

Gestion des matrices

Améliorer sa programmation sous MATLAB



par Jérôme Briot ([Dut sur developpez.com](#))


Date de publication : 19/11/2007

Dernière mise à jour : 17/03/2009

Cet article est une introduction à la gestion des matrices (tableaux numériques 2D) sous MATLAB.

Contenu : l'article débute par quelques généralités pour mieux comprendre comment MATLAB traite les tableaux 2D. Les différentes méthodes d'indexage sont ensuite présentées ainsi que quelques opérations couramment faites sur les matrices. Enfin, cet article se termine par un glossaire de génération de quelques matrices usuelles.

Public visé : cet article est destiné aux débutants ayant déjà quelques notions de MATLAB et surtout aux utilisateurs avancés qui y trouveront une introduction au concept de vectorisation.

 ***Votre avis et vos suggestions sur cet article m'intéressent ! Alors après votre lecture, n'hésitez pas :***

I - Avant propos.....	3
II - Généralités.....	4
II-A - Matrice ou tableau ?.....	4
II-B - Les scalaires, les vecteurs sont-ils des matrices ?.....	4
II-C - Que contient une matrice ?.....	5
II-D - Stockage des matrices en mémoire.....	5
II-D-1 - Toute matrice est vecteur ?.....	5
II-D-2 - Out of Memory.....	6
III - Méthodes d'indexage.....	7
III-A - Indexage classique (ligne,colonne).....	7
III-B - Indexage linéaire.....	7
III-C - Indexage logique.....	8
III-D - Les outils d'indexage.....	9
III-D-1 - L'opérateur end.....	9
III-D-2 - L'opérateur :.....	10
III-D-3 - Les fonctions sub2ind et ind2sub.....	11
IV - Opérations courantes.....	13
IV-A - Concaténation.....	13
IV-A-1 - Concaténation verticale.....	13
IV-A-2 - Concaténation horizontale.....	14
IV-B - Réplication.....	16
IV-B-1 - Réplication entière.....	16
IV-B-2 - Réplication entrelacée.....	16
IV-C - Redimensionnement.....	17
V - Génération de matrices usuelles.....	19
VI - Conclusions.....	22
VII - Remerciements.....	23

I - Avant propos

De nombreux développeurs, débutants ou familiers avec les langages de bas niveau (C, Fortran, ...), utilisent MATLAB sans prendre le temps de bien comprendre les spécificités de ce langage.

La compréhension de la gestion des matrices (tableaux 2D) par MATLAB est une étape essentielle dans la prise en main de ce langage. En effet, MATLAB est avant tout un logiciel de calcul matriciel et donc, maîtriser la manipulation des matrices, permet d'améliorer les performances des programmes par un codage propre et efficace.



Comprendre la gestion des matrices permet également de comprendre comment MATLAB gère les autres types de variables (tableaux multidimensionnels, structures et des tableaux de cellules) qui ne seront pas abordés ici.

II - Généralités

Commençons par présenter quelques généralités qui permettront d'aborder efficacement la gestion des matrices sous MATLAB.

II-A - Matrice ou tableau ?

D'un point de vue mathématique, une matrice est un tableau rectangulaire composé de m lignes et de n colonnes dont chaque élément correspond à une valeur numérique.

En informatique, on parle plus volontiers de tableaux. On parle même de tableaux multidimensionnels (3D et plus). Etant fortement lié au monde des mathématiques, sous MATLAB, ces deux terminologies coexistent. On parlera donc de matrices dans le cas de tableaux 2D.

Note : il peut arriver que l'on parle abusivement de matrice 3D

II-B - Les scalaires, les vecteurs sont-ils des matrices ?

