



---

# MATLAB

---

Outils de programmation pour les mathématiques



2017-2018

UNIVERSITE DE GHARDAÏA  
Département de mathématique et d'Informatique

# 1. Introduction

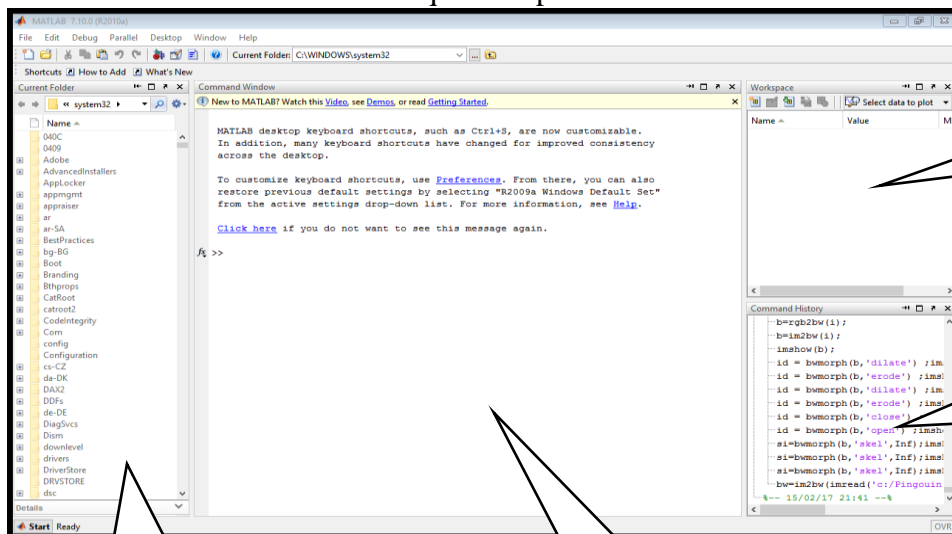
Matlab est un logiciel qui permet d'effectuer des calculs numériques sans avoir besoin de programmer dans un langage traditionnel de type C, C++, Fortran ou Pascal. Matlab signifie *Matrix laboratory*. L'élément de base est la matrice et les fonctions sont optimisées pour s'exécuter rapidement sur ce type d'objet. En particulier, un scalaire est une matrice de taille  $1 \times 1$ .

C'est un langage interprété, ce qui signifie que l'on peut taper des commandes qui s'exécutent directement sans avoir à passer par une phase de compilation. Néanmoins, il est possible d'écrire des programmes qui sont alors une suite de commandes que l'on met dans un fichier.

A la différence de Maple, les instructions sont effectuées les unes après les autres. Il n'est pas possible de revenir en arrière.

Pour démarrer Matlab, il faut taper *matlab* dans une ligne de commande du terminal.

Une fenêtre Matlab se divise en quatre espaces affichés :



**Current Directory :**  
qui affiche les dossiers  
et fichiers du répertoire  
courant

**Command Window :**  
où s'affichent les  
commandes exécutées  
et leur résultat

**Workspace :** qui  
recense les variables  
affectées

**Command History :**  
où s'affiche  
l'historique des  
commandes

AhnvdaazdddfDDFyig gdjo

Commande	Effet
↑ ou Ctrl + P	rappeler la ligne précédente
↓ ou Ctrl + N	rappeler la ligne suivante
← ou Ctrl + B	déplacer le curseur vers la gauche
→ ou Ctrl + F	déplacer le curseur vers la droite
Ctrl + L	déplacer le curseur d'un mot vers la gauche
Ctrl + R	déplacer le curseur d'un mot vers la droite
Ctrl + A	placer le curseur en début de ligne
Ctrl + E	placer le curseur en fin de ligne
Ctrl + U	effacer la ligne courante
Ctrl + T	passer du mode insertion au mode écrasement et vice-versa
Ctrl + D	effacer le caractère sous le curseur
Ctrl + K	effacer la fin de la ligne après le curseur

**Remarque :** en tapant : *mat* ↑,

la dernière ligne commençant par *mat* s'affiche.

## 2. Première interaction avec MATLAB

Le moyen le plus simple d'utiliser MATLAB est d'écrire directement dans la fenêtre de commande (Command Window) juste après le curseur (prompt) `>>`

Pour calculer une expression mathématique il suffit de l'écrire comme ceci :

```
>> 5+6      Puis on clique sur la touche Entrer pour voir le résultat
```

```
ans = 11
```

Si nous voulons qu'une expression soit calculée mais sans afficher le résultat, on ajoute un point-virgule ';' à la fin de l'expression comme suit :

```
>> 5+6 ;
```

```
>>
```

Pour créer une variable on utilise la structure simple : '**variable = définition**' sans se préoccuper du type de la variable.

Par exemple :

```
>> a = 10 ;
```

```
>> u = cos(a) ;
```

```
>> v = sin(a) ;
```

```
>> u^2+v^2
```

```
ans = 1
```

```
>> ans+10
```

```
ans = 11
```

```
>>
```

Lorsqu'on n'utilise pas des variables, le résultat de la commande est automatiquement affecté à la variable `ans` qui peut être par la suite utilisée comme une variable normale.

Il est possible d'écrire plusieurs expressions dans la même ligne en les faisant séparées par des virgules ou des points virgules. Par exemple :

```
>> 5+6, 2*5-1, 12-4
```

```
ans = 11
```

```
ans = 9
```

```
ans = 8
```

```
>> 5+6; 2*5-1, 12-4;
```

```
ans = 9
```

```
>>
```

Le nom d'une variable ne doit contenir que des caractères alphanumériques ou le symbole '\_' (underscore), et doit commencer par un alphabet. Nous devons aussi faire attention aux majuscules car le MATLAB est sensible à la casse (**A** et **a** sont deux identifiants différents). Les opérations de base dans une expression sont résumées dans le tableau suivant :

L'opération	La signification
+	L'addition
-	La soustraction
*	La multiplication
/	La division
\	La division gauche (ou la division inverse)
^	La puissance
'	Le transposé
( et )	Les parenthèses spécifient l'ordre d'évaluation

Pour voir la liste des variables utilisées, soit on regarde à la fenêtre '**Workspace**' soit on utilise les commandes '**whos**' ou '**who**'.

