République algérienne Démocratique et Populaire
Ministère de l'enseignement supérieur et de la recherche scientifique
Université de JIJEL
Faculté des Sciences Exactes et Informatique
Département d'informatique

Les Outils de Programmation pour les Mathématiques (OPM)

L1 Informatique/Mathématiques/ MI

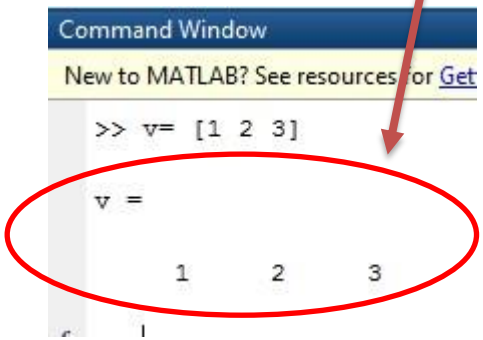
Les vecteurs et les matrices dans Matlab

Les vecteurs

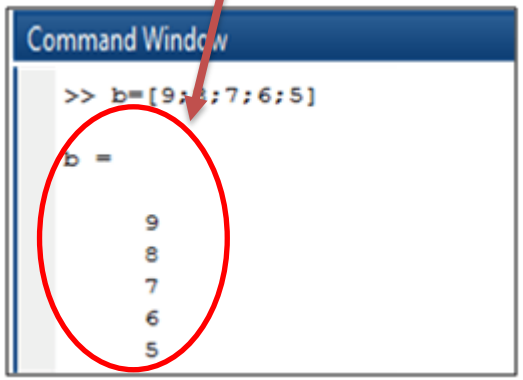
Un vecteur est une liste ordonnée d'éléments. Si les éléments sont arrangés horizontalement on dit que le vecteur est **un vecteur ligne**, par contre si les éléments sont arrangés verticalement on dit que c'est **un vecteur colonne**.

Exemple:

Vecteur ligne



Vecteur colonne



Les vecteurs

Vecteur ligne

Pour créer un **vecteur ligne** il suffit d'écrire la liste de ses composants entre crochets **[ ]**, et les séparer par des **espaces** ou des **virgules.**

Exemple

```
>> u=[1 4 7 9]
u =
     1     4     7     9

>> u=[1,4,7,9]
u =
     1     4     7     9
```

Les vecteurs

Vecteur colonne

Pour créer un **vecteur colonne** il est possible d'utiliser une des méthodes suivantes :

1. Ecrire les composants du vecteur entre crochets **[et]** et les **séparer** par des points-virgules (**;**)

Exemple

```
>> U = [ 4 ; -2 ; 1 ]                    % Création d'un vecteur colonne U
      U =
            4
           -2
            1
```

Les vecteurs

Vecteur colonne

2. On définit un vecteur colonne en donnant la liste de ses éléments séparés par des retours chariots (touche Entrée)

```
>> U = [
    4
   -2
    1
]
```

```
U =

     4
    -2
     1
```

3. calculer le transposé d'un vecteur ligne :

```
>> U = [ 4 -2 1 ]'                    % Création d'un vecteur colonne U
```

```
U =

     4
    -2
     1
```

Les vecteurs

Longueur d'un vecteur

Il est inutile de définir la longueur d'un vecteur au préalable. Cette longueur sera établie automatiquement à partir de l'expression mathématique définissant le vecteur ou à partir des données.

On peut obtenir la longueur d'un vecteur donné grâce à la commande **length**.

Exemple:

```
>> x1 = [1 2 3];
>> length(x1)
ans=
     3
```

Les vecteurs

L'opérateur colon " :"

Avant d'aller plus loin, il est nécessaire de se familiariser avec une notation que l'on retrouve partout en Matlab, celle de l'opérateur " **:**" (en anglais colon).

En Matlab, quand on écrit **1 :10** par exemple, cela signifie "tous les nombres de 1 à 10" (c.a.d. 1 2 3 4 5 6 7 8 9 10), et "**1 : 2:10**" signifie "tous les nombres de 1 à 10 avec un pas de 2 (et donc les nombres : 1 3 5 7 9)", le pas étant unitaire par défaut.

Cette notation doit être impérativement comprise car elle est utilisée sans arrêt.

Les vecteurs

Il est très facile de définir un vecteur dont les composantes forment une suite arithmétique.

Pour définir un vecteur x dont les composantes forment une suite arithmétique de raison h(le pas d'incrémentation/décrémentation) , de premier terme « a » et de dernier terme « b» , on utilisera **l'opérateur colon " :"** **et** on écrira

x = a:h:b

Les vecteurs

Donc pour définir un vecteur X dont les composantes forment une suite arithmétique on écrit:

```
X = premier_élément : le_pas : dernier_élément
```

(Les crochets sont facultatifs)

Par exemple :

```
>> X = [0:2:10]                    % le vecteur X contient les nombres pairs < 12
      X =
            0     2     4     6     8    10
>> X = [-4:2:6]                    % on peut aussi écrire colon(-4,2,6)
      X =
           -4    -2     0     2     4     6
>> X = 0:0.2:1                     % on peut aussi écrire colon(0,0.2,1)
      X =
            0    0.2000    0.4000    0.6000    0.8000    1.0000
```

Les vecteurs

Si le pas = 1 on écrit:

```
X = premier_élément : dernier_élément
```

(Les crochets sont facultatifs dans ce cas)

Par exemple :

```
>> X = 1:8                                  % on peut aussi écrire colon(1,8)
      X =
            1     2     3     4     5     6     7     8
>> X = [1:8]
      X =
            1     2     3     4     5     6     7     8
```

Les vecteurs

La fonction linspace

La fonction **linspace** permet de définir un vecteur x de longueur *N* dont les composantes forment une suite arithmétique de premier terme *a* et de dernier terme *b* (donc de raison (*b*-*a*)/(*N*-*1*)). Les composantes du vecteur sont donc *lin*éairement e*spacé*s.

