

MATLAB

M. LAMLILI

2019-2020

1 Introduction à l'environnement MATLAB

Plan

- 1 Introduction à l'environnement MATLAB
- 2 Les Vecteurs et les Matrices

- 1 Introduction à l'environnement MATLAB
- 2 Les Vecteurs et les Matrices
- 3 Programmation avec MATLAB

- 1 Introduction à l'environnement MATLAB
- 2 Les Vecteurs et les Matrices
- 3 Programmation avec MATLAB
- 4 Les Graphiques avec MATLAB

- 1 Introduction à l'environnement MATLAB
- 2 Les Vecteurs et les Matrices
- 3 Programmation avec MATLAB
- 4 Les Graphiques avec MATLAB

Introduction

MATLAB est une abréviation de MAtrix LABoratory , est écrit à l'origine en Fortron , la version actuelle est écrite en langage C par Math Works Inc.

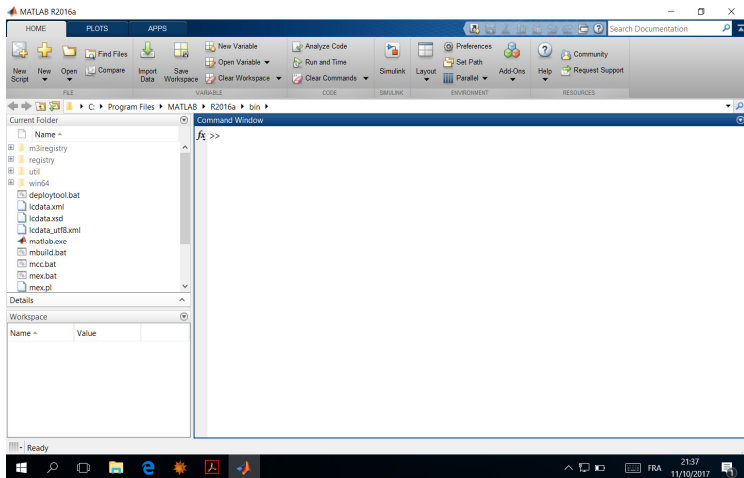
MATLAB est un environnement puissant, complet et facile à utiliser destiné aux calculs scientifiques. Il apporte à l'utilisateur un système interactif intégrant calcul numérique et visualisation.

L'approche matricielle de MATLAB permet de traiter les données sans aucune limitation de taille et de réaliser des calculs numériques de façon fiable et rapide.

MATLAB possède son propre langage qui est naturel, intuitif et dispose de centaines de fonctions mathématique et techniques. L'approche ouverte de MATLAB permet d'inspecter, modifier le code source et les algorithmes de ces fonctions et ajouter d'autres.

L'environnement MATLAB

MATLAB affiche au démarrage plusieurs fenêtres, on peut trouver par exemple dans la version R2016a les fenêtres suivantes :



- ◇ Current Folder : indique le répertoire courant ainsi que les fichiers existants.
- ◇ Workspace : indique toutes les variables existantes avec leurs types et valeurs.
- ◇ Command Window : utilisée pour formaliser nos expressions et interagir avec MATLAB.

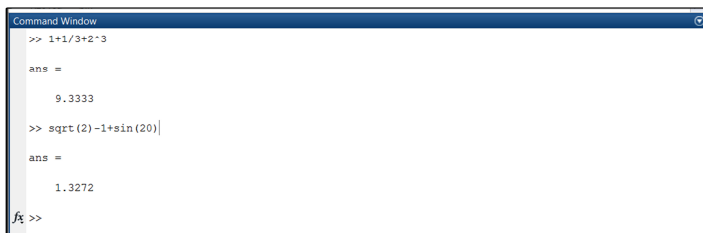
MATLAB est beaucoup plus qu'un langage de programmation, il procède une console d'exécution (shell) qui permet :

- ✓ D'effectuer des opérations mathématiques..
- ✓ Exécuter des fonctions et attribuer des valeurs à des variables.
- ✓ Manipuler des matrices et tracer facilement des graphiques.

L'environnement MATLAB

Le langage MATLAB n'est pas un langage compilé, à chaque appel d'un script (ou d'une fonction) , le logiciel lit et exécute les programmes ligne par ligne.

On peut par exemple utiliser l'invite MATLAB comme une simple calculatrice pour effectuer les opérations arithmétiques.



```
Command Window
>> 1+1/3+2*3

ans =

    9.3333

>> sqrt(2)-1+sin(20)

ans =

    1.3272

fx >>
```

L'environnement MATLAB

On peut affecter des valeurs à des variables et effectuer des opérations sur ces variables. Au MATLAB nous ne sommes pas obligés à déclarer les variables mais le MATLAB choisit le type en tenant en compte l'affectation que nous avons fait.