Créons trois variables représentant respectivement un scalaire, un vecteur et une matrice, et regardons ce que le workspace contient :

```
a=1;
b=[1 1];
c=[1 1 ; 1 1];
workspace
```

Name	Size	Bytes	Class	Value
a	1x1	8	double	1
b	1x2	16	double	[1,1]
c	2x2	32	double	[1,1;1,1]

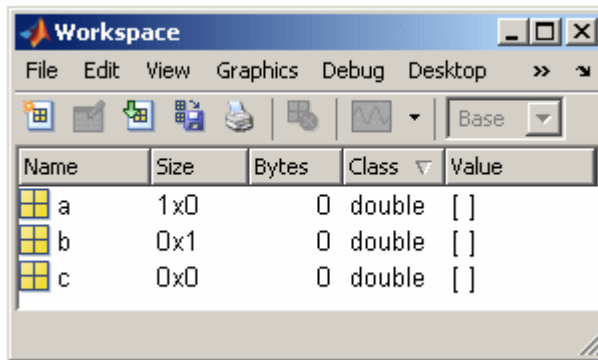
On remarque que les trois variables possèdent toutes 2 dimensions :

- le scalaire "a" a pour dimension 1x1
- le vecteur "b" a pour dimension 1x2
- la matrice "c" a pour dimension 2x2

MATLAB ne fait donc pas la différence entre ces trois variables, ce sont toutes des matrices.

Cas particuliers : il est également possible de définir des matrices de dimension 0 ou vide :

```
a=ones(1,0);
b=ones(0,1);
c=[];
```



L'utilité de ces matrices ne sera pas discutée dans cet article.

II-C - Que contient une matrice ?

Par définition, une matrice contient des valeurs numériques. En informatique, ces valeurs sont stockées en mémoire sous forme de nombres binaires codés sur un certain nombre de **bits**.

Bien que différents types de variables numériques soient disponibles sous MATLAB, une matrice ne peut contenir qu'un seul et unique type parmi les suivants :

Type	Désignation	Taille	Plage
int8	Entier signé (+/-)	8 bits - 1 octet	[-128 127]
uint8	Entier non signé	8 bits - 1 octet	[0 255]
int16	Entier signé (+/-)	16 bits - 2 octets	[-32768 32767]
uint16	Entier non signé	16 bits - 2 octets	[0 65535]
int32	Entier signé (+/-)	32 bits - 4 octets	[-2147483648 2147483647]
uint32	Entier non signé	32 bits - 4 octets	[0 4294967295]
int64	Entier signé (+/-)	64 bits - 8 octets	[-9223372036854775808 9223372036854775807]
uint64	Entier non signé	64 bits - 8 octets	[0 18446744073709551615]
single	Réel simple précision	32 bits - 4 octets	[realmin('single') realmax('single')]
double	Réel double précision	64 bits - 8 octets	[realmin('double') realmax('double')]

Note : les plages de valeurs sont données à titre indicatif. Se référer à la documentation MATLAB pour plus d'informations (voir **intmin**, **intmax**, **realmin**, **realmax**)

Le type par défaut de MATLAB est le type double (64bits).

II-D - Stockage des matrices en mémoire

II-D-1 - Toute matrice est vecteur ?

Une matrice se représente visuellement comme un tableau 2D composé de lignes et de colonnes. Le stockage informatique de ce type de données en mémoire ne suit pas cette représentation. Chaque élément de la matrice est donc stocké à un endroit précis de la mémoire. Dans la majorité des langages informatiques, l'espace de stockage d'une variable peut être fragmenté. Ce qui signifie que les éléments n'ont pas besoin de se trouver côte à côte. MATLAB lui, ne peut pas gérer le stockage des variables de cette manière.

En effet, MATLAB a été conçu dès le départ pour effectuer du calcul matriciel de manière simplifiée et optimisée. MATLAB stocke donc les valeurs des matrices dans des blocs de mémoire contigüe (non fragmentée). Une matrice est donc stockée "sous forme" d'un vecteur, colonne par colonne, avec chaque élément mis bout à bout.