La syntaxe est

x = linspace(a,b,N)

Les vecteurs

La fonction linspace
Exemple:

```
>> X = linspace(1,10,4)          % un vecteur de quatre élément de 1 à 10
      X =
            1     4     7    10

>> Y = linspace(13,40,4)         % un vecteur de quatre élément de 13 à 40
      Y =
           13    22    31    40
```

Les vecteurs

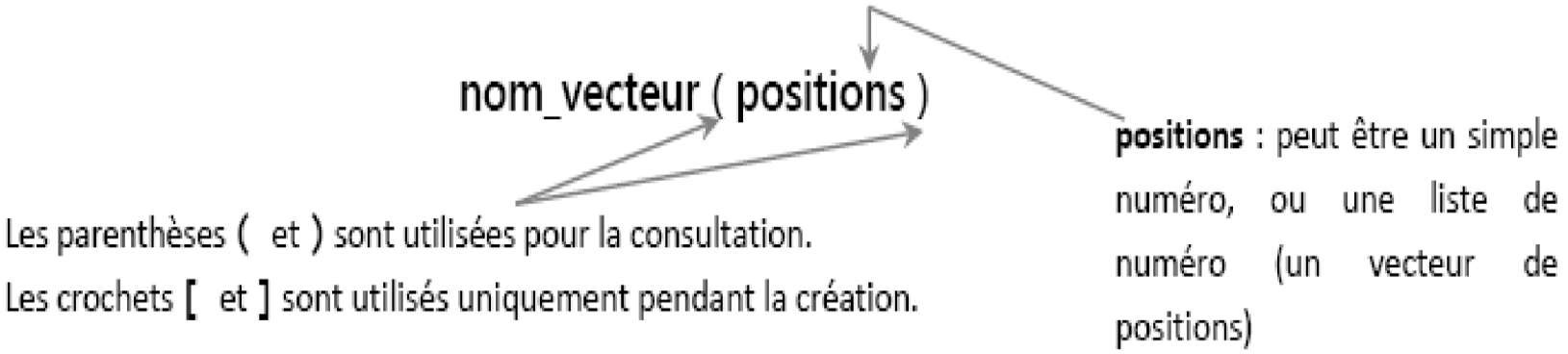
Référencement et accès aux éléments d'un vecteur

Les éléments d'un vecteur peuvent être manipulés grâce à leur indice dans le vecteur.

Le k-ieme élément du vecteur x est désignée par x(k).

Le premier élément d'un vecteur a obligatoirement pour indice 1.

L'accès aux éléments d'un vecteur se fait en utilisant la syntaxe générale suivante :



Les vecteurs

Référencement et accès aux éléments d'un vecteur

Exemple1:

```
>> V =[1:2:12,-1:1:2]
V =
 1   3   5   7   9   11   -1   0   1   2

>> V([1,3,4])
Ans=
    1   5   7
>> V(1)=8
Ans=
 8   3   5   7   9   11   -1   0   1   2
>> V(12)=-5
Ans=
 8   3   5   7   9   11   -1   0   1   2   0   -5
```

Les vecteurs

Référencement et accès aux éléments d'un vecteur

Exemple 1(suite):

```
>> V(2) =[]
V =
 8    5    7    9   11   -1    0    1    2  0  -5
%supprimer le deuxième élément

>> V(3:5)=[]
Ans=
    8    5   -1  0  1  2  0  -5
%supprimer du 3eme au  5eme élément
```

Les vecteurs

Exemple2:

```
>> v=[5 -1 13 -6 7]      %création du vecteur v de 5 valeurs
v =
   5   -1   13   -6    7
>> v(3)                  % la 3eme position
ans =
   13
>> v(2:4)                % de la deuxième position jusqu'au quatrième
ans =
   -1   13   -6
>> v(4:-2:1)             % de la 4eme position jusqu'à la 1ere avec un pas = -2
ans =
   -6   -1
>> v(3:end)              % de la 3eme position jusqu'à la dernière
ans =
   13   -6    7
```

Les vecteurs

Exemple2:

```
>> v([1,3,4])            % la 1ere, 3eme et 4eme position uniquement
ans =
   5   13   -6
>> v(1)=8                % modifier la valeur du premier élément par la valeur 8
v =
   8   -1   13   -6   7
>> v(6)=-3               % ajouter un 6ème élément avec la valeur -3
v =
   8   -1   13   -6   7   -3
>> v(9)=5                % ajouter un 9eme élément avec la valeur 5
v =
   8   -1   13   -6   7   -3   0   0   5
```

Les vecteurs

Les opérations élément-par-élément pour les vecteurs

Avec deux vecteurs $\vec{u}$ et $\vec{v}$, il est possible de réaliser des calculs élément par élément en utilisant les opérations suivantes :