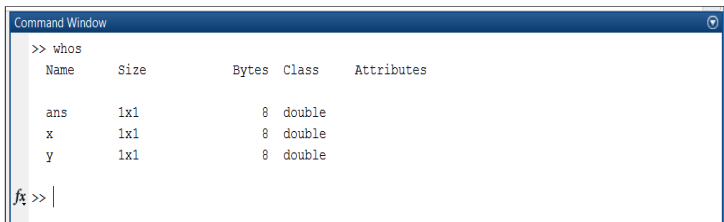


```
Command Window
>> x=3
x =
    3
>> y=5
y =
    5
>> x+y
ans =
    8
>> x*y
ans =
   15
```

Lorsque l'utilisateur ne fixe pas le variable de sortie, Matlab place le résultat d'une opération dans **ans**.

L'environnement MATLAB

Il est toujours possible de connaître les variables qu'on a utilisé ainsi type à l'aide de la fonction **whos**. Pour les manipulations précédentes on a :



```
>> whos
```

Name	Size	Bytes	Class	Attributes
ans	1x1	8	double	
x	1x1	8	double	
y	1x1	8	double	

```
fx >> |
```

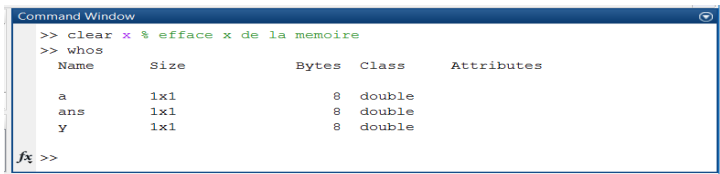
Une fois qu'on exécute une autre commande l'ancienne valeur de **ans** est perdu et remplacé par le résultat de la dernière exécution. Il est préférable de donner toujours des noms aux variables de sortie.

L'environnement MATLAB



```
Command Window
>> x=7
x =
    7
>> y=3
y =
    3
>> a=x^y
a =
   343
fx >> |
```

La fonction clear permet d'effacer des variable par exemple:

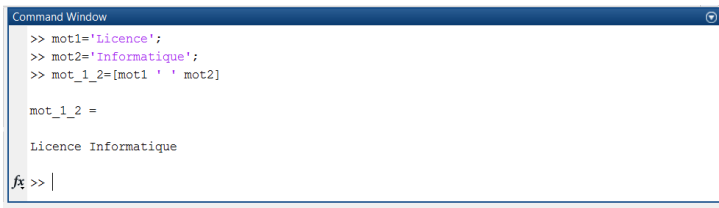


```
Command Window
>> clear x % efface x de la memoire
>> whos
  Name      Size      Bytes  Class  Attributes
  ----      -
  a         1x1         8  double
  ans       1x1         8  double
  y         1x1         8  double
fx >>
```

Le signe % permet de mettre ce qui suit sur la ligne en commentaire (Matlab n'en tiendra pas compte à l'exécution)

L'environnement MATLAB

Pour les variables de type caractère : **char**, la déclaration se fait entre apostrophes. Il est possible de concaténer (lier) des mots à l'aide des crochets.



```
Command Window

>> mot1='Licence';
>> mot2='Informatique';
>> mot_1_2=[mot1 ' ' mot2]

mot_1_2 =

Licence Informatique

fx >> |
```

Le point-virgule ; est utilisé pour stopper l'affichage.

L'emploi de ' ' permet d'introduire un espace entre les mots. On peut aussi créer une autre variable **mot3** de type **char** qui contient un espace et concaténer les trois mots pour avoir le même résultat.

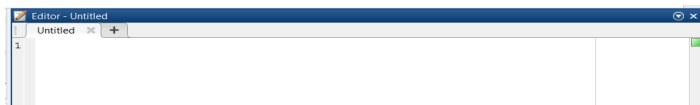
```
>> mot3 = ' ';
>> mot_1_2=[mot1 mot3 mot2]
```

Les fichiers SCRIPT et FUNCTION

Pour les taches répétitives, il est pratique d'écrire de courts programmes et de les sauvegarder dans des fichiers qu'on appelle chaque fois qu'on veut.

Au MATLAB on trouve deux types de fichiers : les fichiers **SCRIPT** et les fichiers **FUNCTION**. Dans les deux cas, il faut lancer **l'éditeur** de MATLAB et créer vos fichiers et les sauvegarder avec l'extension **.m** . Cette approche est défini en Matlab par les M-Files.

Pour créer un M-Files il suffit de taper la commande `edit` au console au aller dans le menu et cliquer sur New M-files. Une fenêtre d'édition comme celle-ci va apparaître :



Les fichiers SCRIPT et FUNCTION

On peut par exemple reprendre les taches précédentes et les exécuter dans un fichier qu'on nome test.m

```
% test.m  
clear all  
x = 3;  
y = 2;  
a = x * y  
b = xy  
whos
```

Pour exécuter le fichier test.m on le sauvegarde puis soit qu'on tape test.m au console et on valide soit directement on clique sur **Run** dans le menu de l'éditeur.