Par exemple, soit la matrice 3x3 suivante :

1	4	7
2	5	8
3	6	9

MATLAB la stockera sous forme d'un vecteur, colonne par colonne (1), comme ceci :

1
2
3
4
5
6
7
8
9

II-D-2 - Out of Memory

Le stockage des données dans des blocs de mémoire contigüe est un réel avantage afin d'accéder rapidement au contenu d'une matrice ou de le modifier.

Par contre, cela est très pénalisant en quantité de mémoire disponible en vue du stockage d'une matrice. En effet, avec les langages qui utilisent la mémoire de façon fragmentée, on peut dire que la taille maximale de la matrice que l'on peut stocker est (environ) égale à la quantité de mémoire disponible.

Sous MATLAB, la taille maximale de la matrice que l'on peut stocker, est égale à la taille du plus gros bloc de mémoire contigüe disponible.

Ceci explique que de nombreux développeurs travaillant sur des données volumineuses ne comprennent généralement pas pourquoi MATLAB n'est pas capable, par exemple, de gérer une matrice d'environ 100 Mo alors qu'ils disposent de plusieurs centaines de Mo de mémoire disponibles. Ils sont alors confrontés à ce message d'erreur frustrant :

Out of memory. Type `HELP MEMORY` for your options.

La solution ne consiste donc pas à augmenter la quantité de mémoire disponible, mais plutôt à chercher à maximiser la taille des blocs de mémoire contigüe.

Plus d'informations à ce sujet dans la FAQ MATLAB : **[Out of memory. Type HELP MEMORY for your options.](#)**

III - Méthodes d'indexage

Après ces généralités permettant au final de comprendre comment MATLAB stocke les matrices en mémoire, passons aux méthodes d'indexage qui consistent à repérer des éléments de la matrice à l'aide d'indices.

III-A - Indexage classique (ligne,colonne)

La méthode d'indexage la plus évidente, consiste à spécifier la position d'un élément en fonction de l'indice de la ligne et de l'indice de la colonne où il se trouve dans la matrice en prenant comme premier élément, celui situé en haut à gauche. L'indexage s'effectue entre parenthèses avec en premier l'indice de la ligne et en second, l'indice de la colonne, soit : $M(\text{idx ligne}, \text{idx colonne})$.

Par exemple :

```
M=magic(4)

M =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1

M(4,3) % Élément situé à la ligne 4 et à la colonne 3 de M

ans =

    15
```

III-B - Indexage linéaire

Comme on l'a vu dans le chapitre précédent, MATLAB ne stocke pas les matrices dans la mémoire sous forme de tableaux à 2 dimensions, mais sous forme de vecteurs (colonne par colonne) dont chaque élément est mis bout à bout. Ce type de stockage permet d'utiliser une autre technique d'indexage, que l'on appelle indexage linéaire. On ne localise plus un élément d'une matrice par le couple d'indices ligne-colonne, mais directement par la position de l'élément dans le vecteur stocké en mémoire.

Par exemple, la matrice suivante :

```
M=magic(4)

M =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

Cette matrice sera stockée en mémoire sous cette forme :

```
M =

    16
     5
     9
     4
     2
    11
     7
    14
```

```

3
10
6
15
13
8
12
1
    
```

On pourra donc aisément récupérer n'importe quel élément en donnant un seul indice :

```

>> M(4,3) % Indexage classique

ans =

    15

>> M(12) % Indexage linéaire

ans =

    15
    
```

La relation de passage entre l'indexage classique et l'indexage linéaire est :

$M(i,j) \Rightarrow M(i+(j-1)*\text{size}(M,1))$

Soit dans l'exemple précédent, $M(4,3) \Rightarrow M(4+(3-1)*4) \Rightarrow M(12)$

De la même manière, la relation de passage entre l'indexage linéaire et l'indexage classique est :

$M(k) \Rightarrow M(k-\text{ceil}(k/\text{size}(M,1))*\text{size}(M,1)+\text{size}(M,1),\text{ceil}(k/\text{size}(M,1)))$

Soit dans l'exemple précédent, $M(12) \Rightarrow M(12-(\text{ceil}(12/4))*4+4,\text{ceil}(12/4)) \Rightarrow M(12-3*4+4,3) \Rightarrow M(4,3)$

III-C - Indexage logique

Il existe une troisième forme d'indexage basée sur les conditions logiques. On la désigne par indexage logique et on l'utilise principalement avec les opérateurs relationnels et les opérateurs logiques. Ce type d'indexage permet d'améliorer l'efficacité des codes en évitant l'utilisation de fonctions supplémentaires (principalement **find**). L'indexage logique est souvent utilisé avec les fonctions **any** et **all**.

Par exemple, on souhaite trouver toutes les valeurs supérieures à 3 (soit 8 et 7) dans la matrice suivante :