Opération	Exemple d'application :
	>> u=[-2 6 1]; % création de u >> v=[3 -1 4]; % création de v
Addition (+)	>> u+2 % Addition d'un scalaire à un vecteur ans = 0 8 3 >> u+v % Addition de deux vecteurs ans = 1 5 5
Soustraction (-)	>> u-2 % soustraction d'un scalaire d'un vecteur ans = -4 4 -1 >> u-v % soustraction élément par élément de deux vecteurs ans = -5 7 -3

Les vecteurs

Les opérations élément-par-élément pour les vecteurs

Multiplication élément par élément (.*)	>> u*2 % Produit d'un scalaire par un vecteur ans = -4 12 2 >> u.*2 % produit élément par élément d'un scalaire par un vecteur ans = -4 12 2 >> u.*v % produit élément par élément de deux vecteurs ans = -6 -6 4 >> v.*u % produit élément par élément de deux vecteurs ans = -6 -6 4
Division élément par élément (./)	>> u/2 % Division d'un vecteur par un scalaire ans = -1.0000 3.0000 0.5000 >> u./2 % Division élément par élément d'un vecteur par un scalaire ans = -1.0000 3.0000 0.5000 >> u./v % Division élément par élément de deux vecteurs ans = -0.6667 -6.0000 0.2500
Puissance élément par élément (.^)	>> u.^2 % puissance élément par élément d'un vecteur à un scalaire ans = 4 36 1 >> u.^v % puissance élément par élément de deux vecteurs ans = -8.0000 0.1667 1.0000

Remarque:

u*u génère une erreur car cette expression réfère a une multiplication de matrices (u*u doit être réécrite u*u' ou u'*u pour être valide).

Vecteurs spéciaux

Les commandes ones, zeros et rand permettent de définir des vecteurs dont les éléments ont respectivement pour valeurs 0, 1 et des nombres générés de manière aléatoire.

`ones (1,n)`	: vecteur ligne de longueur n dont tous les éléments valent 1
`ones(m,1)`	: vecteur colonne de longueur m dont tous les éléments valent 1
`zeros(1,n)`	: vecteur ligne de longueur n dont tous les éléments valent 0
`zeros(m,1)`	: vecteur colonne de longueur m dont tous les éléments valent 0
`rand(1,n)`	: vecteur ligne de longueur n dont les éléments sont générés de manière aléatoire entre 0 et 1
`rand(m,1)`	: vecteur colonne de longueur m dont les éléments sont générés de manière aléatoire entre 0 et 1

Fonctions communes aux vecteurs lignes et colonnes

Fonction	Objectif
sum(x)	Somme des éléments du vecteur x
prod(x)	Produit des éléments du vecteur x
max (x)	Plus grand élément dans x
min(x)	Plus petit élément dans x
mean(x)	Moyenne des éléments de x
sort(x)	Ordonne les éléments du vecteur x par ordre croissant
fliplr(x)	Renverse l'ordre des éléments du vecteur x
length(x)	Retourne la taille d'un vecteur

Matlab et les matrices

Comme son nom l'indique, Matlab est un langage de programmation pensé pour le calcul matriciel, où il excelle en terme de performances.

Tout le but d'un programme en Matlab est de faire passer un calcul classique à travers un calcul matriciel pour que Matlab le réalise très rapidement. Cela s'appelle la **vectorisation**

Nous allons voir maintenant comment Matlab gère les matrices.

Déclarer des matrices dans Matlab

On distinguera plusieurs types de matrices dans Matlab. Tout d'abord, il y a les **matrices classiques**

- Pour déclarer une matrice dans une variable A, on énumère entre **crochets** ses éléments (séparés par des espaces ou des virgules), et on utilise un point virgule(ou la touche **entrée**) pour passer d'une ligne à l'autre :

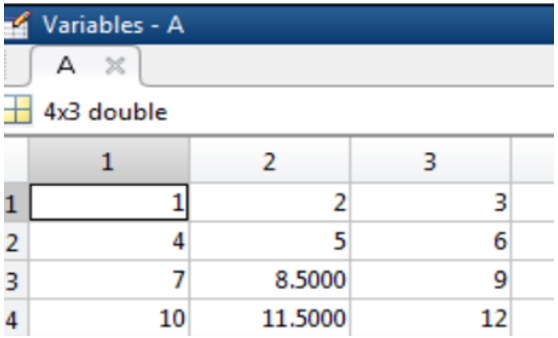
```
>> A = [1 2 3; 4 5 6; 7 8.5 9; 10 11.5 12]
```

D' après le code ci-dessus, la matrice A possède quatre ligne et trois colonnes.

On peut verifier cela dans la zone des variables.

Si on double clique sur la variable dans la zone des variables, son contenu s'affiche.

On peut aussi utiliser la fonction disp pour afficher une matrice.



Déclarer des matrices dans Matlab

Pour illustrer cela, considérant la matrice suivante:

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

Cette Matrice peut être déclarer en Matlab avec une des syntaxes suivantes:

```
>> A=[1,2,3,4;5,6,7,8;9,10,11,12];

>> A=[1 2 3 4;5 6 7 8;9 10 11 12];

>> A=[1 2 3 4
      5 6 7 8
      9 10 11 12];

>> A=[[1;5;9],[2;6;10],[3;7;11],[4;8;12]]
```

Déclarer des matrices dans Matlab

Le nombre d'éléments dans chaque ligne (nombre de colonnes) doit être identique dans toutes les lignes de la matrice, sinon une erreur sera signalée par MATLAB.