Les fichiers SCRIPT et FUNCTION

Matlab contient un grand nombre de fonctions prédéfinies comme **sin**, **cos**, **sqrt**, **sum** etc... Il est possible de créer nos propres fonctions en écrivant leurs codes source dans des fichiers M-files (portant le même nom de fonction).

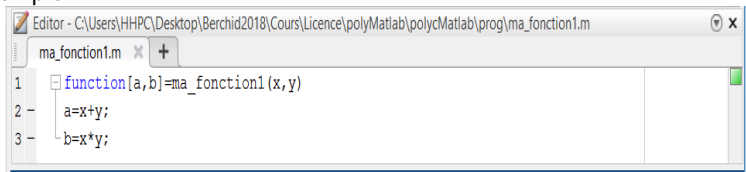
```
function[r1,r2,...,rn]= nom_fonction(arg1,arg2,...,argp)  
% le corps de la fonction  
....  
r1 = ... % la valeur retournée pour r1  
r2 = ... % la valeur retournée pour r2  
....  
rn = ... % la valeur retournée pour rn
```

Où r_1, r_2, \dots, r_n sont les valeurs retournées et $arg_1, arg_2, \dots, arg_p$ sont les arguments.

Le rôle d'une fonction est d'effectuer des opérations sur une ou plusieurs entrées pour obtenir un résultat qui sera appelé sortie.

Les fichiers SCRIPT et FUNCTION

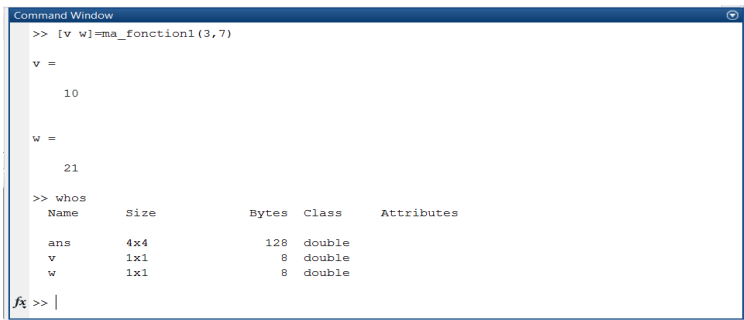
Par exemple:



The screenshot shows a MATLAB editor window with the title bar "Editor - C:\Users\HHP\ Desktop\Berchid2018\Cours\Licence\polyMatlab\polycMatlab\prog\ma_fonction1.m". The editor contains the following code:

```
1 function[a,b]=ma_fonction1(x,y)
2     a=x+y;
3     b=x*y;
```

Si on execute cette fonction au console on aura:



The screenshot shows a MATLAB Command Window with the following text:

```
>> [v w]=ma_fonction1(3,7)

v =

    10

w =

    21

>> whos
```

Name	Size	Bytes	Class	Attributes
ans	4x4	128	double	
v	1x1	8	double	
w	1x1	8	double	

fx >> |

Les principales constantes, fonctions et commandes

Matlab définit les constantes suivantes:

La constante	Signification	.
pi	3.1416	
exp(1)	2.7183	
i	$\sqrt{-1}$	
j	$\sqrt{-1}$	
Inf	∞	
NaN	Not a Number (pas de nombre)	
eps	2.2204e-16	

Parmi les fonctions fréquemment utilisées, on peut noter les suivantes

La fonction	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 cos de x (en radian)	

Les principales constantes, fonctions et commandes

La fonction	Signification
$\text{atan}(x)$	l'arc tangent de x (en radian)
$\text{sqrt}(x)$	la racine carrée de x (\sqrt{x})
$\text{abs}(x)$	la valeur absolue de x ($ x $)
$\text{exp}(x)$	l'exponentielle de x (e^x)
$\text{log}(x)$	le logarithme naturel de x ($\ln(x)$)
$\text{log10}(x)$	le logarithme à base 10 de x ($\ln_{10}(x)$)
$\text{imag}(x)$	la partie imaginaire du nombre complexe x
$\text{real}(x)$	la partie réelle du nombre complexe x
$\text{round}(x)$	arrondi un nombre vers l'entier le plus proche
$\text{floor}(x)$	arrondi vers l'entier le plus petit ($\max\{n n \leq x, n \in \mathbb{N}\}$)
$\text{ceil}(x)$	arrondi vers l'entier le plus grand ($\min\{n x \leq n, n \in \mathbb{N}\}$)

Matlab offre beaucoup de commandes pour l'interaction avec l'utilisateur. Nous nous contentons pour l'instant d'un petit ensemble, et nous donnons les autres au fur et à mesure de l'avancement du cours

La commande	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 ou clear all	Supprime toutes les variables
clc	Efface l'écran
exit, quit	Fermer l'environnement Matlab.

Les nombres en Matlab

Matlab utilise une notation décimale conventionnelle, avec un point décimal facultatif `'.'` et le signe `'+'` et `'-'` pour les nombres signés. La notation scientifique utilise la lettre `'e'` pour spécifier la puissance de 10. Les nombres complexes utilisent les caractères `'i'` et `'j'` (indifféremment) pour désigner la partie imaginaire.

