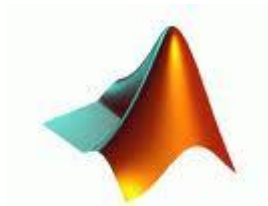
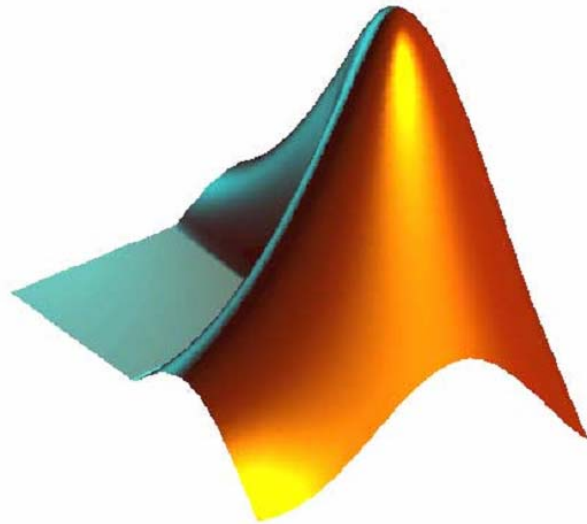


Support du cours LCS
Langage du calcul scientifique (Matlab)
FB/MEHDA



Université Abderrahmane MIRA de Bejaia
Département ST



Programme Matlab

1. Présentation et généralités

- 1.1 Une session Matlab
- 1.2 L'espace de travail
- 1.3 Obtenir de l'aide
- 1.4 Syntaxe d'une ligne d'instruction
- 1.5 Messages d'erreurs
- 1.6 Les fichiers « .m »

2. Types de données et variables

- 2.1 Le type complexe
- 2.2 Le type chaîne de caractères
- 2.3 Le type logique
- 2.4 Le type vecteur
- 2.5 Le type matrice
- 2.6 Lecture des données
- 2.7 Affichage des données
- 2.8 Sauvegarde des données

3. Calculer avec Matlab

- 3.1 Opération portant sur les scalaires
- 3.2 Opération portant sur les vecteurs
- 3.3 Opération portant sur les matrices

4. Programmer sous Matlab

- 4.1 Opérateurs de comparaison et opérateurs logiques
- 4.2 Instructions de contrôle
 - 4.2.1 Boucle for (parcours d'un intervalle)
 - 4.2.2 Boucle While (tant que)
 - 4.2.3 L'instruction if (si)
 - 4.2.4 L'instruction switch
- 4.3 Instructions d'interruption d'une boucle
- 4.4 La programmation vectorielle
 - 4.4.1 Manipulation des vecteurs
 - 4.4.2 Manipulation des matrices

5. Graphisme

- Gérer les fenêtres graphiques
- Tracer le graphe d'une fonction (fplot, plot, subplot...)

6. Introduction à Simulink

Chapitre 1 : Présentation et généralités

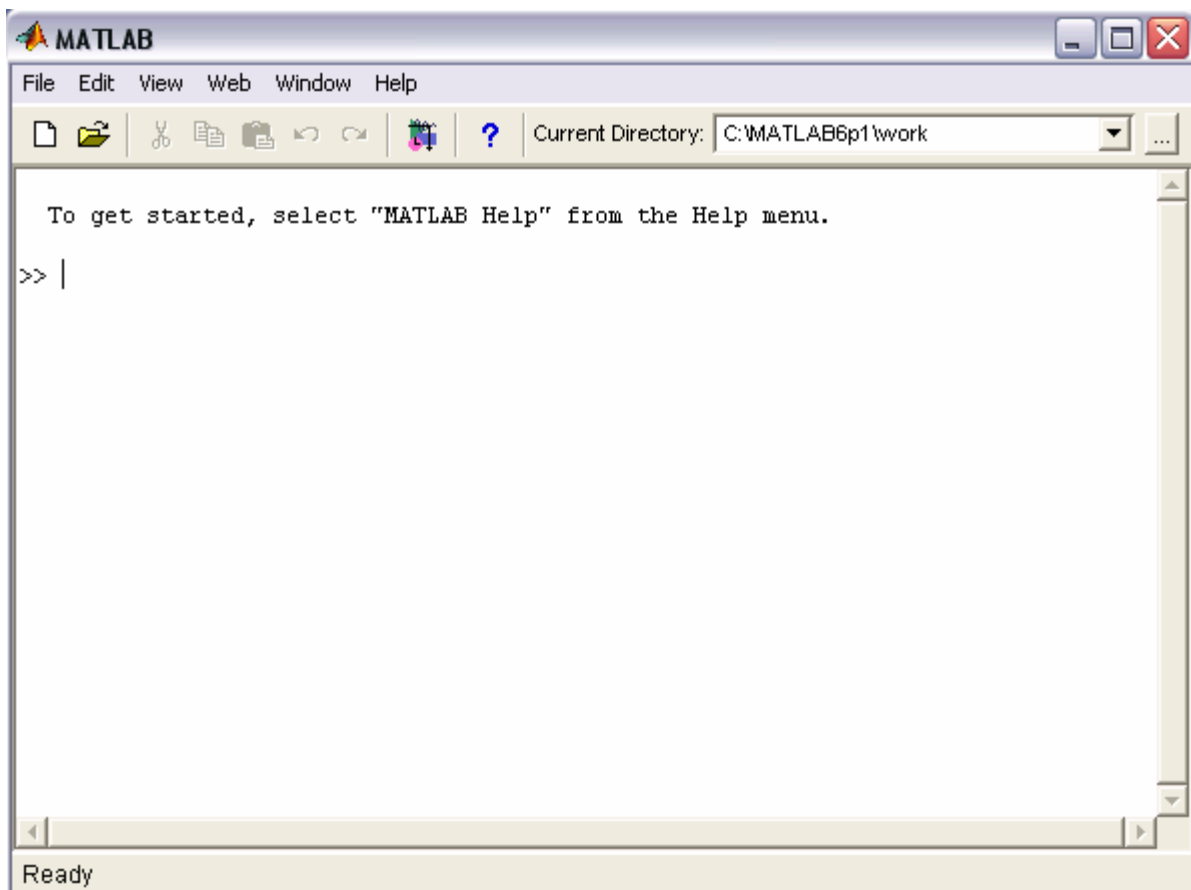
Introduction FB/MEHDA

Matlab est un logiciel de calcul numérique produit par MathWorks (voir le site web <http://www.mathworks.com/>). Il est disponible sur plusieurs plateformes. Matlab est un langage simple et très efficace, optimisé pour le traitement des matrices, d'où son nom. Pour le calcul numérique, Matlab est beaucoup plus concis que les "vieux" langages (C, Pascal, Fortran, Basic). Un exemple: plus besoin de programmer des boucles modifier pour un à un les éléments d'une matrice. On peut traiter la matrice comme une simple variable. Matlab contient également une interface graphique puissante, ainsi qu'une grande variété d'algorithmes scientifiques.

On peut enrichir Matlab en ajoutant des "boîtes à outils" (toolbox) qui sont des ensembles de fonctions supplémentaires, profilées pour des applications particulières (traitement de signaux, analyses statistiques, optimisation, etc.).

1.1 Une session Matlab

Pour lancer Matlab commencer par ouvrir une fenêtre de commande Matlab



Le prompt Matlab (>>) indique que Matlab attend des instructions. Voici un exemple de session Matlab

```
>> A=2 (après retour chariot voici ce q'on va obtenir)
A=
2
```

FB/MEHDA

Remarque : quand une instruction comporte une variable = à une expression alors l'affichage du résultat est la même variable = au résultat. Si l'instruction est seulement le calcul d'une instruction alors l'affichage du résultat est **ans** = au résultat.

```
exemple : >> B=2*A+1 ↵          >> 3*A ↵
           B =                    ans =
           5                       6
```

Chaque ligne d'instruction doit se terminer par un retour chariot (validation). La commande pour quitter Matlab est **quit**.

1.2 L'espace de travail

Matlab permet de définir des données variables. Les variables sont définies au fur et à mesure que l'on donne leurs noms (identification) et leurs valeurs numériques ou leurs expressions mathématiques. Matlab ne nécessite pas de déclaration de type ou de dimension pour une variable (tableau...).

Voici quelques commandes pour faciliter la programmation :

who : fournit la liste des variables définies dans l'espace de travail (workspace).

whos : donne plus d'informations sur les variables.

clear : efface les variables du workspace. Il est possible de ne détruire qu'une partie des variables en tapant **clear** liste de noms de variables.

clc : efface l'écran.