**whos** donne une description détaillée (le nom de la variable, son type et sa taille), par contre **who** donne juste les noms des variables.

Par exemple, dans ce cours on a utilisé 3 variables **a**, **u** et **v** :

```
>> who
```

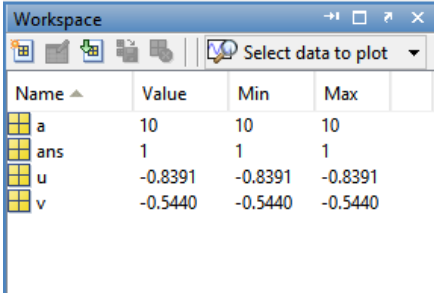
Your variables are:

```
a    ans    u    v
```

```
>> whos
```

Name	Size	Bytes	Class	Attributes
a	1x1	8	double	
ans	1x1	8	double	
u	1x1	8	double	
v	1x1	8	double	

L'utilisation de ces deux commandes peut être omise car des informations sur les variables sont visibles directement dans la fenêtre workspace



Name	Value	Min	Max
a	10	10	10
ans	1	1	1
u	-0.8391	-0.8391	-0.8391
v	-0.5440	-0.5440	-0.5440

## 2.1. Les nombres en MATLAB

MATLAB utilise une notation décimale conventionnelle, avec un point décimal facultatif '.' et le signe '+' ou '-' pour les nombres signés. La notation scientifique utilise la lettre 'e' pour spécifier le facteur d'échelle en puissance de 10.

Les nombres complexes utilisent les caractères 'i' et 'j' (indifféremment) pour désigner la partie imaginaire. Le tableau suivant donne un résumé :

Le type	Exemples
Entier	5 -83
Réel en notation décimale	0.0205 3.1415926
Réel en notation scientifique	1.60210e-20 6.02252e23 (1.60210x10-20 et 6.02252x1023)
Complexe	5+3i -3.14159j

MATLAB utilise toujours les nombres réels (double précision) pour faire les calculs, ce qui permet d'obtenir une précision de calcul allant jusqu'à 16 chiffres significatifs. Mais il faut noter les points suivants :

- Le résultat d'une opération de calcul est par défaut affichée avec quatre chiffres après la virgule.
- Pour afficher davantage de chiffres utiliser la commande **format long** (14 chiffres après la virgule).
- Pour retourner à l'affichage par défaut, utiliser la commande **format short**.
- Pour afficher uniquement 02 chiffres après la virgule, utiliser la commande **format bank**.
- Pour afficher les nombres sous forme d'une ration, utiliser la commande **format rat**.

La commande	Signification
format short	affiche les nombres avec 04 chiffres après la virgule
format long	affiche les nombres avec 14 chiffres après la virgule
format bank	affiche les nombres avec 02 chiffres après la virgule
format short e	point flottant, affiche les nombres avec 04 chiffres et l'exposant
format long e	point flottant, affiche les nombres avec 14 chiffres et l'exposant
format rat	affiche les nombres sous forme d'une ration (a/b)

Exemple :

```
>> 8/3
```

```
ans =2.6667
```

```
>> format long
```

```
>> 8/3
```

```

ans =2.666666666666667
>> format bank
>> 8/3
ans =2.67
>> format short
>> 8/3
ans =2.6667
>> 7.2*3.1
ans =22.3200
>> format rat
>> 7.2*3.1
ans =558/25
>> 2.6667
ans =26667/10000

```

La fonction **vpa** peut être utilisé afin de forcer le calcul de présenter plus de décimaux significatifs en spécifiant le nombre de décimaux désirés.

*Exemple :*

```

>> sqrt(2)
ans =1.4142
>> vpa(sqrt(2),50)
ans =1.4142135623730950488016887242096980785696718753769

```

## 2.2. Les principales constantes, fonctions et commandes

MATLAB définit les constantes suivantes :

La constante	Sa valeur
pi	$\pi=3.1415\dots$
exp(1)	$e=2.7183\dots$
i	$=\sqrt{-1}$
j	$=\sqrt{-1}$
Inf	$\infty$
NaN	Not a Number (Pas un numéro)
eps	$\varepsilon \approx 2 \times 10^{-16}$
realmax	la valeur maximale absolue des réels
realmin	la valeur minimale absolue des réels

Parmi les fonctions fréquemment utilisées, on peut noter les suivantes :

La fonction	Sa signification
sin(x)	le sinus de x (en radian)
cos(x)	le cosinus de x (en radian)
tan(x)	le tangent de x (en radian)
asin(x)	l'arc sinus de x (en radian)
acos(x)	l'arc cosinus de x (en radian)
atan(x)	l'arc tangent de x (en radian)
sqrt(x)	la racine carrée de x $\rightarrow \sqrt{\phantom{x}}$
abs(x)	la valeur absolue de x $\rightarrow  x $
exp(x)	$= e^x$
log(x)	logarithme naturel de x $\rightarrow \ln(x)=\log_e(x)$
log10(x)	logarithme à base 10 de x $\rightarrow \log_{10}(x)$
imag(x)	la partie imaginaire du nombre complexe x
real(x)	la partie réelle du nombre complexe x
round(x)	arrondi un nombre vers l'entier le plus proche
floor(x)	arrondi un nombre vers l'entier le plus petit $\rightarrow \min \{n n \leq x, n \text{ entier}\}$
ceil(x)	arrondi un nombre vers l'entier le plus grand $\rightarrow \max \{n n \geq x, n \text{ entier}\}$

MATLAB offre beaucoup de commandes pour l'interaction avec l'utilisateur.