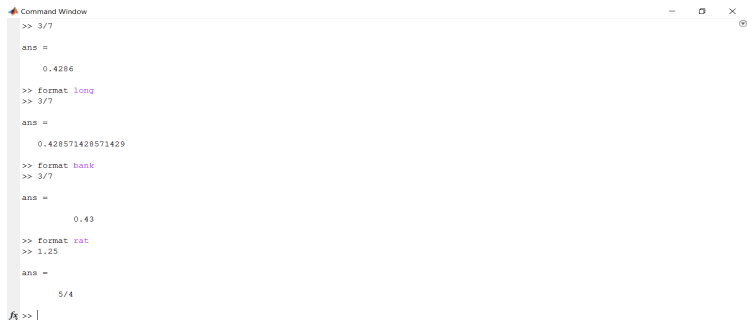
Le type	Exemples	.
Entier	5 -83	
Réel en notation décimale	0.0205 3.1415926	
Réel en notation scientifique	1.60210e -20	
Complexe	5+3i -314159j	

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

Les nombres en Matlab

Le résultat d'une opération de calcul est par défaut affiché avec 4 chiffres après la virgule. Pour modifier l'affichage on utilise les commandes:

La commande	Signification
<code>format short</code>	Affiche les nombres avec 04 chiffres après la virgule.
<code>format long</code>	Affiche les nombres avec 14 chiffres après la virgule.
<code>format bank</code>	Affiche les nombres sous la forme ratio $\frac{a}{b}$.



```
Command Window
>> 3/7
ans =
    0.4286

>> format long
>> 3/7
ans =
    0.428571428571429

>> format bank
>> 3/7
ans =
    0.43

>> format rat
>> 1.25
ans =
    5/4
```

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

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


- 1 Introduction à l'environnement MATLAB
- 2 Les Vecteurs et les Matrices
- 3 Programmation avec MATLAB
- 4 Les Graphiques avec MATLAB

Les Vecteurs en Matlab

L'élément de base de Matlab est la matrice. Un scalaire est une matrice de dimension 1×1 , un vecteur colonne de dimension n est une matrice $n \times 1$, un vecteur ligne de dimension n est une matrice $1 \times n$.

Contrairement au langage de programmation usuelle, il n'est obligatoire de déclarer les variables avant de les utiliser mais il faut prendre toutes les précautions dans la manipulation de ces objets.

Les scalaires se déclarent directement, par exemple:

```
>> x=1;
>> a=x
ans =
1
>> a+a
ans =
2
```

Les Vecteurs en Matlab

Les vecteurs lignes se déclarent entre crochets en séparant les éléments par des espaces ou des virgules.

```
>> V_ligne=[1 3 4]
V_ligne =
1 3 4
>> V1=[1 ,3, 4.3]
V1 =
1.0000 3.0000 4.3000
```

Les vecteurs colonnes se déclarent entre crochets en séparant les éléments par des des points virgules.

```
>> V_colonne=[10 ;3; 41]
V_colonne =
10
3
41
```

Il est possible de transposer un à l'aide de la fonction **transpose** ou directement par une apostrophe .

```
>> A=[2 3 -4]
A =
1 3 4
>> B=transpose(A)
B=
2
3
-4
>> C = B'
C =
1 3 4
```

Les Vecteurs en Matlab

Le double point (:) est l'opérateur d'incrémentation dans Matlab. Pour créer un vecteur ligne des valeurs de 0 à 1 par incrément de 0.2 il suffit d'utiliser.

```
>>T=[0:0.2:1]
T=
0 0.2000 0.4000 0.6000 0.8000 1.0000
```

Par défaut, l'incrément c'est 1 . Pour créer un vecteur ligne des valeurs par exemple de 0 à 5 par incrément de 1. On utilise.

```
>>TT=[0:5]
TT=
0 1 2 3 4 5
```

Les Vecteurs en Matlab

Le double point (:) est l'opérateur d'incrémentation dans Matlab. Pour créer un vecteur ligne des valeurs de 0 à 1 par incrément de 0.2 il suffit d'utiliser.

```
>>T=[0:0.2:1]
T=
0 0.2000 0.4000 0.6000 0.8000 1.0000
```

Par défaut, l'incrément c'est 1. Pour créer un vecteur ligne des valeurs par exemple de 0 à 5 par incrément de 1. On utilise.

```
>>TT1=[0:5]
TT1=
0 1 2 3 4 5
>>TT=0:2:8
TT2=
0 2 4 6 8
```

On peut aussi en utilisant (:) créer un vecteur ligne des valeurs sans utilisé les crochets.