```

>> X=[8 1 ; 2 7]

X =

     8     1
     2     7
    
```

La méthode classique consiste à utiliser **find** comme ceci :

```

>> X=[8 1 ; 2 7]

X =

     8     1
     2     7

>> idx=find(X>3) % Indexage linéaire

idx =

     1
     4
    
```

```
>> X(idx)

ans =

     8
     7
```

On remarque que idx correspond aux indices linéaires de X>3
L'indexage logique consiste simplement à se passer de la fonction **find** :

```
>> X=[8 1 ; 2 7]

X =

     8     1
     2     7

>> idx = (X>3) % Indexage logique

idx =

     1     0
     0     1

>> X(idx)

ans =

     8
     7
```

On remarque maintenant que idx est une matrice contenant des valeurs logiques (0 ou 1).

III-D - Les outils d'indexage

MATLAB possède plusieurs outils qui permettent de simplifier ou d'optimiser l'indexage des matrices.

III-D-1 - L'opérateur end

end est un mot-clé de MATLAB. Il sert à fermer les structures itératives (FOR-END par exemple) ou les structures conditionnelles (IF-END, par exemple). Mais ce mot-clé peut aussi être employé comme opérateur d'indexage. Dans le cas d'un vecteur, le dernier élément est retourné :

```
>> M = [4 7 2]

M =

     4     7     2

>> M(end)

ans =

     2
```

Dans le cas d'une matrice, il indexe automatiquement le dernier élément d'une des dimensions.

```
>> X = [8 1 ; 2 7]

X =
```

```

      8     1
      2     7

>> X(end,1) % Dernier élément de la première colonne

ans =

     2

>> X(end,2) % Dernier élément de la seconde colonne

ans =

     7

>> X(1,end) % Dernier élément de la première ligne

ans =

     1

>> X(2,end) % Dernier élément de la seconde ligne

ans =

     7

>> X(end,end) % Dernier élément de la matrice (indexage classique)

ans =

     7

>> X(end) % Dernier élément de la matrice (indexage linéaire)

ans =

     7
    
```

III-D-2 - L'opérateur :

Classiquement, l'opérateur : (*colon* en anglais) sert lors de la définition d'un vecteur :

```

>> v = 1:5

v =

     1     2     3     4     5

>> v = 7:0.2:8

v =

 7.0000  7.2000  7.4000  7.6000  7.8000  8.0000
    
```

Dans le cas de l'indexage, il sert d'abord à spécifier une plage d'indices :

```

>> X = [8 1 ; 2 7 ; 5 9]

X =

     8     1
     2     7
     5     9
    
```

```
>> X(2:3,2) % Eléments lignes 2 à 3 et colonne 2

ans =

     7
     9
```

Employé seul, il sert aussi à spécifier tous les indices d'une dimension :

```
>> X = [8 1 ; 2 7 ; 5 9]

X =

     8     1
     2     7
     5     9

>> X(:,1) % Tous les éléments de la première colonne

ans =

     8
     2
     5

>> X(2,:) % Tous les éléments de la seconde ligne

ans =

     2     7

>> X(:,:) % Tous les éléments de X (indexage classique)

ans =

     8     1
     2     7
     5     9

>> X(:) % Tous les éléments de X (indexage linéaire)

ans =

     8
     2
     5
     1
     7
     9
```

III-D-3 - Les fonctions sub2ind et ind2sub

La fonction `sub2ind` sert à passer simplement de l'indexage classique (ligne,colonne) vers l'indexage linéaire. La fonction `ind2sub` sert à passer simplement de l'indexage linéaire vers l'indexage classique (ligne,colonne).