La commande	Sa signification
who	Affiche le nom des variables utilisées
whos	Affiche des informations sur les variables utilisées
clear x y	Supprime les variables x et y
clear, clear all	Supprime toutes les variables
clc	Efface l'écran des commandes
exit, quit	Fermer l'environnement MATLAB

### 2.3. La priorité des opérations dans une expression

L'évaluation d'une expression s'exécute de gauche à droite en considérant la priorité des opérations indiquée dans le tableau suivant :

Les opérations	La priorité (1=max, 4=min)
Les parenthèses (et)	1
La puissance et le transposé ^ et '	2
La multiplication et la division * et /	3
L'addition et la soustraction + et -	4

Par exemple  $5+2*3 = 11$  et  $2*3^2 = 18$

### 2.4. Les Coordonnées Polaires :

Un complexe est généralement représenté sous la *forme algébrique* ou *cartésienne*. Il peut également être représenté sous la forme polaire  $pe^{i\theta}$ , où p est son module et  $\theta$  son argument.

```
>> c = 2 + 2i                                % forme algébrique ou cartésienne
c =
    2.0000 + 2.0000i
>> theta = angle(c)                          % angle de c
theta =
    0.7854
>> ro = abs(c)                               % module de c
ro =
    2.8284
>> ro*exp(theta*i)                          % forme polaire
ans =
    2.0000 + 2.0000i
>> [theta, ro] = cart2pol(2,2)               % conversion au polaire
theta =
    0.7854
ro =
    2.8284
>> [x y] = pol2cart(theta, ro)               % conversion au cartésien
x =
    2.0000
y =
    2
```

*Exercice récapitulatif :*

Créer une variable x et donnez-la la valeur 2, puis écrivez les expressions suivantes :

- $3x_3 - 2x_2 + 4x$
- $\frac{e^{1+x}}{1-\sqrt{2x}}$
- $|\sin^{-1}(2x)|$
- $\frac{\ln(x)}{2x^3} - 1$

La solution :

```
>> x=2 ;
>> 3*x^3-2*x^2+4*x ;
>> exp(1+x)/(1-sqrt(2*x)) ;
>> abs(asin(2*x)) ;
>> log(x)/(2*x^3)-1 ;
```

### 3. Les vecteurs et les matrices

MATLAB était conçu à l'origine pour permettre aux mathématiciens, scientifiques et ingénieurs d'utiliser facilement les mécanismes de l'algèbre linéaire. Par conséquent, l'utilisation des vecteurs et des matrices est très intuitive et commode en MATLAB.

MATLAB traite un seul type d'objet : les matrices !

- Les scalaires sont des matrices 1 x 1.
- Les vecteurs lignes sont des matrices 1 x n.
- Les vecteurs colonnes sont des matrices n x 1.

#### 3.1. Vecteur ligne

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

```
>> V = [ 5 , 2 , 13 , -6 ]           % Création d'un vecteur ligne V
```

V =

5      2      13      -6

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

U =

4      -2      1

Un vecteur ligne peut être aussi créé par description de la valeur initiale, l'incrément, et la valeur finale :

```
>> v = [1 : 1 : 10]
```

v =

1      2      3      4      5      6      7      8      9      10

```
>> v = [1 : 0.5 : 8]
```

v =

Columns 1 through 9

1.0000   1.5000   2.0000   2.5000   3.0000   3.5000   4.0000   4.5000   5.0000

Columns 10 through 15

5.5000   6.0000   6.5000   7.0000   7.5000   8.0000

```
>> v = [0 : pi/10: pi]
```

v =

Columns 1 through 9

0 0.3142   0.6283   0.9425   1.2566   1.5708   1.8850   2.1991   2.5133

Columns 10 through 11

2.8274   3.1416

La fonction *linspace*(i, j, k) génère un vecteur de k éléments allant de i à j :

***linspace*** (début, fin, nombre d'éléments)

Le pas d'incrémentation est calculé automatiquement par MATLAB selon la formule :

$$\text{le pas} = \frac{\text{fin} - \text{début}}{\text{nombre d'éléments} - 1}$$

```
>> v = linspace(1, 10, 10)
```

v =

1 2 3 4 5 6 7 8 9 10

La fonction *logspace*(i, j, k) génère un vecteur de k éléments allant de  $10^i$  à  $10^j$  :

```
v = logspace(0, 2, 5)
```

v =

1.0000	3.1623	10.0000	31.6228	100.0000
% 10^0,	10^0.5,	10^1,	10^1.5,	10^2

### 3.2. Vecteur colonne

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

1. Écrire les composants du vecteur entre crochets [ et ] et de les séparer par des points-virgules (;) comme suit :

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

% Création d'un vecteur colonne U

U =

4  
-2  
1

2. Écrire verticalement le vecteur :

```
>> 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

Si les composants d'un vecteur **X** sont ordonnés avec des valeurs consécutives, nous pouvons le noter avec la notation suivante :

**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



Si les composants d'un vecteur **X** sont ordonnés avec des valeurs consécutives mais avec un pas (d'incrément/décroissement) différent de 1, nous pouvons spécifier le pas avec la notation : **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
```

On peut écrire des expressions plus complexes comme :

```
>> V = [ 1:2:5 , -2:2:1 ]
```

```
V =
    1    3    5   -2    0
```

```
>> A = [1 2 3]
```

```
A =
    1    2    3
```

```
>> B = [A, 4, 5, 6]
```

```
B =
    1    2    3    4    5    6
```

Les fonctions **zeros** et **ones** initialisent des vecteurs lignes (**zeros(1, k)** et **ones (1, k)**) ou colonnes (**zeros(k, 1)** et **ones(k, 1)**) par des zéros ou des uns :

```
>> v = zeros(1,5)
```

```
v =
    0    0    0    0    0
```

```
>> v = zeros(5,1)
```

```
v =
    0
    0
    0
    0
    0
```

```
>> v = ones(1,5)
```

```
v =
    1    1    1    1    1
```

```
>> v = ones(5,1)
```

```
v =
    1
    1
    1
    1
    1
```

La fonction **rand** initialise des vecteurs lignes (**rand(1, k)**) ou colonnes (**rand(1, k)**) par des valeurs aléatoires entre 0 et 1 :

```
>> v1 = rand(1, 5) % 5 valeurs aléatoires (0-1)
```

```
v1 =
```

```

0.0975  0.2785  0.5469  0.9575  0.9649
>> v2 = rand(1, 5)           % 5 autres valeurs aléatoires (0-1)
v2 =
0.1576  0.9706  0.9572  0.4854  0.8003
>> v3 = 10 * rand(1, 5)      % 5 valeurs aléatoires (0-10)
v3 =
1.4189  4.2176  9.1574  7.9221  9.5949