Les Vecteurs en Matlab

Soit V un vecteur de type $1 \times n$ pour accéder à la i ème élément de ce vecteur on écrit $V(i)$. Par exemple pour afficher la 4 ème composante de on écrit $V(4)$. On peut aussi changer la valeur d'une composante on affectant à cette composante la valeur qu'on veut comme suit: $V(i) = a$.

```
>>V=[1 0 2.5 1 7]
V=
1.0000    0    2.5000    1.0000    7.0000
>>V(3) ans=
2.5000
>>V(4)=40
V=
1.0000    0    2.5000   40.0000    7.0000
```

On peut aussi accéder à la dernière composante du vecteur V en mettant $V(end)$.

La fonction linspace

La création d'un vecteur dont les composantes sont ordonnées par intervalle régulier et avec un nombre d'éléments connu peut se réaliser avec la fonction :

$$\text{linspace}(\text{début}, \text{fin}, \text{nombre d'éléments})$$

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

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

```
>>V1=linspace(0,4,5)
V1=
0    1    2    3    4
>>V2=linspace(0,4,15)
V2=
Columns 1 through 10
0 0.2857 0.5714 0.8571 1.1429 1.4286 1.7143 2.0000 2.2857 2.5714
Columns 11 through 15
2.8571 3.1429 3.4286 3.7143 4.0000
```


Les Vecteurs en Matlab

La taille d'un vecteur (le nombres de ces composantes) peut être obtenue avec la fonction **length** comme suit:

```
>>X=[7 3 2 1 0 1];  
>>n=length(X)  
n=  
6
```

Les opérations usuelles d'addition, la soustraction et de multiplication par un scalaire sur les vecteurs sont définies dans Matlab comme suit:

```
>>V1=[7 3 2] ; V2=[8 7 5]  
>> V1+V2  
15  10  7  
>> V1-V2  
-1  -4  -3  
>> V1+V2  
15  10  7  
>> V3=2*V1  
ans=  
14  6  4
```

Les Matrices en Matlab

On peut créer les Matrices en séparant les lignes par des points virgules :.

```
>>V=[1 0 2; 0 0 1; 1 3 5]
```

```
V=
```

```
1 0 2
```

```
0 0 1
```

```
1 3 5
```

On peut aussi créer des matrice par concaténation des vecteurs lignes ou des vecteurs colonnes.

```
>>V1=[1 0 2];V2=[0 0 1];V3=[ 1 3 5];
```

```
>> V=[V1;V2;V3]
```

```
V=
```

```
1 0 2
```

```
0 0 1
```

```
1 3 5
```

Les Matrices en Matlab

La commande **inv** donne l'inverse d'une matrice :

```
>>V=[1 2; 3 4];  
>>V1=inv(V);  
V1=  
-2.0000  1.0000  
 1.5000 -0.5000
```

La commande **transpose** donne la transposé d'une matrice on peut aussi transposer une matrice par L'apostrophe:

```
>>V=[1 2; 3 4];  
>>V2=transpose(V);  
V1=  
1 3  
2 4
```

Les Matrices en Matlab

Un des intérêts de Matlab est la possibilité d'utiliser directement les opérations arithmétiques (addition, soustraction, multiplication par un scalaire et multiplication) prédéfinies pour les matrice:

```
>>A=[1 2; 7 3];  
>>B=[5 5; 10 12];  
>>C1=A+B;  
C1=  
6 7  
17 15  
>>C2=B-A;  
C2=  
4 3  
3 9
```

```
>>A=[1 2; 7 3];  
>>B=[5 5; 10 12];  
>>C3=3*A;  
C3=  
3 6  
15 15  
>>C4=A*B;  
C4=  
25 29  
65 71  
>>D=B/A;  
D=  
1.8182 0.4545  
4.9091 0.7273
```

Attention: En Mathématique la division des matrice n'est pas autorisée.

Les Matrices en Matlab

Soit $A = (a_{ij})$ une matrice de type $m \times n$

- L'accès à l'élément a_{ij} 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 $A(:,j)$.

```
>>A=[1 2 6 2 ; 7 3 9 8];
```

```
>>A(2,3)
```

```
ans=
```

```
9
```

```
>>A(2,:);
```

```
ans=
```

```
7 3 9 8
```

```
>>A(:,3)
```

```
ans=
```

```
6
```

```
9
```

Les Matrices en Matlab

```
>>A=[1 2 3 4 ; 5 6 7 8;9 10 11 12];  
>>A(2:3,:) % tous les elements de la 2 et la 3 ligne  
5 6 7 8  
9 10 11 12  
>>A(2,:) % la sous matrice supérieure droite de taille  $2 \times 2$   
3 4  
7 8  
>>A([1 3],[2 4]) % la sous matrice : lignes (1,3) et colonnes (2,4)  
2 4  
10 12  
>>A(:,3)=[ ] % supprimer la troisième colonne  
1 2 4  
5 6 8  
9 10 12  
>>A=[A,[0;17;4]] % ajouter une colonne (ou A(:,4)=[0;17;4])  
1 2 4 0  
5 6 8 17  
9 10 12 4
```

Les Matrices en Matlab

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

```
A=[1 2 3 4 ; 5 6 7 8;9 10 11 12];  
>>d=size(A)  
d=  
3 4
```