Exemple:

```
>> X=[1 2;3 4 5]
Error using vertcat
Dimensions of matrices being concatenated are not consistent.
```

Déclarer des matrices dans Matlab

A vous de jouer : écrivez cette matrice sous forme de tableau

```
>> B = [4.2 7.6 8.2; 4.1 0.5 0]
```

Solution

4.2	7.6	8.2
4.1	0.5	0

Et cette matrice, que donne-t-elle ?

```
>> C = [5.7  0.2  6.2  5.1; 7.1  2.4  8.4; 1.2  0.4  8.4  6.4]
```

Solution : Une erreur de syntaxe car il n'y a pas le même nombre d'élements à chaque ligne.

Accéder aux éléments des matrices

L'accès aux éléments d'une matrice se fait en utilisant la syntaxe générale suivante :

nom_matrice (positions_lignes , positions_colonnes)

positions : peut être un simple numéro, ou une liste de numéro (un vecteur de positions)

Les parenthèses **(et) sont utilisées pour la consultation.**

Les crochets **[et] sont utilisés uniquement pendant la création.**

Accéder aux éléments des matrices

Remarques:

- L'accès à un élément de la ligne i et la colonne j de la matrice **A** se fait par **A(i,j)**

- L'accès à toute la ligne numéro i de la matrice **A** se fait par **: A(i,:)**

- L'accès à toute la colonne numéro j de la matrice **A** se fait par **A(:,j)**

Notez qu'en Matlab, on commence la numérotation de lignes/colonnes d'une matrice à 1. En C, elles commencen à 0 (ne confondez pas).

Accéder aux éléments des matrices

On peut aussi accéder aux éléments d'une matrice par un unique numéro qui est leur ordre dans la matrice. Le premier élément d'une matrice est celui à la 1° ligne et 1° colonne, le second élément est celui à la 2° ligne et 1° colonne, etc…

Si on a

```
>> A = [5.7  0.2  6.2  5.1;  8.7  7.1  2.4  8.4; 1.2  0.4  8.2  6.4]
```

Voici la matrice A avec les positions de ses éléments en rouge :

1 5.7	4 0.2	7 6.2	10 5.1
2 8.7	5 7.1	8 2.4	11 8.4
3 1.2	6 0.4	9 8.2	12 6.4

En général, on ne se sert pas de cette méthode. Mais certaines fonctions renvoient, comme résultat, les numéros d'ordre de certains éléments d'une matrice.

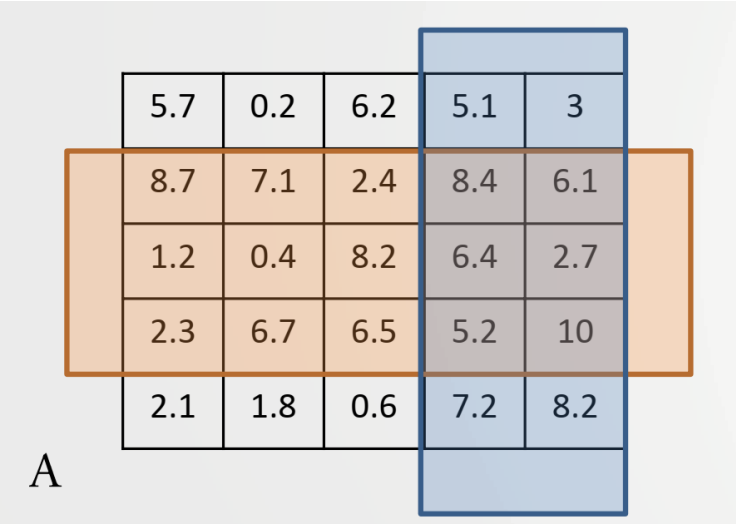
```
>> A(5)

ans =

    7.1000
```

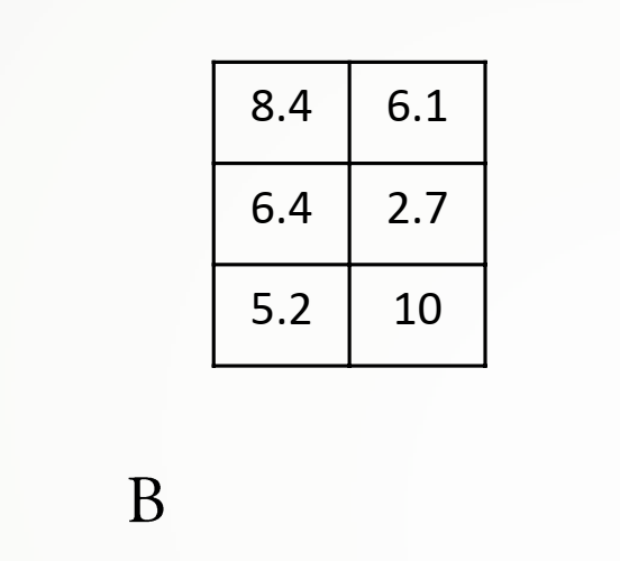
Accéder aux éléments des matrices

On peut aussi facilement extraire des sous-matrices d'une matrice à l'aide de cette syntaxe :

```
>> B = A(2:4, 4:5 )
```

Ici, B sera la matrice constituée des éléments de A aux lignes 2 à 4 et aux colonnes 4 à 5.