```
>> M=magic(4)

M =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1

>> idx=sub2ind(size(M),4,3)

idx =
```

```
12
>> [r,c]=ind2sub(size(M),12)

r =
    4

c =
    3
```

IV - Opérations courantes

IV-A - Concaténation

La concaténation consiste à coller des matrices bout à bout afin d'obtenir une matrice supplémentaire. Cette opération s'effectue entre crochets. A l'intérieur de ces crochets, les différentes matrices doivent être séparées, soit par des point-virgules pour une concaténation verticale, soit par des virgules ou des espaces pour une concaténation horizontale.

IV-A-1 - Concaténation verticale

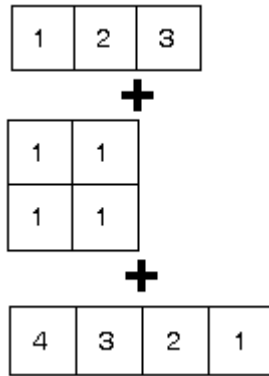
La concaténation verticale consiste à mettre des matrices les unes sur les autres. Les différentes matrices doivent impérativement avoir le même nombre de colonnes. Par exemple, pour concaténer verticalement les trois matrices suivantes :

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline \end{array}
 +
 \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}
 +
 \begin{array}{|c|c|c|} \hline 3 & 2 & 1 \\ \hline \end{array}$$

```

>> A=[1 2 3]
A =
     1     2     3
>> B=ones(2,3)
B =
     1     1     1
     1     1     1
>> C=[3 2 1]
C =
     3     2     1
>> X=[A ; B ; C]
X =
     1     2     3
     1     1     1
     1     1     1
     3     2     1
    
```

Si les matrices n'ont pas le même nombre de colonnes, MATLAB retourne un message d'erreur. Par exemple, si l'on tente de concaténer verticalement ces trois matrices :



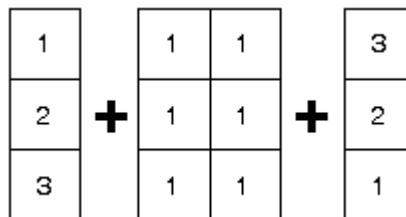
```

>> A=[1 2 3]
A =
     1     2     3
>> B=ones(2)
B =
     1     1
     1     1
>> C=[4 3 2 1]
C =
     4     3     2     1
>> X=[A ; B ; C]
??? Error using ==> vertcat
CAT arguments dimensions are not consistent.

```

IV-A-2 - Concaténation horizontale

La concaténation horizontale consiste à mettre des matrices les unes à côté des autres. Les différentes matrices doivent impérativement avoir le même nombre de lignes. Par exemple, pour concaténer horizontalement les trois matrices suivantes :



```

>> A=[1;2;3]
A =
     1
     2
     3
>> B=ones(3,2)
B =
     1     1
     1     1
     1     1

```

```

1     1
1     1

>> C=[3;2;1]

C =

     3
     2
     1

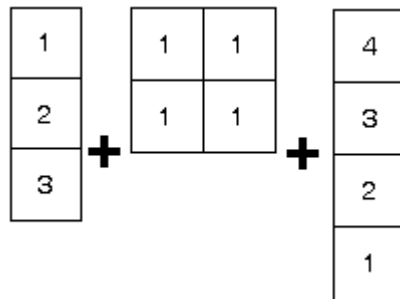
>> X=[A,B,C]

X =

     1     1     1     3
     2     1     1     2
     3     1     1     1

```

Si les matrices n'ont pas le même nombre de lignes, MATLAB retourne un message d'erreur. Par exemple, si l'on tente de concaténer horizontalement ces trois matrices :