```

### 3.3. Référencement et accès aux éléments d'un vecteur

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

*nom\_vecteur(positions)*

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

*Exemples :*

```

>> V = [5, -1, 13, -6, 7]      % création du vecteur V qui contient 5 éléments
V =
5  -1  13  -6  7
>> V(3)                       % la 3ème position
ans =
13
>> V(2:4)                     % de la deuxième position jusqu'au quatrième
ans =
-1  13  -6
>> V(4:-2:1)                  % de la 4ème pos jusqu'à la 1ère avec le pas = -2
ans =
-6  -1
>> V(3:end)                   % de la 3ème position jusqu'à la dernière
ans =
13  -6  7
>> V([1,3,4])                 % la 1ère, la 3ème et la 4ème position uniquement
ans =
5  13  -6
>> V(1) = 8                   % donner la valeur 8 au premier élément
V =
8  -1  13  -6  7
>> V(6) = -3                  % ajouter un sixième élément avec la valeur -3
V =
8  -1  13  -6  7  -3
>> V(9) = 5                   % ajouter un neuvième élément avec la valeur 5
V =
8  -1  13  -6  7  -3  0  0  5
>> V(2) = []                  % Supprimer le deuxième élément
V =
8  13  -6  7  -3  0  0  5
>> V(3:5) = []                % Supprimer du 3ème jusqu'au 5ème élément
V =
8  13  0  0  5

```

### 3.4. 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 :

L'opération	Signification	Exemple avec : >> <b>u</b> = [-2, 6, 1]; >> <b>v</b> = [ 3, -1, 4];
+	Addition des vecteurs	>> <b>u+2</b> ans = 0 8 3 >> <b>u+v</b> ans = 1 5 5
-	Soustraction des vecteurs	>> <b>u-2</b> ans = -4 4 -1 >> <b>u-v</b> ans = -5 7 -3
.*	Multiplication élément par élément	>> <b>u*2</b> ans = -4 12 2 >> <b>u.*2</b> ans = -4 12 2 >> <b>u.*v</b> ans = -6 -6 4
./	Division élément par élément	>> <b>u/2</b> ans = -1.0000 3.0000 0.5000 >> <b>u./2</b> ans = -1.0000 3.0000 0.5000 >> <b>u./v</b> ans = -0.6667 -6.0000 0.2500
.^	Puissance élément par élément	>> <b>u.^2</b> ans = 4 36 1 >> <b>u.^v</b> ans = -8.0000 0.1667 1.0000

L'écriture d'une expression tel que : **u^2** génère une erreur car cette expression réfère à une multiplication de matrices (**u\*u** doit être réécrite **u\*u'** ou **u'\*u** pour être valide).

### 3.5. Opérations Statistiques

```
>> x = [2 4 5 6 7];
>> max(x)           % maximum
    ans =
        7
>> min(x)           % minimum
    ans =
        2
>> sum(x)            % somme des valeurs
    ans =
       24
>> prod(x)           % produit des valeurs
    ans =
```

```

1680
>> mean(x)          % moyenne des valeurs
ans =
4.8000
>> median(x)        % la valeur médiane (après le tri)
ans =
5
>> var(x)           % la variance des valeurs
ans =
3.7000
>> std(x)           % l'écart type des valeurs
ans =
1.9235

```

### 3.6. Les matrices

Considérant la matrice suivante :

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

Cette matrice peut être écrite 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]] ;

```

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. Par exemple :

```

>> X = [1 2 ; 4 5 6]
Error using vertcat
CAT arguments dimensions are not consistent.

```

Une matrice peut être générée par des vecteurs comme le montre les exemples 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
>> 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

```

```

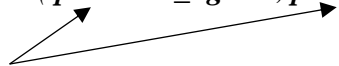
2 10 8
3 15 12
4 20 16
>> C = [x ; x]      % C est formée par le même vecteur ligne x 2 fois
C =
1 2 3 4
1 2 3 4

```

### 3.7. Référencement et accès aux éléments d'une matrice

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)

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

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

Il est utile de noter les possibilités suivantes :

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

*Exemples* :

```

>> 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

>> 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
>> 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

```

Les dimensions d'une matrice peuvent être acquises en utilisant la fonction `size`. Cependant, avec une matrice **A** de dimension  $m \times n$  le résultat de cette fonction est un vecteur de deux composants, une pour  $m$  et l'autre pour  $n$ .

```

>> d = size(A)
       d =
        3     4

```

Ici, la variable **d** contient les dimensions de la matrice **A** sous forme d'un vecteur.

Pour obtenir les dimensions séparément on peut utiliser la syntaxe :

```

>> d1 = size (A, 1)       % d1 contient le nombre de ligne (m)
       d1 =
        3
>> d2 = size (A, 2)       % d2 contient le nombre de colonne (n)
       d2 =
        4

```

### 3.8. Génération automatique des matrices

En MATLAB, il existe des fonctions qui permettent de générer automatiquement des matrices particulières. Le tableau suivant présente quelques-uns :

La fonction	Signification
<code>zeros(n)</code>	Génère une matrice $n \times n$ avec tous les éléments = 0
<code>zeros(m,n)</code>	Génère une matrice $m \times n$ avec tous les éléments = 0
<code>ones(n)</code>	Génère une matrice $n \times n$ avec tous les éléments = 1
<code>ones(m,n)</code>	Génère une matrice $m \times n$ avec tous les éléments = 1
<code>eye(n)</code>	Génère une matrice identité de dimension $n \times n$
<code>magic(n)</code>	Génère une matrice magique de dimension $n \times n$
<code>rand(m,n)</code>	Génère une matrice de dimension $m \times n$ de valeurs aléatoires
<code>randi(max,m,n)</code>	Génère une matrice de dimension $m \times n$ de valeurs aléatoires entre 1 et max
<code>Repmat(k,m,n)</code>	Génère une matrice de dimension $m \times n$ de valeurs dupliqués de k
<code>reshape(Vec,m,n)</code>	Génère une matrice de dimension $m \times n$ à partir du vecteur Vec

### 3.9. Les opérations de base sur les matrices

L'opération	Signification
+	L'addition
-	La soustraction
.*	La multiplication élément par élément
./	La division élément par élément
.\	La division inverse élément par élément
.^	La puissance élément par élément
*	La multiplication matricielle
/	La division matricielle $(A/B) = (A*B^{-1})$

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). Par contre la multiplication ou la division des matrices requiert quelques contraintes.