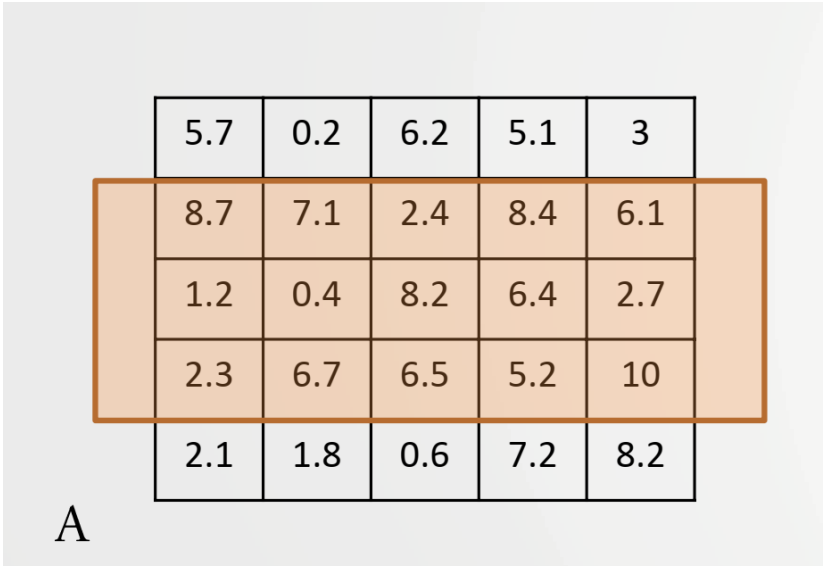


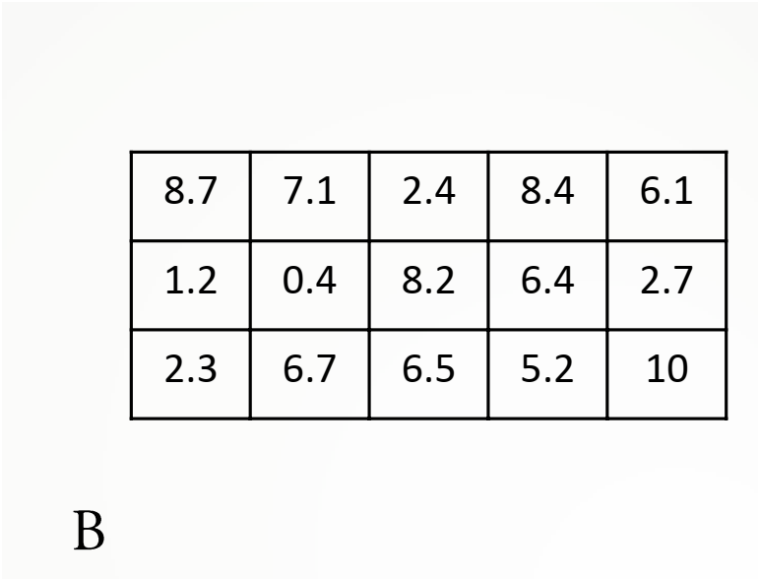
Accéder aux éléments des matrices

On peut aussi récupérer l'intégralité des lignes ou des colonnes d'une matrice dans une sous matrice en laissant des « deux points » seuls :

```
>> B = A(2:4,:)
```

Ici, B sera la matrice constitué des éléments de A aux ligne 2 à 4.





Accéder aux éléments des matrices

Suppression d'une ligne ou une colonne d'une matrice M:

La commande : $M(:, c) = []$ permet de supprimer la $c-$ème colonne de la matrice M.

$M(l, :) = []$ permet de supprimer la $l-$ème ligne de la matrice M

Accéder aux éléments des matrices

Exemple: Suppression d'une ligne ou une colonne d'une matrice M:

```
>> M=[1 2 3; 4 5 6; 7 8 9]                    % définition de la matrice M
M =
   1   2   3
   4   5   6
   7   8   9

>> M(:,3)=[]                                  % supprimer tous les éléments de la 3 ème colonne
M =
   1   2
   4   5
   7   8

>> M(3,:)=[]                                  % supprimer tous les éléments de la 3 ème ligne
M =
   1   2
   4   5
```

Accéder aux éléments des matrices

Ajout de nouvelles lignes ou de colonnes à une matrice M:

- Pour ajouter une nouvelle colonne on utilise la commande :

$$M = [M,[la\ nouvelle\ colonne]]$$

avec la condition que le nombre des éléments dans la nouvelle colonne = $nombre\ de\ lignes\ de\ M$.

- Pour ajouter une nouvelle ligne on utilise la commande :

$$M = [M;[la\ nouvelle\ ligne]]$$

avec la condition que le nombre des éléments de la nouvelle ligne = le $nombre\ de\ colonnes\ de\ M$.