Exemple :

```
>> x=2;y=x*x;z=y/4;
>> A=[1 5; 5 8];B=A*A;
>> t='bonjour';
```

```
>> who
```

Your variables are:

```
A B t x y z
```

```
>> whos
```

Name	Size	Bytes	Class
A	2x2	32	double array
B	2x2	32	double array
t	1x7	14	char array
x	1x1	8	double array
y	1x1	8	double array
z	1x1	8	double array

Grand total is 18 elements using 102 bytes

```
>> clear x y t
```

```
>> who
```

Your variables are:

```
A B z
```

```
>> clear
```

```
>> who
```

```
>>
```

1.3 Obtenir de l'aide **FB/MEHDA**

Pour obtenir de l'aide on utilise la fonction help suivie du nom de la fonction.

Exemple :

```
>> help
```

```
HELP topics:
matlab\general - General purpose commands.
matlab\ops     - Operators and special characters.
matlab\elmat  - Elementary matrices and matrix manipulation.
matlab\elfun  - Elementary math functions.
matlab\matfun - Matrix functions - numerical linear algebra.
```

...

```
>> help clear
```

```
CLEAR Clear variables and functions from memory.
CLEAR removes all variables from the workspace.
CLEAR VARIABLES does the same thing.
CLEAR GLOBAL removes all global variables.
CLEAR FUNCTIONS removes all compiled M- and MEX-functions.
```

Autres fonctions :

```
helpwin -> aide en ligne dans une fenêtre séparée
lookfor -> recherche d'un mot clé
which -> localise fonctions et fichiers (exp : which CHOL)
what -> liste des fichiers matlab dans le répertoire courant (exp : C:\MATLAB6p5\work)
exist -> check si une fonction ou une variable existe dans le workspace (exp : exist var → 1 ou 0)
whos -> liste des variables dans le workspace
```

D'autres exemples seront traités en TP

1.4 Syntaxe d'une ligne d'instruction

- Si une instruction est suivie d'un point virgule (;) le résultat de cette instruction n'est pas affiché.
- Pour re-afficher un résultat contenu dans une variable il suffit de taper le nom de la variable.
- Le résultat de la dernière instruction exécutée peut être rappelé par la commande ans.

Exemple:

```
>> an = 2009 ↵
an=
2009
>> jour = 30 ; ↵
>>
>> jour ↵
jour =
30
```

```
>> B = 3*16 ; ↵
>>
>> B ↵
B =
48
>> 2*6 ↵
ans=
12
>> B+an ↵
ans=
2057
```

FB/MEHDA

- Plusieurs instructions Matlab peuvent figurer sur une même ligne. Il faut les séparer par une virgule ou par un point virgule.
- Si une instruction est précédée du symbole % l'instruction est ignoré par Matlab il l'a considère comme commentaire

Exemple :

```
>> x=5 ; y=0 ; z=1 ;
```

```
>> a=3, b=-2, c=1
```

```
a =
```

```
3
```

```
b =
```

```
-2
```

```
c =
```

```
1
```

```
>> % Calcul du discriminant Delta
```

```
>> D = b*b - 4*a*c ;
```

- Si une commande est trop longue pour tenir sur une ligne, il est possible de poursuivre sur la ligne suivante en terminant la ligne par 3 points (. . .).

```
>> cout_moyen = cout ... % commande sur deux lignes  
/ nombre;
```

```
>> t=x+2*y... ↙
```

```
+3*z-1 ↙
```

```
t =
```

```
7
```

```
>>
```

1.5 Messages d'erreurs

Si la syntaxe de l'instruction soumise est erronée ou si vous demandez à MATLAB d'exécuter une instruction illégale (qui n'a pas de sens mathématique par exemple), vous obtiendrez un message d'erreur. Ce message vous indique les sources d'erreurs possibles qui doit vous permettre de les corriger rapidement.

Exemple :

```
>> A=[1 2]; B=[0 1 5];
```

```
>> A + B
```

```
??? Error using ==> +  
Matrix dimensions must agree.
```

```
>> C = [1 2 3; 4 5]
```

```
??? Number of elements in each row must be the same.
```

```
>> whose
```

```
??? Undefined function or variable 'whose'.
```

```
>>
```

Dans la première instruction, on tente d'effectuer la somme de 2 matrices aux dimensions incompatibles. Dans le second exemple on tente de définir une matrice dont le nombre d'éléments dans chaque ligne diffère. Enfin la troisième instruction est inconnue de MATLAB: il ne s'agit n'y d'une fonction ni d'une variable incorporée ou utilisateur.

1.6 Les fichiers (.m) **FB/MEHDA**

Ces fichiers textes contiennent des lignes d'instructions MATLAB et ont une extension « .m ». Ils sont exécutés ligne par ligne par MATLAB. Ils peuvent être de 2 types différents, scripts ou fonctions.

1.6.1 Les scripts sous MATLAB

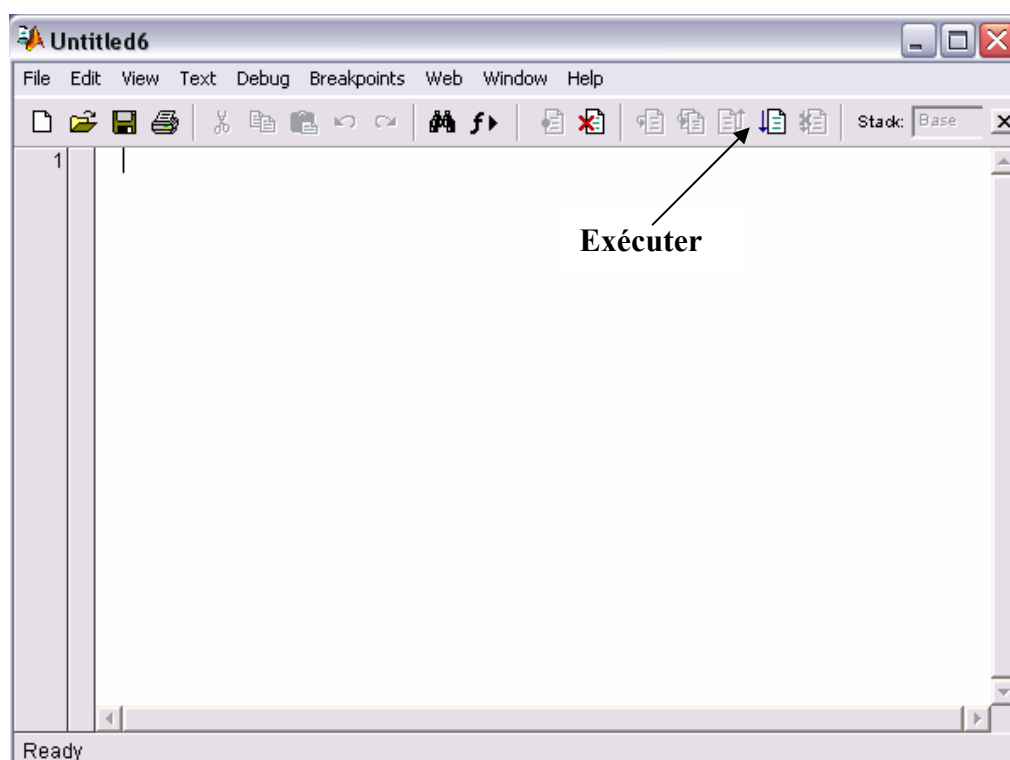
Les scripts sous Matlab sont équivalents aux procédures, ils ne prennent pas d'argument. Ils peuvent être exécutés directement en tapant simplement leur nom dans l'espace de travail MATLAB.

Les scripts partagent l'espace de travail de base (workspace) avec la session interactive Matlab et les autres scripts. Si vous utilisez des variables temporaires, indices de boucles, etc, il est conseillé de les supprimer de l'espace de travail à la fin du script avec la commande clear.

Les étapes à suivre pour la création d'un fichier script:

- Cliquer sur le menu File de la fenêtre Matlab
- Cliquer ensuite sur *New* puis cliquer sur *M-File* la fenêtre de l'éditeur de Matlab sera ouverte.
- Ecrire le programme voulu
- Pour sauvegarder le programme : cliquer sur le menu *File* de la fenêtre éditeur puis cliquer sur *save as* écrire le nom du fichier exemple (calcul) puis cliquer sur *enregistrer*.
- Pour exécuter le programme, Nous avons 2 manières :
 1. cliquer sur le menu *debug* puis sur *Run* s'il n'y a pas d'erreurs le programme sera exécuté. Pour voir le résultat il faut revenir à la fenêtre Matlab (workspace).
 2. la 2^{ème} méthode consiste à revenir à la fenêtre Matlab (fenêtre de commande) puis d'écrire le nom du fichier. Le résultat du programme sera affiché directement.

On peut aussi exécuter un fichier script dans un autre fichier script en tapant seulement le nom du fichier.



1.6.2 Les fonctions sous MATLAB **FB/MEHDA**

Les fichiers de fonctions ont deux rôles. Ils permettent à l'utilisateur de définir des fonctions qui ne figurent pas parmi les *fonctions incorporées* de MATLAB et de les utiliser de la même manière que ces dernières (ces fonctions sont nommées *fonctions utilisateur*). Ils sont également un élément important dans la programmation d'applications où les fonctions jouent le rôle des fonctions et procédures des langages de programmation usuels. Une fonction peut posséder des arguments d'entrée et des arguments de sortie.