```

>> A=[1;2;3]

A =

     1
     2
     3

>> B=ones(2)

B =

     1     1
     1     1

>> C=[4;3;2;1]

C =

     4
     3
     2
     1

>> X=[A,B,C]
??? Error using ==> horzcat
CAT arguments dimensions are not consistent.

```

IV-B - Réplication

IV-B-1 - Réplication entière

La méthode la plus simple pour répliquer une matrice consiste à employer la fonction **repmat** :

```
>> A=[1 2 3 ; 4 5 6]

A =

     1     2     3
     4     5     6

>> X=repmat(A,3,2)

X =

     1     2     3     1     2     3
     4     5     6     4     5     6
     1     2     3     1     2     3
     4     5     6     4     5     6
     1     2     3     1     2     3
     4     5     6     4     5     6
```

L'autre solution consiste à répéter les indices lors de l'indexage :

```
>> A

A =

     1     2     3
     4     5     6

>> X=A([1 2 1 2 1 2],[1 2 3 1 2 3])

X =

     1     2     3     1     2     3
     4     5     6     4     5     6
     1     2     3     1     2     3
     4     5     6     4     5     6
     1     2     3     1     2     3
     4     5     6     4     5     6
```

IV-B-2 - Réplication entrelacée

Il peut arriver que l'on souhaite obtenir la réplication d'une matrice, non pas de façon entière mise bout à bout, mais de façon entrelacée en répétant un certain nombre de fois chaque colonne et/ou chaque ligne. Dans ce cas, il faut utiliser la fonction **kron** :

```
>> M=[8 4 ; 1 7]

M =

     8     4
     1     7

>> M=kron(M,ones(2,3))

M =
```

8	8	8	4	4	4
8	8	8	4	4	4
1	1	1	7	7	7
1	1	1	7	7	7

On peut également toujours utiliser la méthode de répétition des indices :

```
>> M=[8 4 ; 1 7]

M =

     8     4
     1     7

>> M([1 1 2 2],[1 1 1 2 2 2])

ans =

     8     8     8     4     4     4
     8     8     8     4     4     4
     1     1     1     7     7     7
     1     1     1     7     7     7
```

IV-C - Redimensionnement

Le redimensionnement d'une matrice consiste à modifier le nombre de lignes et/ou de colonnes d'une matrice en conservant le même nombre d'éléments à l'intérieur de celle-ci.

Pour modifier les dimensions d'une matrice, il faut utiliser la fonction **reshape**.

```
>> M=[8 4 5; 1 7 6]

M =

     8     4     5
     1     7     6

>> M=reshape(M,3,2)

M =

     8     7
     1     5
     4     6
```

On remarque que les éléments sont réordonnés colonne par colonne, conformément à la manière dont MATLAB stocke les matrices.

Il est toujours possible de manipuler les indices pour obtenir le même résultat. Mais dans ce cas, il faudra utiliser l'indexage linéaire :

```
>> M=[8 4 5; 1 7 6]


M =

     8     4     5
     1     7     6

>> M([[1;2;3] [4;5;6]])