Ajout de nouvelles lignes ou de colonnes à une matrice M

Exemple

```
>> M=[1 2 3; 4 5 6;7 8 9]                    % définition de M
M =
   1   2   3
   4   5   6
   7   8   9
>> M=[M,[5 ;66; 99]]                         % ajout de la nouvelle colonne [5 ;66 ;99]
M =
   1   2   3   5
   4   5   6  66
   7   8   9  99
>> M=[M;[5 66 99 100]]                       % ajout d'une nouvelle ligne à M [5 66 99 100]
M =
   1   2   3   5
   4   5   6  66
   7   8   9  99
   5  66  99  100
```

Accéder aux éléments des matrices

Exemple :

```
>> A = [1,2,3,4 ; 5,6,7,8 ; 9,10,11,12]      % création de la matrice A
      A =
            1     2     3     4
            5     6     7     8
            9    10    11    12

>> A(2,3)                                    % l'élément sur la 2ème ligne à la 3ème colonne
      ans =
            7

>> A(1,:)                                    % tous les éléments de la 1ère ligne
      ans =
            1     2     3     4

>> A(:,2)                                    % tous les éléments de la 2ème colonne
      ans =
            2
            6
            10
```

Accéder aux éléments des matrices

Exemple(suite):

```
>> A(2:3,:)                    % tous les éléments de la 2ème et la 3ème ligne
      ans =
            5     6     7     8
            9    10    11    12

>> A(1:2,3:4)                  % La sous matrice supérieure droite de taille 2x2
      ans =
            3     4
            7     8

>> A([1,3],[2,4])              % la sous matrice : lignes(1,3) et colonnes (2,4)
      ans =
            2     4
           10    12

>> A(:,3) = []                 % Supprimer la troisième colonne
      A =
            1     2     4
            5     6     8
            9    10    12
```

Accéder aux éléments des matrices

Exemple(suite):

```
>> A(2,:) = []                          % Supprimer la deuxième ligne
      A =
            1     2     4
            9    10    12
```

```
>> A = [A , [0;0]]                      % Ajouter une nouvelle colonne avec A(:,4)=[0;0]
      A =
            1     2     4     0
            9    10    12     0
```

```
>> A = [A ; [1,1,1,1]]                  % Ajouter une nouvelle ligne avec A(3,:)=[1,1,1,1]
      A =
            1     2     4     0
            9    10    12     0
            1     1     1     1
```

Taille d'une matrice

On peut obtenir, avec la fonction size, la taille d'une matrice. La fonction génère un vecteur ligne où le premier élément est le nombre de lignes de la matrice, et le second élément est le nombre de colonnes.

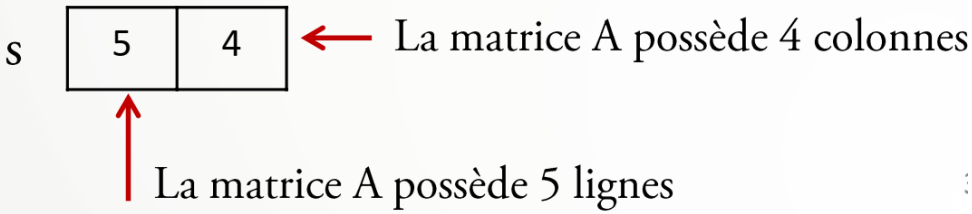
Par exemple, si on a la matrice A suivante

5.7	0.2	6.2	5.1
8.7	7.1	2.4	8.4
1.2	0.4	8.2	6.4
2.3	67	6.5	5.2
2.1	1.8	0.6	7.2

A

Alors que donnera ce code ?

```
>> s = size(A)
```



Taille d'une matrice

Pour obtenir les dimensions séparément on peut utiliser la syntaxe :

```
>> d1 = size (A, 1)          % d1 contient le nombre de ligne (m)
      d1 =
            5

>> d2 = size (A, 2)          % d2 contient le nombre de colonne (n)
      d2 =
            4
```

Taille d'une matrice

La fonction **numel** renvoie le nombre d' éléments d'une matrice passée en paramètre.

Par exemple, si on a la matrice A suivante :

5.7	0.2	6.2	5.1
8.7	7.1	2.4	8.4
1.2	0.4	8.2	6.4
2.3	6.7	6.5	5.2
2.1	1.8	0.6	7.2

A

Alors on aura:

```
>> numel(A)

ans =

                    20
```

Générer de nouvelles matrices

Matlab propose des fonctions permettant de générer de nouvelles matrices.

La fonction	Signification
zeros(n)	Génère une matrice n x n avec tous les éléments = 0
zeros(m,n)	Génère une matrice m x n avec tous les éléments = 0
ones(n)	Génère une matrice n x n avec tous les éléments = 1
ones(m,n)	Génère une matrice m x n avec tous les éléments = 1
eye(n)	Génère une matrice identité de dimension n x n
magic(n)	Génère une matrice magique de dimension n x n
rand(m,n)	Génère une matrice de dimension m x n de valeurs aléatoires

Générer de nouvelles matrices

Remarque:

La fonction magic(n) en Matlab crée une matrice carrée de taille n dont chaque élément est un entier allant de 1 à n^2. De plus la somme des éléments de chaque ligne, chaque colonne et des deux diagonales est égale à la même valeur.

```
>> magic(4)

ans =

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

Opérations de base sur les matrices

Matlab propose tout un ensemble d'opérations usuelles sur les matrices

Symbole	Description	Exemple
+ - *	Les opérations de base (addition, soustraction, produit matriciel). Les tailles des matrices doivent être compatibles	A+B A*B
^	La puissance matricielle (itération du produit matriciel)	A^3
’	Transposée d’une matrice	A’
inv	L’inversion d’une matrice (si son inverse existe)	inv(A)

D’autres opérations peuvent être réalisées sur chaque élément de A

Symbole	Description	Exemple
+ - * /	Réalise l’opération entre un scalaire et chaque élément de la matrice.	5.4*A
.*	Réalise la multiplication terme à terme de deux matrices de même taille.	A.*B
.^	Met à une certaine puissance chaque élément de la matrice	A.^3

Opérations de base sur les matrices

La division des matrices:

./	La division élément par élément
.\	La division inverse élément par élément
/	La division matricielle (A/B) = ($A*B^{-1}$)

Opérations de base sur les matrices

Remarque:

Les opérations élément par éléments sur les matrices sont les mêmes que ceux pour les vecteurs (la seule condition nécessaire pour faire une opération élément par élément est que les deux matrices aient les mêmes dimensions).

Opérations de base sur les matrices

Exemple

```
>> A=ones(2,3)
      A =
            1     1     1
            1     1     1
>> B=zeros(3,2)
      B =
            0     0
            0     0
            0     0
>> B=B+3
B =
     3     3
     3     3
     3     3
>> A*B
      ans =
            9     9
            9     9
>> B=[B , [3 3 3]']            % ou bien B(:,3)=[3 3 3]'
      B =
            3     3     3
            3     3     3
            3     3     3
```

Opérations de base sur les matrices

Exemple(suite)

```
>> B=B(1:2,:)                    % ou bien B(3,:)=[]
      B =
            3     3     3
            3     3     3
>> A=A*2
      A =
            2     2     2
            2     2     2

>> A.*B
      ans =
            6     6     6
            6     6     6
>> A*eye(3)
      ans =
            2     2     2
            2     2     2
```

Opérations de base sur les matrices

II est aussi possible d'appeler des fonctions usuelles **(sqrt, cos, sin, exp, ...)** sur une matrice, afin qu elles s'exécutent sur chaque élément de la matrice.

Par exemple, si on a cette matrice A :

4	25	36
2	9	49

A

Alors on aura :

2	5	6
1.4	3	7

sqrt(A)

Opérations avancées sur les matrices

On peut calculer la somme des éléments d'une matrice, le long des lignes ou des colonnes, avec la fonction **sum**

Par exemple, si on a cette matrice A :

2.6	3.9	10
1.0	-3	3.2

A

Alors **sum(A,l)** calculera la somme des éléments de A le long des colonnes, tandis que **sum(A,2)** calculera la somme le long des lignes :

3.6	0.9	13.2

sum(A,1)

16.5
1.2

sum(A,2)

prod fait le même travail, mais en faisant le produit des éléments.

Opérations avancées sur les matrices

A vous de jouer : on considère la matrice A ci-dessous. Que donne la commande ci-dessous comme résultat ?

```
>> b = sum(A,2)
```

3	2	1	8
4	1	3	6
2	2	0	5

A

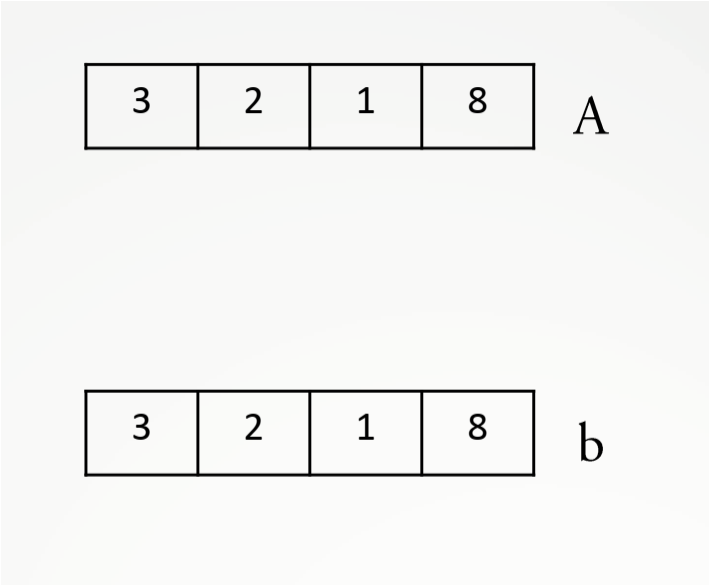
14
14
9

b

Opérations avancées sur les matrices

A vous de jouer : on considère la matrice A ci-dessous. Que donne la commande ci-dessous comme résultat ?

```
>> b = sum(A,1)
```



Opérations avancées sur les matrices

Comment réaliser la somme de tous les éléments d'une matrice A ?