La syntaxe la plus générale des fichiers *function* est la suivante :

```
function [vars1, vars2, ...,varsN]=nomfonct(vare1, vare2, ...,vareM)
...
    Séquence d'instructions
...
```

Avec :

- *Vars1*, ..., *varsN* sont les variables de sortie (arguments de sortie) de la fonction;
- *Vare1*, ..., *vareM* sont les variables d'entrée (arguments d'entrée) de la fonction;
- *Séquence d'instructions* est le corps de la fonction.

Le fichier doit impérativement commencer par le mot-clé *function*. Suit entre crochets les variables de sortie de la fonction, le symbole =, le nom de la fonction et enfin les variables d'entrée entre parenthèses. Si la fonction ne possède qu'une seule variable de sortie, les crochets sont inutiles. Il est impératif que la fonction ayant pour nom « *Nomfonct* » soit enregistrée dans un fichier de nom *Nomfonct.m* sans quoi cette fonction ne sera pas << visible >> par MATLAB.

L'appel d'une fonction utilisateur s'effectue de la même façon que l'appel de n'importe quelle fonction MATLAB:

```
[var_s1, var_s2, ...,var_sn]=nomfonct(var_e1, var_e2, ...,var_en) ;
```

➔ **Remarquer** que le mot « *function* » n'y figure pas.

Exemple :

```
function [Fn]=facto(N)
% Cette fonction calcule
% le factoriel de l'entier N
Fn=1;
for i=2:N
    Fn=Fn*i;
end
```

Les lignes précédentes doivent être enregistrées dans un fichier de nom « *facto.m* »

Les lignes précédées du symbole % sont des lignes de commentaire. Les lignes de commentaire situées entre la ligne **function** ... et la 1^{ère} ligne d'instructions sont affichées si l'on demande de l'aide sur la fonction *facto*.

```
>> help facto
    Cette fonction calcule
    le factoriel de l'entier N
>>
```


L'exécution de cette fonction à partir du workspace s'effectue comme suit :

```
>>N=5 ;  
>> [Fn]=facto(N)  
    Fn =  
        120  
>>
```

Règles et propriétés **FB/MEHDA**

- Le nom de la fonction et celui du fichier m-file qui en contient la définition doivent être identiques. Ce fichier est le fichier m-file associé à la fonction.
- La commande help affiche les premières lignes de la section de commentaires ;
- Chaque fonction possède son propre espace de travail et toute variable apparaissant dans le corps d'une fonction est locale à celle-ci. Toutefois: il est possible de déclarer certaines variables comme des *variables globales* . On déclare une variable globale grâce au mot clé **global**. Par exemple pour déclarer la variable numex globale on écrit *global numex*. Attention, la déclaration *global numex* doit être reprise dans chaque fonction utilisant *numex* comme variable.
- Un fichier m-file associé à une fonction (i.e. qui porte le nom d'une fonction et contient sa définition) peut contenir d'autres définitions de fonctions. La fonction qui partage son nom avec le fichier ou fonction principale doit apparaître en premier. Les autres fonctions ou fonctions internes peuvent être appelées par la fonction principale, mais pas par d'autres fonctions ou depuis la fenêtre de commande.
- Si le fichier ne commence pas par le mot-clé *function* on a tout simplement écrit un script!

Exemples : **FB/MEHDA**

1^{er} cas

```
function facto1
% Cette fonction calcule
% le factoriel de l'entier N
N=10 ;% sert d'argument
      d'entrée
Fn=1;
for i=2:N
    Fn=Fn*i;
end
Fn % sert à afficher le résultat
    % car il n'y a pas d'argument
    % de sortie.
```

Exécution :

```
>> facto1
Fn =
    120
```

2^{ème} cas

```
function facto2(N)
% Cette fonction calcule
% le factoriel de l'entier N
Fn=1;
for i=2:N
    Fn=Fn*i;
end
Fn % sert à afficher le résultat
    % car il n'y a pas
d'argument
    % de sortie.
```

Exécution :

```
>> facto2
??? Input argument 'N' is undefined.
Error in ==>
C:\MATLAB6p5\work\facto.m
On line 5 ==> for i=2:N

>> facto2(3)
Fn =
     6
```

3^{ème} cas

```
function [Fn]=facto3(N)
ou function Fn=facto3(N)% car 1
    seul argS

% Cette fonction calcule
% le factoriel de l'entier N
Fn=1;
for i=2:N
    Fn=Fn*i;
end
%Fn inutile car la fonction possède
% un argument de sortie.
```

Exécution :

```
>> facto3(4)
ans =
    24

ou
>> [X]=facto3(4)
X =
    24
```

4^{ème} cas

```
Function [Fn]= facto4
% Cette fonction calcule
% le factoriel de l'entier N
Global N
Fn=1;
for i=2:N
    Fn=Fn*i;
end
```

Exécution :

```
>> global N % à déclarer d'abord
>> N=6 ; % puis l'utiliser
>> facto4 % la fct connaît
maintenant N
ans =
    720
```

Chapitre 2: Types de données et variables

2.1 Particularités de MATLAB FB/MEHDA

Comme tout langage de programmation MATLAB permet de définir des données variables. Une variable est désignée par un identificateur qui est formé d'une combinaison de lettres et de chiffres. Le premier caractère de l'identificateur doit nécessairement être une lettre. Attention, MATLAB différencie majuscules et minuscules! Ainsi `x33` et `x33` désignent deux variables distinctes. Les variables sont définies au fur et à mesure que l'on donne leurs noms (identificateur) et leurs valeurs numériques ou leurs expressions mathématiques. L'utilisation de variables avec MATLAB ne nécessite pas de déclaration de type ou de dimension. Le type et la dimension d'une variable sont déterminés de manière automatique à partir de l'expression mathématique ou de la valeur affectée à la variable.

2.2 Les 4 types de données

Les 3 principaux types de variables utilisés par Matlab sont : les réels, les complexes et les chaînes de caractères. Le type logique est associé au résultat de certaines fonctions

Pour MATLAB toute variable est considérée comme étant un tableau d'éléments d'un type donné. MATLAB différencie trois formes particulières de tableaux. Les *scalaires* qui sont des tableaux à une ligne et une colonne. Les *vecteurs* qui sont des tableaux à une ligne ou à une colonne. Les *matrices* qui sont des tableaux ayant plusieurs lignes et colonnes. Une variable MATLAB est donc toujours un tableau que l'on appelle variable scalaire, vecteur ou matrice suivant la forme du tableau.

2.2.1 Le type complexe

L'unité imaginaire est désignée par i ou j (se transforme en i). les nombres complexes peuvent être écrits sous forme cartésienne $a+ib$ ou sous forme polaire re^{it} . Les différentes écritures possibles sont :

$$a+i*b \text{ (ou } a+b*i) \text{ et } r*\exp(i*t) \text{ (ou } r*\exp(t*i))$$

Avec a, b, r et t des variables de type réel.

Mais on peut écrire :

```
>> z = 2+i*5           >> z = 2+5*i
>> z = 2+5i          (mais z = 2+i5 error!  i5 = ???)
>> z = 7*exp(i*3)    >> z = 7*exp(3*i)           >> z = 7*exp(3i)
```

Voici quelque commande concernant les nombres complexes :

si Z est de type complexe

`imag(Z)` retourne la partie imaginaire de Z

`real(Z)` retourne la partie réelle de Z

`abs(Z)` retourne le module de Z

`angle(Z)` retourne la partie imaginaire de Z

`conj(Z)` retourne le conjugué de Z (Z^*)

Ces dernières commandes permettent de passer aisément de la forme polaire à la forme cartésienne.

Il est possible que des variables de noms *i* ou *j* aient été redéfinies au cours d'un calcul antérieur alors on peut soit détruire ces deux variables (`clear i,j`)

i et *j* redeviennent alors l'unité imaginaire, soit réaffecter à *i* ou à *j* la valeur unité imaginaire, soit l'instruction : `i = sqrt(-1)`.

2.2.2 Le type chaîne de caractères **FB/MEHDA**

Une chaîne de caractères est un tableau de caractères. Une donnée de type chaîne de caractère (`char`) est représentée sous la forme d'une suite de caractères encadrée d'apostrophes simples (').

Exemples:

```
>> ch1='bon'  
ch1 =  
    bon
```

```
>> ch2='jour'  
ch2 =  
    jour
```

```
>> whos  
Name      Size      Bytes   Class  
ch1       1x3         6   char array  
ch2       1x4         8   char array
```

Grand total is 7 elements using 14 bytes

```
>> ch=[ch1,ch2]  
ch =  
    bonjour  
>> ch(1)  
ans =  
    b  
>> ch(7)  
ans =  
    r  
>> ch(1:3)  
ans =  
    bon  
>> ch3='soi' ;  
>> ch=[ch(1:3),ch3,ch(7)]  
ch =  
    bonsoir
```

- Si une chaîne de caractères doit contenir le caractère apostrophe (') celui-ci doit être double dans la chaîne.