ans =

     8     7
     1     5
     4     6
```

 *Le nombre total d'éléments dans la matrice doit rester inchangé.*
*Dans le cas contraire, MATLAB retournera un message d'erreur : **To RESHAPE the number of elements must not change.***

V - Génération de matrices usuelles


Voici quelques matrices usuelles et les codes qui permettent de les obtenir.

 *Il faudra veiller à nettoyer l'espace de travail (avec la fonction **clear**) et à ne surtout pas avoir une variable nommée **M** avant d'essayer ces codes.*

Matrice					Code MATLAB
0	0	0	0	0	<code>M = zeros(3,5);</code>
0	0	0	0	0	OU
0	0	0	0	0	<code>M(3,5) = 0;</code>
0	0	0	0	0	
0	0	0	0	0	<code>M = zeros(5);</code>
0	0	0	0	0	OU
0	0	0	0	0	<code>M(5,5) = 0;</code>
0	0	0	0	0	
0	0	0	0	0	
1	1	1	1	1	<code>M = ones(3,5);</code>
1	1	1	1	1	
1	1	1	1	1	
1	1	1	1	1	<code>M = ones(5);</code>
1	1	1	1	1	
1	1	1	1	1	
1	1	1	1	1	
1	1	1	1	1	
77	77	77	77	77	<code>M = ones(3,5)*77;</code>
77	77	77	77	77	OU
77	77	77	77	77	<code>M = zeros(3,5);</code> <code>M(:)=77;</code>
77	77	77	77	77	OU
					<code>M = repmat(77,3,5);</code>
					OU
					<code>M = 77;</code>

					<code>M = M(ones(3,5));</code>
					<code>M = ones(5)*77;</code>
<code>77</code>	<code>77</code>	<code>77</code>	<code>77</code>	<code>77</code>	
<code>77</code>	<code>77</code>	<code>77</code>	<code>77</code>	<code>77</code>	OU
					<code>M = zeros(5);</code> <code>M(:) = 77;</code>
<code>77</code>	<code>77</code>	<code>77</code>	<code>77</code>	<code>77</code>	
<code>77</code>	<code>77</code>	<code>77</code>	<code>77</code>	<code>77</code>	OU
					<code>M = repmat(77,5,5);</code>
<code>77</code>	<code>77</code>	<code>77</code>	<code>77</code>	<code>77</code>	
					OU
					<code>M = 77;</code>

	<code>M = M(ones(5));</code>					
	<code>M = eye(3,5);</code>					
1	0	0	0	0		
0	1	0	0	0		
0	0	1	0	0		
	<code>M = eye(5);</code>					
1	0	0	0	0		
0	1	0	0	0		
0	0	1	0	0		
0	0	0	1	0		
0	0	0	0	1		
	<code>M = eye(5);</code> <code>M = M(:,end:-1:1)</code>					
0	0	0	0	1		
0	0	0	1	0		
0	0	1	0	0		
0	1	0	0	0		
1	0	0	0	0		
	<code>M(3,5) = 1;</code>					
0	0	0	0	0		
0	0	0	0	0		
0	0	0	0	1		
	<code>n = 5;</code> <code>M = toeplitz([1 3</code> <code>zeros(1,n-2)], [1</code> <code>2 zeros(1,n-2)]);</code>					
1	2	0	0	0		
3	1	2	0	0		
0	3	1	2	0		
0	0	3	1	2		
0	0	0	3	1		
	<code>M = [1 2 ; 3 4];</code> <code>M =</code> <code>kron(M,ones(2,3));</code>					
1	1	1	2	2	2	
1	1	1	2	2	2	
3	3	3	4	4	4	
3	3	3	4	4	4	

 *N'hésitez pas à soumettre à l'auteur d'autres matrices usuelles afin de compléter ce tableau.*

VI - Conclusions

Pour résumer :

- les scalaires, les vecteurs et les matrices sont tous considérés comme des matrices dans l'espace de travail de MATLAB
- les matrices sont stockées en mémoire sous forme de vecteurs colonne par colonne
- trois méthodes d'indexage sont utilisables : classique, linéaire et logique
- MATLAB possède des fonctions toutes faites pour la concaténation, la réplication ou le redimensionnement des matrices

Certaines parties de cet article, comme celle relative au stockage des matrices en mémoire, sont volontairement simplifiées. En effet, le but de cet article est avant tout de bien faire comprendre aux développeurs que MATLAB est un langage matriciel. Il faut donc, généralement, pour obtenir du code efficace, manipuler les matrices dans leur ensemble et non pas élément par élément. C'est ce que l'on appelle la **vectorisation**

VII - Remerciements

L'auteur tient à remercier **Caro-Line** et **Pedro** pour la correction orthographique de cet article.

1 : Comme c'est le cas en Fortran. Cette convention a été conservée car MATLAB était initialement écrit en Fortran