```

5 Demos

GMATRIX é distribuída com três demos:

- GMatrixEvaluation: contém um projeto Microsoft Visual C++ 6.0 com várias funções. Por simplicidade, a interface do usuário é feita por linha de comando DOS. No entanto, `GMATRIX` tem sido usada com sucesso em outros tipos de projeto MSVC6.0;
- GMatrixEvaluationAVR: projeto WinAVR apresentado na seção 4.2;
- GMatrixMatlab: exemplo do uso de GMATRIX em C-MEX para MATLAB.

Referências

[Papoulis e Pillai 2001]PAPOULIS, A.; PILLAI, S. U. *Probability, Random Variables and Stochastic Processes*. Fourth edition. [S.l.]: McGraw-Hill, 2001.

[Press et al. 1992]PRESS, W. H. et al. *Numerical Recipes in C: The Art of Scientific Computing*. Second edition. [S.l.]: Cambridge University Press, 1992.