Exemple

```
rep='aujourd'hui'
```

```
??? rep='aujourd'hui'
```

Error: Missing MATLAB operator.

```
>> rep='aujourd'hui'
```

```
rep =  
    aujourd'hui
```

```
>> apos=""
```

```
apos =  
    ''
```

- La chaîne de caractères vide s'obtient par 2 apostrophes''.

2.2.3 Le type logique

Le type logique possède 2 formes : 0 pour faux et 1 pour vrai

Un résultat de type logique est retourné par certaines fonctions ou dans le cas de certains tests.

Exemple : **FB/MEHDA**

```
>> a=1; b=2;
>> test_E=(a==b)
test_E =
     0
>> test_S=(a>b)
test_S =
     0
>> test_I=(a<b)
test_I =
     1
>> V=true
V =
     1
>> F=false
F =
     0
```

```
>> whos
Name      Size      Bytes      Class
F         1x1         1      logical array
V         1x1         1      logical array
a         1x1         8      double array
b         1x1         8      double array
test_E    1x1         1      logical array
test_I    1x1         1      logical array
test_S    1x1         1      logical array

Grand total is 7 elements using 21 bytes
```

2.2.4 *Le type vecteur*

On définit un vecteur ligne en donnant la liste de ses éléments entre crochets ([]). Les éléments sont séparés au choix par des espaces ou par des virgules.

On définit un vecteur colonne en donnant la liste de ses éléments séparés au choix par des points virgules (;) ou par des retours chariots.

On peut transformer un vecteur ligne X en un vecteur colonne et réciproquement en tapant X' (' est le symbole de transposition)

On peut obtenir la longueur d'un vecteur donné grâce à la commande length(X).

Exemple :

```
>> x1=[1 2 3], x2=[4,5,6,7], x3=[8;9;10]
x1 =
     1     2     3
x2 =
     4     5     6     7
x3 =
     8
     9
    10

>> length(x2)
ans =
     4

>> x3'
ans =
     8     9    10
```

```
>> whos
Name      Size      Bytes      Class
x1        1x3        24      double array
x2        1x4        32      double array
x3        3x1        24      double array

Grand total is 10 elements using 80 bytes
```

2.2.5 Le type matrice **FB/MEHDA**

On définit une matrice en donnant la liste de ses éléments entre crochets. Les éléments d'une même ligne sont séparés au choix par des espaces ou par des virgules. Les lignes entre elles sont séparées par des retours chariot ou par des points virgules.

On peut obtenir les dimensions d'une matrice par la commande `size`. Soit `A` une matrice quelconque :

- `size(A,1)` donne le nombre de lignes.
- `Size(A,2)` donne le nombre de colonne.
- `Size(A)` donne le nombre de lignes et de colonnes → `[m,n]=Size(A)`

Exemple:

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

```
A =
```

```
1 2 3
4 5 6
7 8 9
8 7 9
```

```
>> size(A)
ans =
    4    3
>> size(A,1)
ans =
    4
>> size(A,2)
ans =
    3
```

```
>> B=[1,2
    3,4]
B= 1 2
    3 4
>> C=[2 0 9
    4 1 3]
C= 2 1 9
    4 1 3
```

Remarques :

➤ Comme on ne définit pas de manière explicite le type d'une variable, il est parfois utile de pouvoir le déterminer. Cela est possible grâce aux commandes `ischar`, `islogical` et `isreal`.

`ischar(x)` retourne 1 si `x` est de type chaîne de caractères et 0 sinon. `islogical(x)` retourne 1 si `x` est de type logique et 0 sinon. La commande `isreal(x)` est à utiliser avec discernement: elle retourne 1 si `x` est réel ou de type chaîne de caractères (`(-:)`) et 0 sinon (`x` est complexe à partie imaginaire non nulle ou n'est pas un tableau de valeurs réelles ou de caractères).

Exemple :

```
>> clear
>> x = 2; z = 2+i; rep = 'oui';
```

```
>> ischar(rep)
ans =
    1
>> ischar(x)
ans =
    0
>> isreal(z)
ans =
    0
>> isreal(x)
ans =
    1
>> isreal(rep)
ans =
    1
>>
```

➤ Variables spéciales **FB/MEHDA**

Ces noms de variables sont utilisés par Matlab :

pi ans inf i or j realmin realmax eps ans etc.

```
>> pi
ans =
    3.1416

>> ans
ans =
    3.1416

>> eps
ans =
    2.2204e-016

>> realmax
ans =
    1.7977e+308
```

```
>> realmin
ans =
    2.2251e-308

>> inf
ans =
    Inf

>> nan
ans =
    NaN

>>
```

2.3 Lecture des données

➤ pour lire une variable simple on utilise la fonction *input* comme suite :

```
>> x=input('introduire la valeur de x')
introduire la valeur de x4
x =
    4

>> x=input('introduire la valeur de x ')
introduire la valeur de x 4
x =
    4
```

La différence entre les deux exemples est l'espace créé dans le commentaire après x ;

➤ pour lire un vecteur ou une matrice on utilise la même fonction :

```
>> y=input('introduire les valeurs du vecteur Y ')
introduire les valeurs du vecteur Y [1 2 3]
y =
    1    2    3

>> Z=input('introduire les valeurs du vecteur Z: ');
introduire les valeurs du vecteur Z: [2 5 8 7]
>>

>> T=input('introduire les valeurs de la matrice T ')
introduire les valeurs de la matrice T [1 2 3; 4 5 6]
T =
    1    2    3
    4    5    6
```

➤ Si la chaîne de caractère contient une apostrophe il est indispensable de doubler l'apostrophe.

2.4 affichage des données **FB/MEHDA**

- pour afficher les données il suffit d'écrire le nom de la variable
- la commande **disp** permet d'afficher un tableau de valeurs numérique ou de caractères

Exemple

```
>> x=6;
>> disp(x)
6
>> disp('bonjour')
bonjour
>> a='bon'
a =
    bon
>> disp(['le mot est : ',a])
le mot est : bon
>> disp(['la valeur de x est: ',num2str(x)])
la valeur de x est: 6
```

num2str → Convertit un nombre en caractère.

Les formats d'affichage des réels

MATLAB dispose de plusieurs formats d'affichage des réels. Par défaut le format est le format court à 5 chiffres. Les autres principaux formats sont:

format long : format long à 15 chiffres.

format short e : format court à 5 chiffres avec notation en virgule flottante.

format long e : format long à 15 chiffres avec notation en virgule flottante.

MATLAB dispose également des formats **format short g** et **format long g** qui utilise la << meilleure >> des deux écritures à virgule fixe ou à virgule flottante. On obtiendra tous les formats d'affichage possibles en tapant `help format`. On impose un format d'affichage en tapant l'instruction de format correspondante dans la fenêtre de contrôle, par exemple `format long`. Pour revenir au format par défaut on utilise la commande `format` ou `format short`.

```
>> pi
ans =
    3.1416

>> format long
>> pi
ans =
    3.14159265358979

>> format short e
>> pi^3
ans =
    3.1006e+01

>> format short g
>> pi^3
ans =
    31.006
```

Exemple :

```
x = [4/3 1.2345e-6]
format short
    1.3333    0.0000
format short e
    1.3333e+000    1.2345e-006
format short g
    1.3333    1.2345e-006
format long
    1.333333333333333    0.00000123450000
format long e
    1.333333333333333e+000    1.234500000000000e-006
format long g
    1.333333333333333    1.2345e-006
format bank
    1.33    0.00
format rat
    4/3    1/810045
format hex
    3ff5555555555555    3eb4b6231abfd271
```


FB/MEHDA

- La commande `format` permet de choisir entre plusieurs modes d'affichage (sans interférer avec le type des valeurs numériques affichées qui est toujours le type `double`) :

Commande	Affichage	Exemple
<code>format short</code>	décimal à 5 chiffres	31.416
<code>format short e</code>	scientifique à 5 chiffres	31.416e+01
<code>format long</code>	décimal à 16 chiffres	31.4159265358979
<code>format long e</code>	scientifique à 16 chiffres	314159265358979e+01
<code>format hex</code>	hexadécimal	
<code>format bank</code>	virgule fixe à deux décimales	31.41
<code>format rat</code>	fractionnaire	3550/113
<code>format +</code>	utilise les symboles +, - et espace pour afficher les nombres positifs négatifs et nuls	+

2.5 Sauvegarde des données

Il est possible de sauvegarder une session MATLAB dans un fichier pour une utilisation ultérieure. L'instruction `save nom-fic` enregistre toutes les variables de l'espace de travail dans le fichier `nom-fic.mat`. Si aucun nom de fichier n'est précisé, le fichier par défaut est `matlab.mat`. Il est possible de ne sauvegarder qu'une partie des variables (par exemple seulement la variable contenant le résultat d'un calcul) en utilisant l'instruction `save nom-fic nom-var` où `nom-var` est le nom de la (ou des) variable(s) à sauvegarder. Attention, seul le contenu des variables est sauvegardé et non pas l'ensemble des instructions effectuées durant la session. Pour ramener dans l'espace de travail les variables sauvegardées dans le fichier `nom-fic.mat`, taper `load nom-fic`.