*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
>> 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

>> A = repmat(1, 4, 3)
A =
    1    1    1
    1    1    1
    1    1    1
    1    1    1

>> A = rand(3, 3) %des valeurs aléatoires entre 0 et 1
A =
    0.9575    0.9706    0.8003
    0.9649    0.9572    0.1419
    0.1576    0.4854    0.4218

>> A = randi(10, 3, 3) %des valeurs entiers aléatoires entre 1 et 10
A =
    10     7    10
     8     1     7
    10     9     8

>> V = [1 2 3 4 5 6 7 8 9 10 11 12]
% vecteur ligne de 12 éléments
V =
    1 2 3 4 5 6 7 8 9 10 11 12 11 12

>> A = reshape(V, 3, 4) % matrice identité de 3x4
A =
     1     4     7    10
     2     5     8    11
     3     6     9    12

>> A = reshape(V, 2, 6) % matrice identité de 2x6
A =
     1     3     5     7     9    11
     2     4     6     8    10    12

```

### 3.10. Fonctions utiles pour le traitement des matrices

Voici quelques fonctions parmi les plus utilisées concernant les matrices :

La fonction	L'utilité	Exemple d'utilisation
<b>det</b>	Calcule de déterminant d'une matrice	<pre>&gt;&gt; A = [1,2;3,4] ; &gt;&gt; det(A) ans =     -2</pre>
<b>inv</b>	Calcule l'inverse d'une matrice	<pre>&gt;&gt; inv(A) ans =    -2.0000    1.0000     1.5000   -0.5000</pre>
<b>rank</b>	Calcule le rang d'une matrice	<pre>&gt;&gt; rank(A) ans =      2</pre>
<b>trace</b>	Calcule la trace d'une matrice	<pre>&gt;&gt; trace(A) ans =      5</pre>
<b>eig</b>	Calcule les valeurs propres	<pre>&gt;&gt; eig(A) ans =    -0.3723     5.3723</pre>



<b>dot</b>	Calcule le produit scalaire de 2 vecteurs	<pre>&gt;&gt; v = [-1,5,3]; &gt;&gt; u = [2,-2,1]; &gt;&gt; dot(u,v) ans = -9</pre>
<b>norm</b>	Calcule la norme d'un vecteur	<pre>&gt;&gt; norm(u) ans = 3</pre>
<b>cross</b>	Calcule le produit vectoriel de 2 vecteurs	<pre>&gt;&gt; cross(u,v) ans = -11 -7 8</pre>
<b>diag</b>	Renvoie le diagonal d'une matrice	<pre>&gt;&gt; diag(A) ans = 1 4</pre>
<b>diag(V)</b>	Crée une matrice ayant le vecteur V dans le diagonal et 0 ailleurs.	<pre>&gt;&gt; V = [-5,1,3] &gt;&gt; diag(V) ans = -5 0 0 0 1 0 0 0 3</pre>
<b>tril</b>	Renvoie la partie triangulaire inferieure	<pre>&gt;&gt; B=[1,2,3;4,5,6;7,8,9] B = 1 2 3 4 5 6 7 8 9 &gt;&gt; tril(B) ans = 1 0 0 4 5 0 7 8 9 &gt;&gt; tril(B,-1) ans = 0 0 0 4 0 0 7 8 0 &gt;&gt; tril(B,-2) ans = 0 0 0 0 0 0 7 0 0</pre>
<b>triu</b>	Renvoie la partie triangulaire supérieure	<pre>&gt;&gt; triu(B) ans = 1 2 3 0 5 6 0 0 9 &gt;&gt; triu(B,-1) ans = 1 2 3 4 5 6 0 8 9 &gt;&gt; triu(B,1) ans = 0 2 3 0 0 6 0 0 0</pre>

*Exemple :*