On peut obtenir les dimensions séparément en utilisant la syntaxe:

```
A=[1 2 3 4 ; 5 6 7 8;9 10 11 12];  
>>d1=size(A,1) % d1 contient le nombre des lignes (m)  
d1=  
3  
>>d2=size(A,2) % d2 contient le nombre des colonnes (n)  
d1=  
4
```


Génération automatique des matrices

Dans, Matlab, il existe des fonctions qui permettent de générer automatiquement des matrices particulières. Dans le tableau suivant nous présentant les plus utilisées:

La fonction	Signification
<code>zeros(n)</code>	Générer une matrice $n \times n$ avec tous les éléments =0
<code>zeros(m,n)</code>	Générer une matrice $m \times n$ avec tous les éléments =0
<code>ones(n)</code>	Générer une matrice $n \times n$ avec tous les éléments =1
<code>eye(n)</code>	Générer une matrice identité de dimension $n \times n$
<code>magic(n)</code>	Générer une matrice magique de dimension $n \times n$
<code>rand(m,n)</code>	Générer une matrice de dimension $m \times n$ de valeurs aléatoires

- 1 Introduction à l'environnement MATLAB
- 2 Les Vecteurs et les Matrices
- 3 Programmation avec MATLAB**
- 4 Les Graphiques avec MATLAB

Dans cette partie, Nous allons présenter les mécanismes d'écriture et d'exécution des programmes en Matlab.

Entrée au clavier:

L'utilisateur peut saisir des informations au clavier grâce à la commande **input**.

```
>> x=input('donner une valeur de x :');  
donner une valeur de x :5  
x=  
5  
>> V=input('donner une valeur de V :')  
donner une valeur de V :[2 3 12]'  
V=  
2  
3  
12
```

Sortie à l'écran :

Pour afficher quelque chose à l'écran, on peut utiliser la commande **disp** qui affiche le contenu d'une variable (scalaire, vecteur, chaîne de caractères ou matrice) .

```
>>A=[1 2 3;4,5,6];  
>> disp(A)  
1 2 3  
4 5 6  
>> C='Nous sommes'  
>> disp(C)  
Nous sommes  
>> disp(A(:,2))  
2  
5
```

Opérateurs de comparaisons et logiques

Les opérateurs de comparaison sont :

`==` : égal à ($x == y$)

`>` : strictement plus grand que ($x > y$)

`<` : strictement plus petit que ($x < y$)

`>=` : plus grand ou égal à ($x >= y$)

`<=` : plus petit ou égal à ($x <= y$)

`~=` : différent de ($x ~= y$)

Le résultat d'un test est un booléen, qui sous Matlab, prend la valeur 1 pour vrai et 0 pour faux. Par exemple, on a les résultats suivants :

```
>>r=8<7
```

```
r= 0
```

```
>> r=[2 7 5]>=[1 7 12]
```

```
r= 1  1  0
```

```
>> r=6>=[1 7 12]
```

```
r= 1  0  0
```

Les opérateurs de comparaison sont utilisés essentiellement dans les instructions de contrôle.

Opérateurs de comparaisons et logiques

Les opérateurs de logique sont :

& : et logique (and)

| : ou logique (or)

~ : non logique (not)

Le résultat d'une opération logique est un booléen, qui sous Matlab, prend la valeur 1 pour vrai et 0 pour faux. Par exemple, on a les résultats suivants :

```
>> r = (3 > 2) & (5 > 2.5);  
r=1  
>> r = (2 > 7) | (5 > 2.5)  
r=1  
>> r = ([3 -2] > 2) & ~(3 > [4 2])  
r=1 0
```

Les opérateurs de logique et sont aussi utilisés essentiellement dans les instructions de contrôle.

Instructions de contrôle

Les instructions de contrôle sous Matlab sont très proches de celles existant dans d'autres langages de programmation.

L'instruction **if**

C'est la plus utilisée dans le contrôle de flux. Elle oriente l'exécution de programme en fonction de la valeur logique d'une condition. Sa syntaxe générale est la suivante :

```
if (expression booléenne 1 )  
    Ensembles d'instructions 1  
elseif (expression booléenne 2 )  
    Ensembles d'instructions 2  
...  
elseif (expression booléenne n )  
    Ensembles d'instructions n  
else  
    Ensembles d'instructions si toutes les expressions booléennes  
    étaient fausses  
end
```

Exemple:

script `pile_face.m` qui simule un tirage à pile ou face en utilisant la fonction **rand()** qui un nombre aléatoire compris entre 0 et 1 selon une loi uniforme :

```
if x > 0.5
    disp('pile')
else
    disp('face')
end
```

Exercice:

Ecrire un script `solveEq2.m` réalisant la résolution d'une équation du second degré $ax^2 + bx + c = 0$ où a , b et c sont des nombres réels connus.

Il existe au Matlab la fonction `solve` qui permet la résolution de ce problème, chercher la dans le help de Matlab.