Exemple :

```
>> x=2*pi/3, y=sin(x), z=cos(x)
x =
    2.0944
y =
    0.8660
z =
   -0.5000

>> save data
>> save toto y z
>> who

Your variables are:

x y z

>> clear all
>> who
>>          % vide
```

```
>> load toto
>> who
Your variables are:
y z

>> y
y =
    0.8660

>> z
z =
   -0.5000

>> x
??? Undefined function or variable 'x'.

>> load data
>> who
Your variables are:
x y z

>>
```

(Voir aussi la commande `diary`)

Chapitre 3: Calculer avec Matlab

FB/MEHDA

3.1 Opération portant sur les scalaires

- Si X et Y sont des variables scalaires de type réel : $X+Y$, $X-Y$, $X*Y$ et X/Y désignent les 4 opérations dans R.
- Si X et Y sont des variables scalaire de type complexe : $X+Y$, $X-Y$, $X*Y$ et X/Y désignent les 4 opérations dans C.
- La puissance s'obtient (^) X^Y (X puissance Y)

3.2 Opération portant sur les vecteurs

Une particularité de MATLAB est de permettre d'effectuer des opérations de manière globale sur les éléments d'un vecteur de type réel ou complexe sans avoir à manipuler directement ses éléments :

- Si K est une variable scalaire et X un vecteur $K*X$ multiplie tous les éléments de X par K.
- Si X et Y sont des vecteurs de longueur identique $Z=X \pm Y$ définit le vecteur Z dont les éléments sont $Z(i)=X(i) \pm Y(i)$.
- Le produit des éléments des 2 vecteurs X et Y s'obtient par: $Z=X.*Y$
- Le quotient est donné par $Z=X./Y$

Exemple:

```
>> X=[1 2 3];
>> K=5;
>> K*X
ans =
     5     10     15

>> Y=[4 5 6];
>> Z=X+Y
Z =
     5     7     9

>> Z=X-Y
Z =
    -3    -3    -3

>> Z=X.*Y
Z =
     4    10    18

>> Z=X./Y
Z =
    0.2500    0.4000    0.5000
```

```
>> x = [1:10:100]; y=sqrt(x);
y =
Columns 1 through 7
1.0000 3.3166 4.5826 5.5678 6.4031 7.1414 7.8102
Columns 8 through 10
8.4261 9.0000 9.5394
>>
>> x=[3 1 2];
>> sum(x)
ans =
     6
>> prod(x)
ans =
     6
>> max(x)
ans =
     3
>> min(x)
ans =
     1
>> sort(x)
ans =
     1     2     3
>> fliplr(x)
ans =
     2     1     3
>>
```

3.3 Opération portant sur les matrices **FB/MEHDA**

Si A, B et C sont des matrices alors:

- $A*B$: désigne le produit de la matrice A par la matrice B
- $A\pm B$: désigne la somme ou la soustraction des 2 matrices A et B
- A^n : désigne la matrice A à la puissance n
- $A.*B$: désigne le produit élément par élément des 2 matrices A et B
- $A.^n$: désigne la matrice dont les termes sont égales aux termes de la matrice A à la puissance n
- Pour résoudre le système $AX=b$ soit on écrit $X=A\backslash b$ soit $X=inv(A)*b$

Exemple:

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

>> B=[4 5
      6 7];

>> A*B
ans =
     16     19
     36     43

>> A+B
ans =
     5     7
     9    11

>> A-B
ans =
    -3    -3
    -3    -3
```

```
>> n=2;

>> A^n
ans =
     7     10      (→ A×A...×A)
    15     22

>> A.*B
ans =
     4     10      (→ A(i,j)×B(i,j))
    18     28

>> A.^n
ans =
     1     4      (→ A(i,j)^n)
     9    16

>> b=[1;2];

>> A\b
ans =
     0      (→ A-1×b)
  0.5000

>> inv(A)*b
ans =
     0
  0.5000

>> A/b      (→ A×b-1)
??? Error using ==> /
Matrix dimensions must agree.
```

Chapitre 4: Programmer sous Matlab

4.1 Opérateurs de comparaison et opérateurs logiques **FB/MEHDA**

a) Les opérateurs de comparaison sont:

`==` : égale à ($X = Y$)

`>` : Strictement plus grand que ($X > Y$)

`<` : Strictement plus petit que ($X < Y$)

`>=` : plus grand ou égale à ($X \geq Y$)

`<=` : plus petit ou égale à ($X \leq Y$)

`~=` : différent de ($X \neq Y$)

b) Les opérateurs logiques sont:

`&` : et ($X \& Y$)

`|` : ou (or) ($X | Y$)

`-` : Non X ($\neg X$)

4.2 Instructions de contrôle

4.2.1 Boucle for (parcours d'un intervalle)

Sa syntaxe est:

for indice = borne_inf : pas : borne_sup

Séquence d'instructions

end

Exemple faire un programme Matlab qui calcule la somme suivante $\sum_{i=3}^n i$

Solution:

```
n=input('donner la valeur de n');
s=0;
for i=3:n
    s=s+i;
end
disp('la somme s est: '),s

%disp(['la somme s est: ',num2str(s)])
```

L'exécution:

```
>> ex1_matlab
donner la valeur de n6
la somme s est:
s =
    18
```

Exemple faire un programme Matlab qui calcule la somme suivante $1+1.2+1.4+1.6+1.8+2$

Solution: **FB/MEHDA**

```
clear all
s=0;
for i=1:0.2:2
    s=s+i
end
disp('la somme s est: '),s
```

L'exécution:

```
>> ex2_matlab
s =
    1
s =
    2.2000
s =
    3.6000
s =
    5.2000
s =
    7
s =
    9
la Somme s est:
s =
    9
```

4.2.2 Boucle While (tant que)

Sa syntaxe est:

while expression logique

Séquence d'instructions

end

Exemple: faire un programme sous matlab qui calcule la somme suivante:

$S=1+2/2! +3/3!+...$ on arrête le calcule quand $S>2.5$

Solution:

```
clear all
s=1;i=1,f=1;
while s<=2.5
    i=i+1
    f=f*i;
    s=s+i/f
end
```

```
>> ex3_matlab
i =
    1
i =
    2
s =
    2
i =
    3
s =
    2.5000
i =
    4
s =
    2.6667
```

4.2.3 L'instruction if (si)

1^{er} cas : Sa syntaxe est:

if *expression logique*

Séquence d'instructions

end

Exemple:

Faire un programme sous Matlab qui résout le problème suivant:

1. $y = x$ si $x < 0$
2. $y = x^2$ si $x > 0$
3. $y = 10$ si $x = 0$

Solution

```
clear all
x=input('introduire la valeur de x ');
if x<0
    y=x;
end
if x>0
    y=x^2;
end
if x==0
    y=10;
end
disp('la valeur de y est: '),y
```

L'exécution:

```
>> ex4_matlab
introduire la valeur de x 5
la valeur de y est:
y =
    25
```

2^{er} cas :Sa syntaxe est: **FB/MEHDA**

if expression logique

Séquence d'instructions

else

Séquence d'instructions

end

Exemple:

Faire un programme sous Matlab qui résout le problème suivant:

1. $y = x$ si $x < 0$
2. $y = x^2$ si $x \geq 0$

Solution

```
clear all
```

```
x=input('introduire la valeur de x ');
```

```
if x<0
```

```
    y=x;
```

```
else
```

```
    y=x^2;
```

```
end
```

```
disp('la valeur de y est: '),y
```

L'exécution:

```
>> ex4_matlab
```

```
introduire la valeur de x 5
```

```
la valeur de y est:
```

```
y =
```

```
    25
```

3^{er} cas :Sa syntaxe est:

if expression logique

Séquence d'instructions

elseif expression logique

Séquence d'instructions

elseif expression logique

Séquence d'instructions

.

.

else

Séquence d'instructions

end

Exemple:

Faire un programme sous Matlab qui résout le problème suivant:

$Y = x$ si $x < 0$

$Y = 10$ si $x = 0$

$Y = \sqrt{x}$ si $0 < x < 20$

$Y = x^2$ si $x = 20$

$Y = x^3$ si $x > 20$

Solution **FB/MEHDA**

```
clear all
x=input('introduire la valeur de x ');
if x<0
    y=x;
elseif x==0
    y=10;
elseif x==20
    y=x^2;
elseif x>0 & x<20
    y=sqrt(x);
else
    y=x^3;
end
disp('la valeur de y est: '),y
```

L'exécution:

```
>> ex5_matlab
introduire la valeur de x 60
la valeur de y est:
y =
    216000
```

4.2.4 L'instruction switch

Sa syntaxe est:

```
Switch var
case cst-1
    Séquence d'instructions-1
case cst-2
    Séquence d'instructions-2
.
.
.
case cst-N
    Séquence d'instructions-N
otherwise
    Séquence d'instructions par défaut
end
```

var: est une variable numérique ou chaîne de caractère.

cst-1, cst-2...cst-N: sont des constantes numérique ou chaîne de caractères.

Si l'instruction à exécuter est la même pour un ensemble de cas alors la syntaxe est :

Case {cst-1, cst-2,...}

Exemples: **FB/MEHDA**

```
jj=input('donner le jour 1 :7 ');
switch jj
case 1
    disp('samedi')
case 2
    disp('dimanche')
case 3
    disp('lundi')
case 4
    disp('mardi')
case 5
    disp('mercredi')
case 6
    disp('jeudi')
case 7
    disp('vendredi')
end
%
```

```
mm=input('donner le mois: ');
switch mm
case 'Ja'
    disp('Janvier')
case 'F'
    disp('Février')
case 'M'
    disp('Mars')
case 'Av'
    disp('Avril')
otherwise
    disp('autre')
end
```

A l'exécution:

```
>> exp_switch
donner le jour: 4
mardi
donner le mois: 'Av'
Avril
>>
```

4.3 Instructions d'interruption d'une boucle

Il est possible de provoquer une sortie prématurée d'une boucle de contrôle.

- L'instruction **break** permet de sortir d'une *boucle for* ou d'une *boucle while*. En cas de boucles imbriquées, on interrompt seulement l'exécution de la boucle intérieure contenant l'instruction *break*.
- L'instruction **return** provoque un retour au programme appelant (ou au clavier). Les instructions suivant le *return* ne sont donc pas exécutées. L'instruction *return* est souvent utilisée conjointement avec une instruction conditionnée par exemple pour tester dans le corps d'une fonction si les paramètres d'entrée ont les valeurs attendues.
- L'instruction **error** permet d'arrêter un programme et d'afficher un message d'erreur. La syntaxe est : `error(' message d"erreur ')`.
- L'instruction **warning** permet d'afficher un message de mise en garde sans suspendre l'exécution du programme. La syntaxe est `warning(' message de mise en garde ')`. Il est possible d'indiquer à MATLAB de ne pas afficher les messages de mise en garde d'un programme en tapant `warning off` dans la fenêtre de commandes. On rétablit l'affichage en tapant `warning on`.
- *pause* : interrompt l'exécution jusqu'à ce que l'utilisateur tape un return
- *pause(n)* : interrompt l'exécution pendant *n* secondes.
- *pause off* : indique que les *pause* rencontrées ultérieurement doivent être ignorées, ce qui permet de faire tourner tous seuls des scripts requérant normalement l'intervention de l'utilisateur.

Exemples: **FB/MEHDA**

a- Commande break

```
S=0;
for i=1:10
    i
    S=S+i^2
    if S > 15
        break
    end
end
```

A l'exécution:

```
i =
    1
S =
    1
i =
    2
S =
    5
i =
    3
S =
   14
i =
    4
S =
   30
>>
```

b- Commande return

```
function somme(n)
%
if n < 0
    return
end
S=0;
for i=1:n
    S=S+i^2
end
```

A l'exécution:

```
>> somme(2)
S =
    1
S =
    5
>> somme(-2)

>> → sortie de la fonction
```

c- Commande error

```
clear all
s=2;
for i=1:5
    i
    s=s+i^2
    if s>5
        error('s est superieur à 5')
    end
end
```

A l'exécution:

```
i =
    1
s =
    3
i =
    2
s =
    7
??? Error using ==> myfile
s est superieur à 5
>>
```

d- Commande *warning* **FB/MEHDA**

```
clear all
% warning off/on → afficher ou masquer warning
s=2;
for i=1:5
    i
    s=s+i^2
    if s>5
        warning('s est superieur à 5')
    end
end
end
```

A l'exécution:

```
>>
i =
    1
s =
    3
i =
    2
s =
    7
Warning: s est superieur à 5

> In C:\MATLAB6p5\work\exp.m at line 8
i =
    3
s =
   16
Warning: s est superieur à 5

> In C:\MATLAB6p5\work\exp.m at line 8
i =
    4
s =
   32
Warning: s est superieur à 5

> In C:\MATLAB6p5\work\exp.m at line 8
i =
    5
s =
   57
Warning: s est superieur à 5

> In C:\MATLAB6p5\work\exp.m at line 8
>>
```

e- Commande *pause*

```
clear all
s=2;
for i=1:5
    i
    s=s+i^2
    if s>5
        pause
    end
end
end
```

A l'exécution:

```
>>
i =
    1
s =
    3
i =
    2
s =
    7
| ← Le curseur clignote ⇒ tapez Entrée
```

4.4 La programmation vectorielle

4.4.1 Manipulation des vecteurs

Les éléments d'un vecteur peuvent être manipulés grâce à leur indice dans le tableau. Le k -ième élément du vecteur x est désignée par $x(k)$. Le premier élément d'un vecteur a obligatoirement pour indice 1.

Il est possible de manipuler plusieurs éléments d'un vecteur simultanément. Ainsi les éléments k à l du vecteur x sont désignés par $x(k:l)$. On peut également manipuler facilement les éléments d'un vecteur dont les indices sont en progression arithmétique. Ainsi si l'on souhaite extraire les éléments $k, k+p, k+2p, \dots, k+Np = l$, on écrira $x(k:p:l)$. Plus généralement, si K est un vecteur de valeurs entières, $X(K)$ retourne les éléments du vecteur X dont les indices sont les éléments du vecteur K .

Exemple :

```
>> x = [ 1      2      3      4      5      6      7      8      9     10 ] ;
>> x(5)
ans =
     5
>> x(4:10)
ans =
     4     5     6     7     8     9     10

>> x(2:2:10)
ans =
     2     4     6     8     10

>> K = [1 3 4 6]; X(K)
ans =
     1     3     4     6
>>
```

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 , de premier terme a et de dernier terme b , on écrira $x = a:h:b$. Si $a-b$ n'est pas un multiple de h , le dernier élément du vecteur x sera $a + \text{ent}((a-b)/h) h$ où ent est la fonction partie entière.

```
>> x = 1.1:0.1:1.9
x =
Columns 1 through 7
    1.1000    1.2000    1.3000    1.4000    1.5000    1.6000    1.7000
Columns 8 through 9
    1.8000    1.9000

>> x = 1.1:0.4:2
x =
    1.1000    1.5000    1.9000
```

La commande **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 pas $h=(b-a)/(N-1)$). Les composantes du vecteur sont donc *linéairement espacées*.

La syntaxe est : $x = \text{linspace}(a, b, N)$.

```
>> x = linspace(1.1,1.9,9)
ans =
Columns 1 through 7
    1.1000    1.2000    1.3000    1.4000    1.5000    1.6000    1.7000
Columns 8 through 9
    1.8000    1.9000
```

- `linspace(a,b)` \Rightarrow $N=100$ par défaut.

Vecteurs spéciaux **FB/MEHDA**

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

4.4.2 Manipulation des matrices

Définir une matrice

On a déjà vu que l'on définissait la matrice : $A = \begin{pmatrix} 1 & 3 \\ 4 & 2 \end{pmatrix}$ en tapant `A = [1 3; 4 2]`. D'une façon générale donc, on définit une matrice en donnant la liste de ses éléments entre crochets.

On peut construire très simplement une matrice « par blocs ». Si A, B, C, D désignent 4 matrices (aux dimensions compatibles), on définit la matrice bloc :

$$K = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

par l'instruction `K = [A B ; C D]`.

Voici un exemple de construction par blocs de la matrice

```
>> A11 = [35 1 6; 3 32 7; 31 9 2];
```

```
>> A12 = [26 19; 21 23; 22 27];
```

```
>> A21 = [ 8 28 33; 30 5 34];
```

```
>> A22 = [17 10; 12 14];
```

$$B = \begin{pmatrix} 35 & 1 & 6 & 26 & 19 & 24 \\ 3 & 32 & 7 & 21 & 23 & 25 \\ 31 & 9 & 2 & 22 & 27 & 20 \\ \hline 8 & 28 & 33 & 17 & 10 & 15 \\ 30 & 5 & 34 & 12 & 14 & 16 \\ \hline 4 & 36 & 29 & 13 & 18 & 11 \end{pmatrix}$$

```

>> B11 = [ A11 A12; A21 A22 ]
B11 =
    35     1     6    26    19
     3    32     7    21    23
    31     9     2    22    27
     8    28    33    17    10
    30     5    34    12    14
>> B12 = [ 24 25 20 15 16]';
>> B = [ B11 B12];
>> B21 = [ 4 36 29 13 18 11];
>> B = [B ; B21]
B =
    35     1     6    26    19    24
     3    32     7    21    23    25
    31     9     2    22    27    20
     8    28    33    17    10    15
    30     5    34    12    14    16
     4    36    29    13    18    11
>>

```

Un élément d'une matrice est référencé par ses numéros de ligne et de colonne. $A(i,j)$ désigne le $i^{\text{ème}}$ élément de la $j^{\text{ème}}$ ligne de la matrice A. Ainsi $A(2,1)$ désigne le premier élément de la deuxième ligne de A,

```

>> A(2,1)
ans =
     4
>>

```

Matrices spéciales **FB/MEHDA**

Certaines matrices se construisent très simplement grâce à des commandes dédiées. Citons les plus utilisées:

eye(n) : la matrice identité de dimension n

ones(m,n) : la matrice à m lignes et n colonnes dont tous les éléments valent 1

zeros(m,n) : la matrice à m lignes et n colonnes dont tous les éléments valent 0

rand(m,n) : une matrice à m lignes et n colonnes dont les éléments sont générés de manière aléatoire entre 0 et 1

magic(n) : permet d'obtenir une matrice magique de dimension n.