```
>> A = [1 2 1; 1 1 2; 2 2 1]
A =
     1     2     1
     1     1     2
     2     2     1
>> det(A)           % déterminant
ans =
     3
>> size(A)          % taille (lignes , colonnes)
ans =
     3     3
>> diag(A)          % le vecteur du diagonal
ans =
     1
     1
     1
>> trace(A)         % la trace (somme su diagonal)
ans =
     3
>> triu(A)          % matrice triangulaire supérieure
ans =
     1     2     1
     0     1     2
     0     0     1
>> tril(A)          % matrice triangulaire inférieure
ans =
     1     0     0
     1     1     0
     2     2     1
>> A'               % le transposé
ans =
     1     1     2
     2     1     2
     1     2     1
>> A_1 = inv(A)      % l'inverse
A_1 =
    -1.0000     0     1.0000
     1.0000    -0.3333    -0.3333
     0     0.6667    -0.3333
>> A*A_1
ans =
     1.0000     0    -0.0000
     0     1.0000     0
     0     0     1.0000
```

#### 4. Les graphiques et la visualisation des données en MATLAB

MATLAB offre un puissant système de visualisation qui permet la présentation et l'affichage graphique des données d'une manière à la fois efficace et facile.

Nous allons voir dans cette partie les principes de base indispensables pour dessiner des courbes en MATLAB.

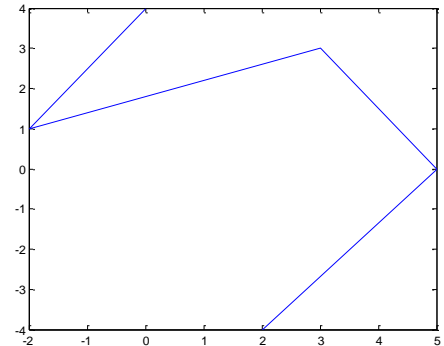
## 4.1. La fonction plot

La fonction **plot** est utilisable avec des vecteurs ou des matrices. Elle trace des lignes en reliant des points de coordonnées définies dans ses arguments, et elle à plusieurs formes :

- a) **Si elle contient deux vecteurs de la même taille comme arguments** : elle considère les valeurs du premier vecteur comme les éléments de l'axe X (les abscisses), et les valeurs du deuxième vecteur comme les éléments de l'axe Y (les ordonnées).

*Exemple :*

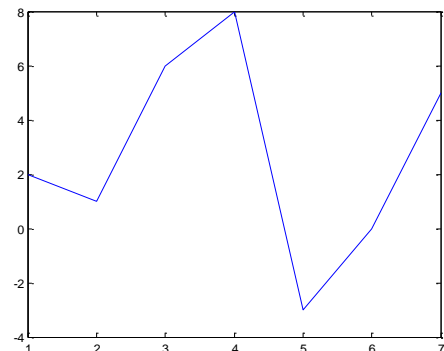
```
>> A = [2, 5, 3, -2, 0]
A =
     2     5     3    -2     0
>> B = [-4, 0, 3, 1, 4]
B =
    -4     0     3     1     4
>> plot(A, B)
```



- b) **Si elle contient un seul vecteur comme argument** : elle considère les valeurs du vecteur comme les éléments de l'axe Y (les ordonnées), et leurs positions relatives définissent l'axe X (les abscisses).

*Exemple :*

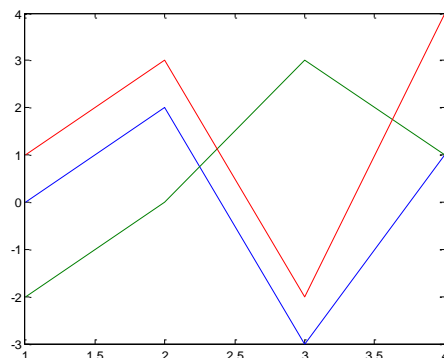
```
>> V = [2, 1, 6, 8, -3, 0, 5]
V =
     2     1     6     8    -3     0     5
>> plot(V)
```



- c) **Si elle contient une seule matrice comme argument** : elle considère les valeurs de chaque colonne comme les éléments de l'axe Y, et leurs positions relatives (le numéro de ligne) comme les valeurs de l'axe X. Donc, elle donnera plusieurs courbes (une pour chaque colonne).

*Exemple :*

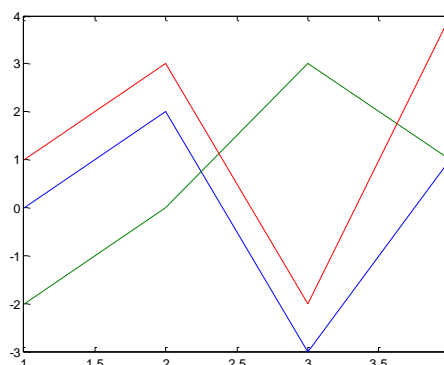
```
>> M = [0 -2 1; 2 0 3; -3 3 -2; 1 1 4]
M =
     0     -2     1
     2      0     3
    -3      3    -2
     1      1     4
>> plot(M)
```



- d) **Si elle contient deux matrices comme arguments** : elle considère les valeurs de chaque colonne de la première matrice comme les éléments de l'axe X, et les valeurs de chaque colonne de la deuxième matrice comme les valeurs de l'axe Y.

*Exemple :*

```
>> K = [1 1 1; 2 2 2; 3 3 3; 4 4 4]
K =
     1     1     1
     2     2     2
     3     3     3
     4     4     4
>> M = [0 -2 1; 2 0 3; -3 3 -2; 1 1 4]
```

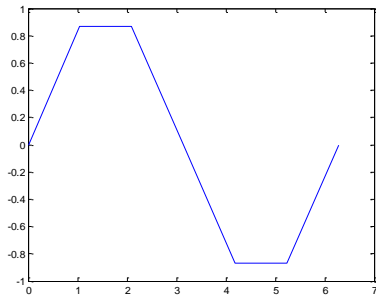


```

M =
0      -2      1
2       0      3
-3       3     -2
1       1      4
>> plot(K,M)

```

Plus le nombre de coordonnées augmente plus la courbe devienne précise. Par exemple pour dessiner la courbe de la fonction  $y = \sin(x)$  sur  $[0, 2\pi]$  on peut écrire :

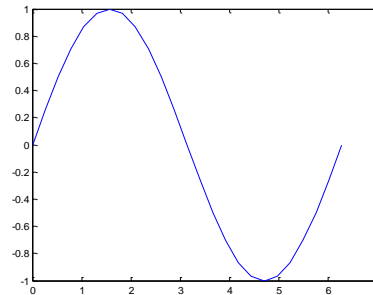


La première figure (le pas =  $\pi/3$ )

```

>> x = 0:pi/3:2*pi;
>> y = sin(x);
>> plot(x,y)

```



La deuxième figure (le pas =  $\pi/12$ )

```

>> x = 0:pi/12:2*pi;
>> y = sin(x);
>> plot(x,y)

```

## 4.2. Modifier l'apparence d'une courbe

Il est possible de manipuler l'apparence d'une courbe en modifiant la couleur de la courbe, la forme des points de coordonnées et le type de ligne reliant les points.

Pour cela, on ajoute un nouveau argument (qu'on peut appeler *un marqueur*) de type chaîne de caractère à la fonction **plot** comme ceci : **plot(x,y,'marqueur')**

Le contenu du marqueur est une combinaison d'un ensemble de caractères spéciaux rassemblés dans le tableau suivant :

Couleur de la courbe		Représentation des points	
Le caractère	Son effet	Le caractère	Son effet
<b>b</b> ou <b>blue</b>	courbe en bleu	.	un point .
<b>g</b> ou <b>green</b>	courbe en vert	<b>o</b>	un cercle ●
<b>r</b> ou <b>red</b>	courbe en rouge	<b>x</b>	le symbole x
<b>c</b> ou <b>cyan</b>	entre le vert et le bleu	+	le symbole +
<b>m</b> ou <b>magenta</b>	en rouge violacé vif	*	une étoile *
<b>y</b> ou <b>yellow</b>	courbe en jaune	<b>s</b>	un carré ■
<b>k</b> ou <b>black</b>	courbe en noir	<b>d</b>	un losange ◆
Style de la courbe		<b>v</b>	Triangle inférieur ▼
Le caractère	Son effet	<b>^</b>	triangle supérieur ▲
-	en ligne plein ———	<	triangle gauche ◀
:	en pointillé .....:	>	triangle droit ▶
-.:	en point tiret - . - .	<b>p</b>	pentagramme ★
--	en tiret - - - -	<b>h</b>	hexagramme ★

*Exemple :*

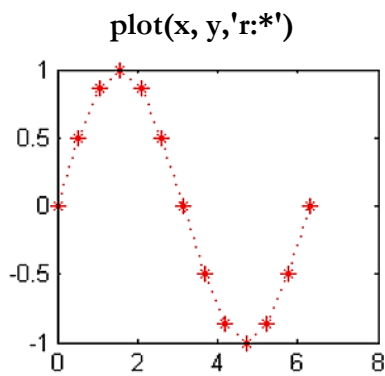
Essayons de dessiner la fonction  $y = \sin(x)$  pour  $x = [0 \dots 2\pi]$  avec un pas =  $\pi/6$ .