Boucle for:

La boucle for répète une suite d'instruction un nombre prédéterminé de fois. Sa structure est la suivante :

```
for var = <liste de valeurs>  
    < suite d'instruction >  
end
```

La liste de valeurs correspond à la définition d'un vecteur (un vecteur de valeur, un vecteur définie par début : fin ou début : pas : fin).

La variable va parcourir tous les éléments du vecteur définie par la liste de valeurs, et pour chacune il va exécuter la suite d'instructions.

Instructions de contrôle

Le tableau suivant donne trois formes de l'instruction for avec leurs résultats d'exécution:

L'instruction for	for i=1:4	for i=1:2:4	[1,4,7]
	j=2*i;	j=2*i;	j=2*i;
	disp(j)	disp(j)	disp(j)
Le résultat	2	2	2
de l'exécution	4	6	8
	6		14
	8		

Exercice:

- 1-Ecrire une fonction Matlab qui prend deux matrices A et B comme arguments et qui retourne leurs sommes.
- 2-Ecrire une fonction Matlab qui prend un scalaire a et une matrices A comme arguments et qui retourne leurs produit.
- 3-Ecrire une fonction Matlab qui prend deux matrices A et B comme arguments et qui retourne leurs produit.

Boucle while:

L'instruction while répète l'exécution d'une suite d'instruction un nombre indéterminé de fois selon la valeur de la condition logique. Elle a la forme générale suivante:

```
while <condition logique>  
    < suite d'instruction >  
end
```

Tant que la condition logique est évaluée à vrai, la suite d'instructions s'exécutera en boucle.

```
epsil=10-3;  
s=0;i=0;  
while(epsil<abs(exp(1)-s))  
s=s+1/factorial(i); i=i+1;  
end  
disp(s);disp(i)
```

Ce programme permet de calculer $e \simeq 2.7183$, après execution on trouve $i = 7$ et $s = 2.7181$.

L'instruction break:

Provoque l'arrêt de la boucle for et while. Par exemple L'instruction for ci-dessous est stoppée au premier i tel que $t(i)$ est nul :

```
t = -2:10;  
for i = 1:length(t)  
    if t(i) == 0  
        break;  
    end  
end  
disp(i)  
3
```

L'instruction continue:

L'instruction continue : dans une boucle for ou while provoque l'arrêt de l'itération courante, et passage le programme au début de l'itération suivante. Supposons que l'on parcourt un tableau t pour réaliser un certain traitement sur tous les éléments, sauf ceux qui sont négatifs:

```
t = [-2,5, 3 -4 7];  
for i = 1:length(t)  
    if t(i) < 0  
        continue ;  
    end  
    disp(t(i)) end  
5  
3  
7
```

L'instruction switch:

L'instruction switch exécute des groupes d'instructions selon la valeur d'une expression. Chaque groupe est associé à un cas qui définit si ce groupe doit être exécuté ou pas selon l'égalité de la valeur de ce cas avec le résultat d'évaluation de l'expression de switch. Si tous les cas ne sont pas réalisés, il est possible d'ajouter une clause otherwise qui sera exécutée. La structure de **switch** est la suivante :

```
switch (expression)
  case valeur_1
    groupes d'instructions_1
  ...
  case valeur_n
    groupes d'instructions_n
  otherwise
    groupes d'instructions
end
```

Exemple:

Ce script affiche en lettres un chiffre inférieur à deux:

```
x=3;
switch x
    case 0
        disp('zéro')
    case 1
        disp('un')
    case 2
        disp('deux')
    otherwise
        disp('ce nombre n existe pas dans cette base')
end
```

Opérations arithmétiques

L'additions +, la soustractions −, la multiplications * , la puissances ^ s'utilisent avec la syntaxe matricielle habituelle.

Attention aux dimensions: on ne peut ajouter ou soustraire que des matrices de même taille. Il existe cependant une manière simple d'ajouter ou soustraire le même scalaire à tous les éléments d'une matrice. Par exemple, $x = [1 \ 2 \ 3]$, $x = x - 1$ produit le résultat $x = [0 \ 1 \ 2]$.

La multiplication de deux matrices n'a de sens que si leurs dimensions 'internes' sont égales. -

$$(A * B)_{ij} = \sum_{k=1}^n A_{ik} B_{kj}$$

où n est le nombre de colonnes de A et le nombre de lignes de B .

Exemple: $A = [1 \ 2 \ 3; 4 \ 5 \ 6]$; $B = [7; 8; 9]$; le résultat de $A * B$ est

ans =

50

112

L'opérateur 'point' '.' :

En calcul matriciel, on a souvent besoin d'effectuer une opération élément par élément. Dans Matlab, ces opérations sont appelées opérations sur des tableaux (array operations).

Bien entendu, l'addition et la soustraction sont des opérations qui se font élément par élément. L'opération $A .* B$ désigne la multiplication, élément par élément, des matrices A et B .