```
>> eye(3)
ans =
    1    0    0
    0    1    0
    0    0    1

>> ones(3,2)
ans =
    1    1
    1    1
    1    1

>> zeros(2)
ans =
    0    0
    0    0
```

```
>> rand(2,3)
ans =
    0.4565    0.8214    0.6154
    0.0185    0.4447    0.7919

>> magic(3)
ans =
    8    1    6           la → somme
= 15
    3    5    7
    4    9    2
```

Le symbole deux-points (:) permet d'extraire simplement des lignes ou des colonnes d'une matrice. Le $j^{\text{ème}}$ vecteur colonne de la matrice A est désigné par A(:,j). C'est simple, il suffit de traduire le symbole deux-points (:) par << tout >>. Ainsi A(:,j) désigne toutes les lignes et la $j^{\text{ème}}$ colonne de la matrice A. Bien entendu, la $i^{\text{ème}}$ ligne de la matrice A est désignée par A(i,:).

```
>> A = magic(5)
A =
    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
```

```
>> A(1,:)
ans =
    17    24     1     8
15
>> A(:,2)
ans =
    24
     5
     6
    12
    18
```

Exercice :
Comment échanger les colonnes 2 et 3 de la matrice A ?

```
>> v = A(:,2); A(:,2) = A(:,3); A(:,3) = v;
A =
    17     1    24     8    15
    23     7     5    14    16
     4    13     6    20    22
    10    19    12    21     3
    11    25    18     2     9
```

On peut également extraire plusieurs lignes ou colonnes simultanément. Si J est un vecteur d'entiers, A(:,J) est la matrice issue de A dont les colonnes sont les colonnes de la matrice A d'indices contenus dans le vecteur J. De même A(J,:) est la matrice issue de A dont les lignes sont les lignes de la matrice A d'indices contenus dans le vecteur J. D'une façon plus générale, il est possible de n'extraire qu'une partie des éléments des lignes et colonnes d'une matrice. Si L et C sont deux vecteurs d'indices, A(L,C) désigne la matrice issue de la matrice A dont les éléments sont les A(i,j) tels que i soit dans L et j soit dans C.

```
>> A = magic(5)
A =
    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9

>> L = [1 3 5]; C = [3 4];
```

```
>> A(L,C)
ans =
     1     8
    13    20
    25     2

>> A(1:2:5,3:4)
ans =
     1     8
    13    20
    25     2

>>
```

Il existe des commandes MATLAB permettant de manipuler globalement des matrices.

La commande `diag` permet d'extraire la diagonale d'une matrice: si A est une matrice, $v=\text{diag}(A)$ est le vecteur composé des éléments diagonaux de A . Elle permet aussi de créer une matrice de diagonale fixée: si v est un vecteur de dimension n , $A=\text{diag}(v)$ est la matrice diagonale dont la diagonale est v .

```
>> A=eye(3)
A =
     1     0     0
     0     1     0
     0     0     1

>> diag(A)
ans =
     1
     1
     1
```

```
>> v=[1:3]
v =
     1     2     3

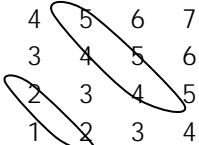
>> diag(v)
ans =
     1     0     0
     0     2     0
     0     0     3

>>
```

On n'est pas obligé de se limiter à la diagonale principale. La commande **diag** admet un second paramètre k pour désigner la k sur-diagonale (si $k>0$) ou la k sous-diagonale (si $k<0$).

```
>> A = [4 5 6 7 ; 3 4 5 6
        2 3 4 5; 1 2 3 4]
```

```
A =
     4     5     6     7
     3     4     5     6
     2     3     4     5
     1     2     3     4
```



```
>> diag(A,1)
ans =
     5
     5
     5

>> diag(A,-2)
ans =
     2
     2

>>
```

On dispose également de la commande **tril** permet d'obtenir la partie triangulaire inférieure (l pour lower) d'une matrice. La commande **triu** permet d'obtenir la partie triangulaire supérieure (u pour upper) d'une matrice.


```
>> A = [2 1 1; -1 2 1; -1 -1 2]
A =
     2     1     1
    -1     2     1
    -1    -1     2

>> triu(A)
ans =
     2     1     1
     0     2     1
     0     0     2

>> tril(A)
ans =
     2     0     0
    -1     2     0
    -1    -1     2

>>
```

Autres fonctions :

inv(A) : renvoie l'inverse de la matrice carrée A

det(A) : renvoie le déterminant de la matrice carrée A

A' : renvoie la transposée de la matrice A à coefficients réels

nnz(A) : renvoie le nombre d'éléments non nuls de la matrice A.

```
>> A = [3 1 2; -1 4 1; -2 -1 7]
A =
     3     1     2
    -1     4     1
    -2    -1     7

>> det(A)
ans =
    110

>> inv(A)
ans =
    0.2636 -0.0818 -0.0636
    0.0455  0.2273 -0.0455
    0.0818  0.0091  0.1182

>> A'
ans =
     3    -1    -2
     1     4    -1
     2     1     7

>> nnz(A)
ans =
     9

>>
```

La structure *sparse*

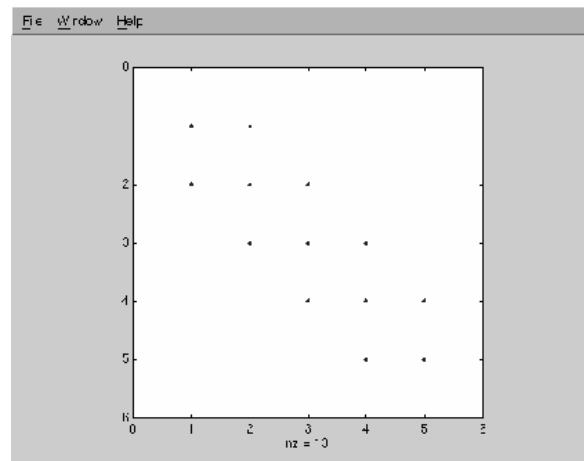
On appelle *matrice creuse* (*sparse matrix* en anglais) une matrice comportant une forte proportion de coefficients nuls. De nombreux problèmes issus de la physique conduisent à l'analyse de systèmes linéaires à matrice creuse. L'intérêt de telles matrices résulte non seulement de la réduction de la place mémoire (on ne stocke pas les zéros) mais aussi de la réduction du nombre d'opérations (on n'effectuera pas les opérations portant sur les zéros). Par défaut dans MATLAB une matrice est considérée comme *pleine* (ou *full* en anglais), c'est-à-dire que tous ses coefficients sont mémorisés.

Si M est une matrice, la commande **sparse(M)** permet d'obtenir la même matrice mais stockée sous la forme *sparse*. Si l'on a une matrice stockée sous la forme *sparse*, on peut obtenir la même matrice stockée sous la forme ordinaire par la commande **full**. Il est possible de visualiser graphiquement la structure d'une matrice grâce à la commande **spy**. Si M est une matrice, la commande **spy(M)** ouvre une fenêtre graphique et affiche sous forme de croix les éléments non-nuls de la matrice. Le numéro des lignes est porté sur l'axe des ordonnées, celui des colonnes en abscisse. On obtient également le nombre d'éléments non-nuls de la matrice.

Exemple :

```
>> N=5;
>> A=diag(2*ones(N,1)) - diag(ones(N-1,1),1) - diag(ones(N-1,1),-1)
A =
     2    -1     0     0     0
    -1     2    -1     0     0
     0    -1     2    -1     0
     0     0    -1     2    -1
     0     0     0    -1     2
>> nnz(A)
ans =
    13
>> whos
Name      Size      Bytes  Class
A         5x5        200   double array
B         5x5        180   sparse array
N         1x1         8     double array
Grand total is 39 elements using 388 bytes
>> spy(A)
```