```

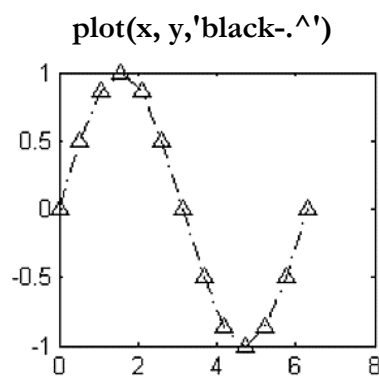
>> x = 0:pi/6:2*pi;
>> y = sin(x);

```

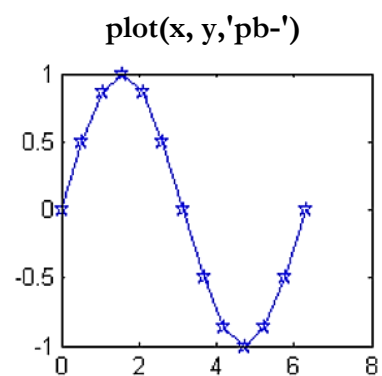
En changeant le marqueur on obtient des résultats différents, et voici quelques exemples :



*Couleur rouge, en pointillé et avec des étoiles*



*Couleur noire, en point tiret et avec des rectangles sup*



*Couleur bleu, en ligne plein et avec des pentagrammes*

### 4.3. Annotation d'une figure

Dans une figure, il est préférable de mettre une description textuelle aidant l'utilisateur à comprendre la signification des axes et de connaître le but ou l'intérêt de la visualisation concernée.

Il est très intéressant également de pouvoir signaler des emplacements ou des points significatifs dans une figure par un commentaire signalant leurs importances.

- Pour donner un titre à une figure contenant une courbe nous utilisons la fonction **title** comme ceci :

```
>> title('titre de la figure')
```

- Pour donner un titre pour l'axe vertical des ordonnées y, nous utilisons la fonction **ylabel** comme ceci :

```
>> ylabel('Ceci est l'axe des ordonnées Y')
```

Pour donner un titre pour l'axe horizontal des abscisses x, nous utilisons la fonction **xlabel** comme ceci :

```
>> xlabel('Ceci est l'axe des abscisses X')
```

- Pour écrire un texte (un message) sur la fenêtre graphique à une position indiquée par les coordonnées **x** et **y**, nous utilisons la fonction **text** comme ceci :

```
>> text(x, y, 'Ce point est important')
```

- Pour mettre un texte sur une position choisie manuellement par la souris, nous utilisons la fonction **gtext**, qui a la syntaxe suivante :

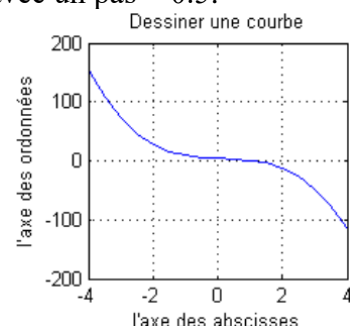
```
>> gtext('Ce point est choisi manuellement')
```

- Pour mettre un quadrillage (une grille), utilisez la commande **grid** (ou **grid on**). Pour l'enlever réutiliser la même commande **grid** (ou **grid off**).

*Exemple :*

Dessignons la fonction :  $y = -2x^3 + x^2 - 2x + 4$  pour  $x$  varie de -4 jusqu'à 4, avec un pas = 0.5.

```
>> x = -4:0.5:4;
>> y = -2*x.^3+x.^2-2*x+4;
>> plot(x,y)
>> grid
>> title('Dessiner une courbe')
>> xlabel('l'axe des abscisses')
>> ylabel('l'axe des ordonnées')
```



#### 4.4. Dessiner plusieurs courbes dans la même figure

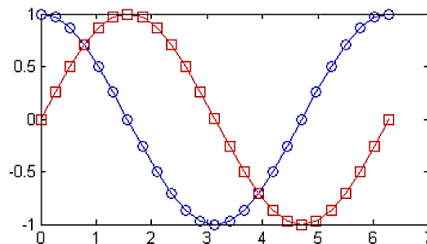
Par défaut en MATLAB, chaque nouveau dessin avec la commande `plot` efface le précédent. Pour forcer une nouvelle courbe à coexister avec les précédentes courbes, Il existe au moins trois méthodes:

##### La commande `hold`

La commande `hold` (ou `hold on`) active le mode « préservation des anciennes courbes » ce qui permette l’affichage de plusieurs courbes dans la même figure. Pour annuler son effet il suffit de réécrire `hold` (ou `hold off`).

Par exemple pour dessiner la courbe des deux fonctions  $\cos(x)$  et  $\sin(x)$  dans la même figure, on peut écrire :

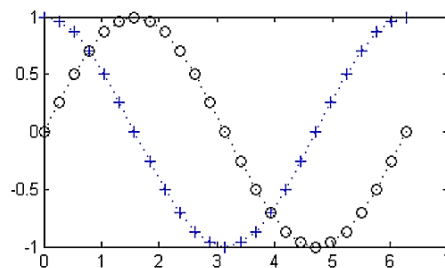
```
>> x=0:pi/12:2*pi;  
>> y1=cos(x);  
>> y2=sin(x);  
>> plot(x,y1,'b-o')  
>> hold on  
>> plot(x,y2,'r-s')
```



#### 4.5. Utiliser `plot` avec plusieurs arguments

On peut utiliser `plot` avec plusieurs couples  $(x,y)$  ou triples  $(x,y, \text{'marqueur'})$  comme arguments. Par exemple pour dessiner les mêmes fonctions précédentes on écrit :