Exemple:

$A = [1 \ 2 \ 3; 4 \ 5 \ 6]; B = [2 \ 5 \ 3; 7 \ 5 \ 2];$ le résultat de $A .* B$ est

ans =

```
2    10    9
28   25   12
```

Exercice: écrire une fonction monply3 qui prend comme un vecteur x et return un vecteur $y = 3x^2 + 5x + 2$.

- 1 Introduction à l'environnement MATLAB
- 2 Les Vecteurs et les Matrices
- 3 Programmation avec MATLAB
- 4 Les Graphiques avec MATLAB**

Graphiques 2D

Plot(vecteur en abscisses, vecteurs en ordonnées)

L'instruction « plot » permet de tracer un graphique simple représentant deux vecteurs.

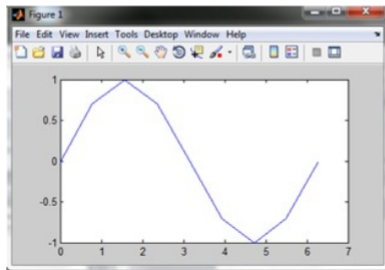
Tappez les commandes suivantes:

```
>> x=0:pi/4:2*pi;  
>> y=sin(x);  
>> plot(x,y)
```

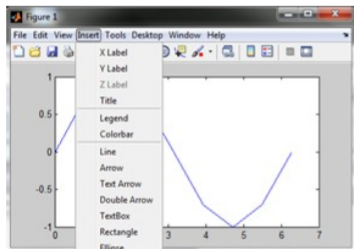
La première commande crée un vecteur x de 0 à 2π avec un incrément de $\pi/4$;

La seconde commande crée un vecteur y qui est le sinus de x;

La troisième commande trace y en fonction de x.



On peut ajouter des informations sur notre graphique pour le rendre plus clair. Par exemple, on peut ajouter des titres pour les deux axes. Il existe deux manières de le faire :



Le plus simple, c'est d'utiliser le menu de la fenêtre graphique. Il suffit de choisir l'anglet « Insert » et sélectionner ensuite « X Label » pour l'axe des abscisses, « Y Label » pour l'axe des ordonnées ou « Title » pour le titre de la figure.

La seconde manière de réaliser cette opération est d'ajouter une commande du type :

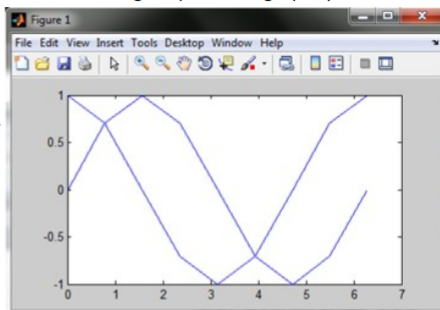
```
>> xlabel('abscisse')
```

Cette commande aura pour effet d'ajouter « abscisse » comme titre des abscisse.

Graphiques 2D

Il est également possible de tracer sur la même figure plusieurs graphiques.

```
>> plot(x,y)
>> x=0:pi/4:2*pi;
>> y=sin(x);
>> z=cos(x);
>> plot(x,y)
>> hold on
>> plot(x,z)
```



La commande « hold on » permet de maintenir la fenêtre « Figure 1 » ouverte pour y ajouter un graphique.

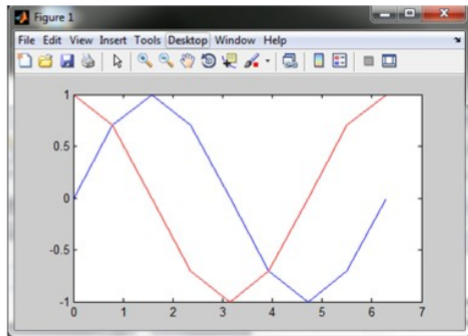
Si on veut créer un nouveau graphique sur la même figure en ayant effacé l'ancien, il suffit d'écrire « hold off »

Si on veut garder l'ancienne figure et tracer un nouveau graphique sur une autre fenêtre, il suffit d'écrire : figure(2) qui ouvre seconde fenêtre dans ce cas.

Graphiques 2D

Pour différencier les deux courbes tracées dans la figure précédente, il suffit de les caractériser par des couleurs différentes ou des symboles différents. Ajouter dans les argument de la fonction plot 'b' pour bleu ou 'r' pour rouge par exemple.

```
>> hold off  
>> plot(x,y,'b')  
>> hold on  
>> plot(x,z,'r')
```



Pour en savoir un peu plus sur la fonction plot, il suffit d'aller dans le menu « Help », « Product Help » et de taper plot.

Graphiques 2D

On peut ajouter une légende, il suffit pour cela d'écrire :

```
>> h = legend('sin(x)', 'cos(x)', 2);
```

Ici, $\sin(x)$ pour la première courbe et $\cos(x)$ pour la seconde.

Ici, la légende est placée en haut à gauche. Retapez la commande en remplaçant 2 par 1, 3 ou 4 pour voir les différents emplacement de la légende.