```
>> b = sum(A,1)
>> c = sum(b,2)
```

Ou, plus court

```
>> c = sum(sum(A,2),1)
```

Opérations avancées sur les matrices

On peut aussi, avec les fonctions **min** et **max,** calculer les éléments maximum ou minimum le long des lignes ou des colonnes d'une matrice.

Par exemple, si on a cette matrice A :

2.6	3.9	10
1.0	-3	3.2

A

Alors **max(A, [ ], 1)** calculera les éléments maximaux de A le long des colonnes, tandis que **max(A, [ ], 2)** calculera les éléments maximaux le long des lignes :

2.6	3.9	10

max(A,[],1)

10
3.2

max(A,[],2)

Concaténation de matrices

On peut générer de nouvelles matrices en concaténant d'anciennes matrices.

Si l'on a ces matrices :

4	5	6
7	1	1
2	8	7

A

4	3
1	2
5	8

B

5	4	4

C

On peut les concaténer de la même façon que l'on déclare les matrices :

```
>> D = [A B]
>> E = [A ; C]
```

Et on a :

4	5	6	4	3
7	1	1	1	2
2	8	7	5	8

D

4	5	6
7	1	1
2	8	7
5	4	4

E

Concaténation de matrices

Exemple de génération de matrices par concaténation de vecteurs:

Soit les vecteurs suivants:

```
>> x = 1:4                                  % création d'un vecteur x
      x =
            1     2     3     4
>> y = 5:5:20                               % création d'un vecteur y
      y =
            5    10    15    20
>> z = 4:4:16                               % création d'un vecteur z
      z =
            4     8    12    16
```

Concaténation de matrices

Exemple de génération de matrices par concaténation de vecteurs:

```
>> A = [x ; y ; z]                % A est formée par les vecteurs lignes x, y et z
      A =
            1     2     3     4
            5    10    15    20
            4     8    12    16

>> B = [x' y' z']                 % B est formée par les vecteurs colonnes x, y et z
      B =
            1     5     4
            2    10     8
            3    15    12
            4    20    16

>> C = [x ; x]                    % C est formée par le même vecteur ligne x 2 fois
      C =
            1     2     3     4
            1     2     3     4
```

Fonctions utiles

La fonction	L'utilité	Exemple d'utilisation

det	Calcule le déterminant d'une matrice	`>> A = [1,2;3,4] ;`
		`>> det(A)`
		`ans =`
		`-2`

inv	Calcule l'inverse d'une matrice	`>> inv(A)`
		`ans =`
		`-2.0000 1.0000`
		`1.5000 -0.5000`

trace	Calcule la trace d'une matrice	`>> trace(A)`
		`ans =`
		`5`

dot	Calcule le produit scalaire de 2 vecteurs	`>> v = [-1,5,3];`
		`>> u = [2,-2,1];`
		`>> dot(u,v)`
		`ans =`
		`-9`

Remarque

La trace d'une matrice est la somme de ses éléments diagonaux.

Fonctions utiles

`norm`	Calcule la norme d'un vecteur	`>> norm(u)`
		`ans =`
		`3`

`cross`	Calcule le produit vectoriel de 2 vecteurs	`>> cross(u,v)`
		`ans =`
		`-11 -7 8`

`diag`	Renvoie le diagonal d'une matrice	`>> diag(A)`
		`ans =`
		`1`
		`4`

`diag(V)`	Crée une matrice ayant le vecteur V dans le diagonal et 0 ailleurs.	`>> V = [-5,1,3]`
		`>> diag(V)`
		`ans =`
		`-5 0 0`
		`0 1 0`
		`0 0 3`

Fonctions utiles

tril	Renvoie la partie triangulaire inferieure	`>> B=[1,2,3;4,5,6;7,8,9]`
		`B =`
		`1 2 3`
		`4 5 6`
		`7 8 9`
		`>> tril(B)`
		`ans =`
		`1 0 0`
		`4 5 0`
		`7 8 9`
		`>> tril(B,-1)`
		`ans =`
		`0 0 0`
		`4 0 0`
		`7 8 0`
		`>> tril(B,-2)`
		`ans =`
		`0 0 0`
		`0 0 0`
		`7 0 0`

Fonctions utiles

triu	Renvoie la partie triangulaire supérieure	`>> triu(B)`
		`ans =`
		`1 2 3`
		`0 5 6`
		`0 0 9`
		`>> triu(B,-1)`
		`ans =`
		`1 2 3`
		`4 5 6`
		`0 8 9`
		`>> triu(B,1)`
		`ans =`
		`0 2 3`
		`0 0 6`
		`0 0 0`

Comparaison des matrices

La comparaison des vecteurs et des matrices diffère quelque peu des scalaires, d'où l'utilité des deux fonctions **'isequal' et 'isempty' (qui permettent de donner une réponse** concise pour la comparaison).

La fonction	Description
isequal	teste si deux (ou plusieurs) matrices sont égales (ayant les mêmes éléments partout). Elle renvoie **1** si c'est le cas, et **0** sinon.
isempty	teste si une matrice est vide (ne contient aucun élément). Elle renvoie **1** si c'est le cas, et **0** sinon.

Comparaison des matrices

Exemple

```
>> A = [5,2;-1,3]                    % Créer la matrice A
        A =
             5     2
            -1     3
>> B = [5,1;0,3]                     % Créer la matrice B
        B =
             5     1
             0     3
>> A==B                              % Tester si A=B ? (1 ou 0 selon la position)
       ans =
             1     0
             0     1
>> isequal(A,B)                      % Tester si effectivement A et B sont égales (les mêmes)
       ans =
             0
>> C=[] ;                            % Créer la matrice vide C
>> isempty(C)                        % Tester si C est vide (affiche vrai = 1)
       ans =
             1
>> isempty(A)                        % Tester si A est vide (affiche faux = 0)
       ans =
             0
```