```
>> x=0:pi/12:2*pi;  
>> y1=cos(x);  
>> y2=sin(x);  
>> plot(x,y1,'b:+',x,y2,'k:o')
```



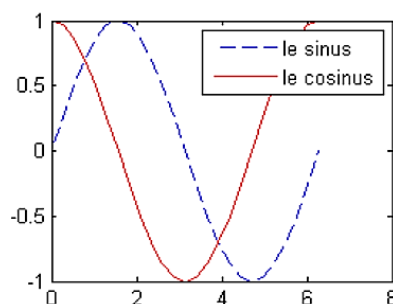
#### 4.6. Utiliser des matrices comme argument de la fonction `plot`

Dans ce cas on obtient plusieurs courbes automatiquement pour chaque colonne (ou parfois les lignes) de la matrice. On a déjà présenté ce cas plus auparavant.

Après pouvoir parvenir à mettre plusieurs courbes dans la même figure, il est possible de les distinguer en mettant une légende indiquant les noms de ces courbes.

Pour cela, on utilise la fonction `legend`, comme illustre l’exemple suivant qui dessine les courbes des deux fonctions  $\sin(x)$  et  $\cos(x)$  :

```
>> x=0:pi/12:2*pi;  
>> y1=sin(x);  
>> y2=cos(x);  
>> plot(x,y1,'b--',x,y2,'-r')  
>> legend('le sinus', 'le cosinus')
```



Il est possible de déplacer la légende (qui se situe par défaut dans le coin supérieur droit) en utilisant la souris avec un glisser-déposer.

#### 4.7. Manipulation des axes d’une figure

MATLAB calcule par défaut les limites (le minimum et le maximum) des axes X et Y et choisie automatiquement le partitionnement adéquat. Mais il est possible de contrôler l’aspect des axes via la commande `axis`.

Pour définir les limites des axes il est possible d'utiliser cette commande avec la syntaxe suivante :

`axis ( [ xmin xmax ymin ymax ] )` Ou `axis ( [ xmin,xmax,ymin,ymax ] )`

Avec : *xmin* et *xmax* définissent le minimum et le maximum pour l'axe des abscisses.  
*ymin* et *ymax* définissent le minimum et le maximum pour l'axe des ordonnées.

Pour revenir au mode d'affichage par défaut, nous écrivons la commande : *axis auto*

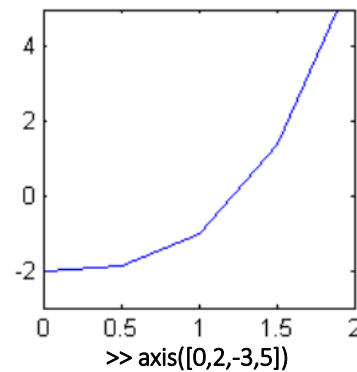
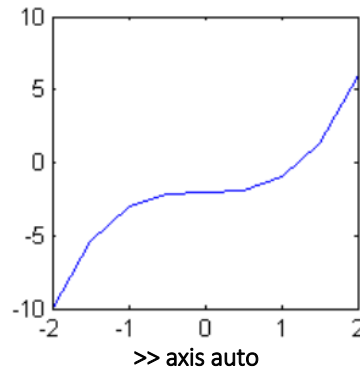
Exemple :

$f(x)=x^3-2$

`>> x = -2:0.5:2;`

`>> y = x.^3-2;`

`>> plot(x,y)`



Parmi les autres options de la commande *axis*, nous présentons les suivantes :

- Pour rendre la taille des deux axes identique (la taille et non le partitionnement, nous utilisons la commande *axis square*. Elle est nommée ainsi car elle rend l'aspect des axes tel un carré.
- Pour rendre le partitionnement des deux axes identiques nous utilisons la commande *axis equal*.
- Pour revenir à l'affichage par défaut et annuler les modifications nous utilisons la commande *axis auto*.
- Pour rendre les axes invisibles nous utilisons la commande *axis off*. Pour les rendre visibles à nouveau nous utilisons la commande *axis on*.

#### 4.8. D'autres types de graphiques

Le langage Matlab ne permet pas uniquement l'affichage des points pour tracer des courbes, mais il offre aussi la possibilité de tracer des graphes à bâtons et des histogrammes.

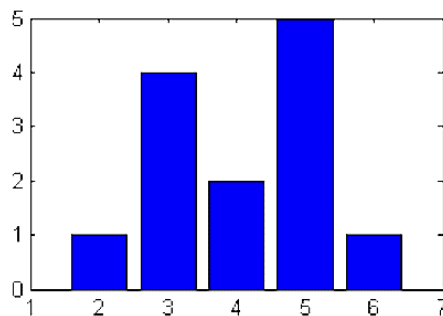
Pour tracer un graphe à bâtons nous utilisons la fonction *bar* qui a le même principe de fonctionnement que la fonction *plot*.

Exemple :

`>> X=[2,3,5,4,6];`

`>> Y=[1,4,5,2,1];`

`>> bar(X,Y)`



Il est possible de modifier l'apparence des bâtons, et il existe la fonction *barh* qui dessine les bâtons horizontalement, et la fonction *bar3* qui ajoute un effet 3D.

Parmi les fonctions de dessins très intéressantes non présentées (faute de place) on peut trouver : *hist*, *stairs*, *stem*, *pie*, *pie3*, ...etc. (que nous vous encourageons à les étudier).

Nous signalons aussi que Matlab permet l'utilisation d'un système de coordonnées autre que le système cartésien comme le système de coordonnées polaire (pour plus de détail chercher les fonctions *compass*, *polar* et *rose*).