```
>> B = sparse(A)
B =
(1,1)    2
(2,1)   -1
(1,2)   -1
(2,2)    2
(3,2)   -1
(2,3)   -1
(3,3)    2
(4,3)   -1
(3,4)   -1
(4,4)    2
(5,4)   -1
(4,5)   -1
(5,5)    2
```



Chapitre 5: Le graphisme sous Matlab

FB/MEHDA

V.1 Tracer le graphe d'une fonction; la commande fplot

La commande `fplot` permet de tracer le graphe d'une fonction sur un intervalle donné. La syntaxe est:

```
fplot('nomf', [xmin , xmax])
```

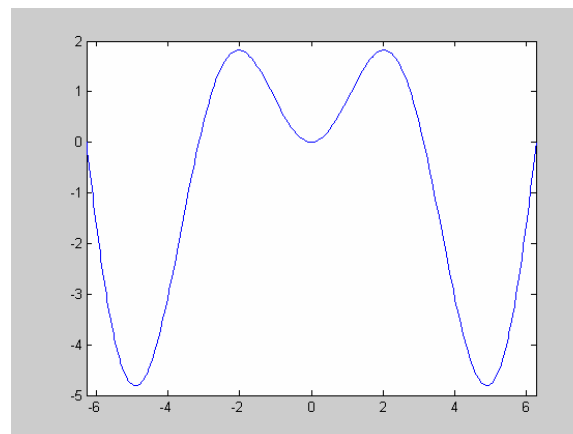
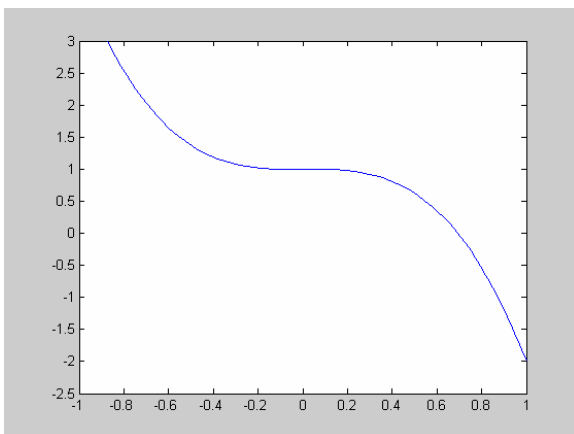
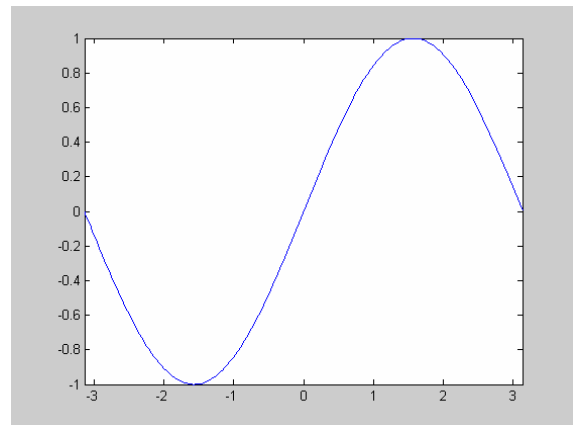
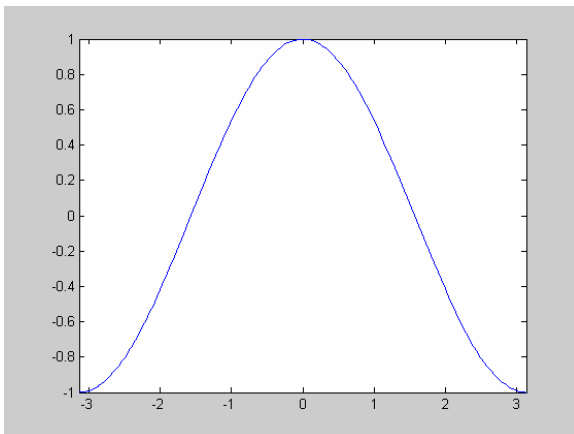
où

- `nomf` est soit le nom d'une fonction MATLAB incorporée, soit une expression définissant une fonction de la variable `x`, soit le nom d'une fonction utilisateur.
- `[xmin , xmax]` est l'intervalle pour lequel est tracé le graphe de la fonction.

Exemples :

```
figure(1),fplot('cos',[pi -pi])  
figure(2),fplot('sin',[pi -pi])  
figure(3),fplot('-3*x^3+1',[-1 +1 -2.5 3])  
figure(4),fplot('exp',[pi -pi])
```

```
function y=MaFonction(x)  
y=x.*sin(x);
```

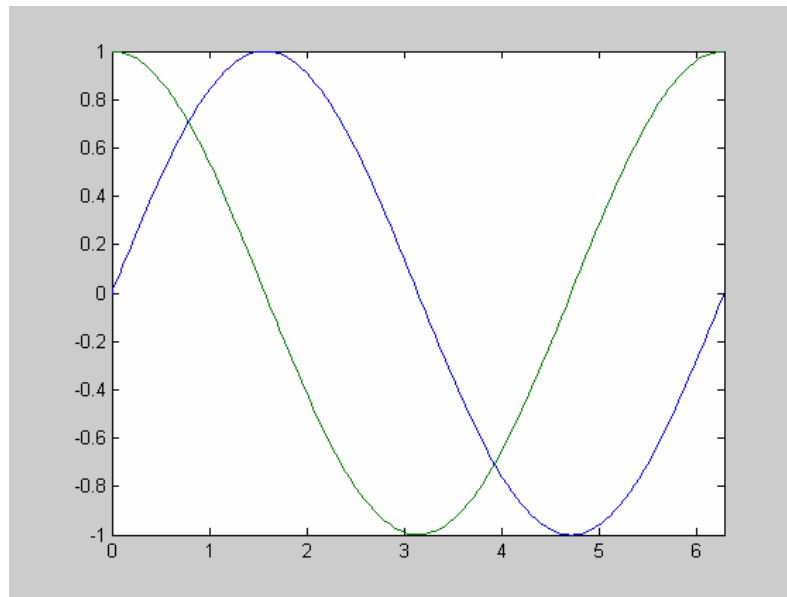


Il est possible de tracer plusieurs fonctions sur la même figure. Il faut pour cela utiliser la commande `fplot` de la manière suivante:

`fplot('nom_f1 , nom_f2 , nom_f3', [x_min , x_max])` où `nom_f1`, `nom_f2`, `nom_f3` est soit le nom d'une fonction MATLAB incorporée, soit une expression définissant une fonction de la variable `x`, soit le nom d'une fonction utilisateur.

Il est également possible de gérer les bornes des valeurs en ordonnées. Pour limiter le graphe aux ordonnées comprises entre les valeurs `y_min` et `y_max` on passera comme second argument de la commande `fplot` le tableau `[x_min , x_max , y_min , y_max]`.

Exemple : `fplot('sin(x), cos(x)', [0 2*pi])`



V.2 La commande plot **FB/MEHDA**

La commande `plot` permet de tracer un ensemble de points de coordonnées (x_i, y_i) $i = 1, \dots, N$. La syntaxe est `plot(x,y)` où `x` est le vecteur contenant les valeurs x_i en abscisse et `y` est le vecteur contenant les valeurs y_i en ordonnée. Bien entendu les vecteurs `x` et `y` doivent être de même dimension mais il peut s'agir de vecteurs lignes ou colonnes. Par défaut, les points (x_i, y_i) sont reliés entre eux par des segments de droites.

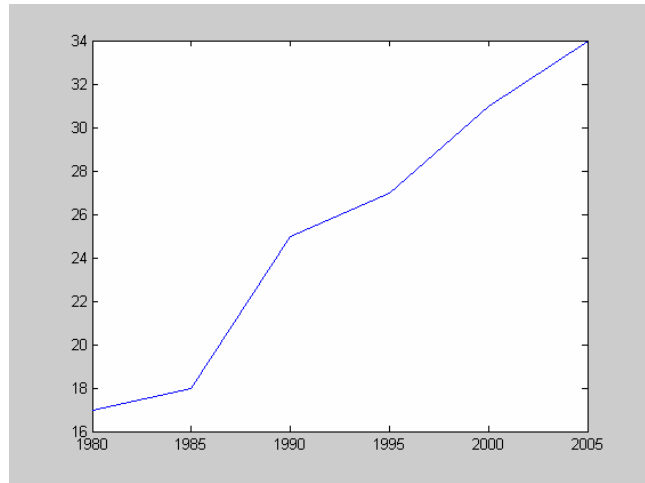
Voici par exemple une autre façon de tracer le graphe de la fonction $h(x) = x \sin(x)$ entre -2π et 2π ,

```
>> x=[-2*pi:0.01:2*pi]; y = x.*sin(x);  
>> plot(x,y)
```

FB/MEHDA

Exemple 01:

```
>> A=[1980 1985 1990 1995 2000 2005];  
>> P=[17 18 25 27 31 34];  
>> plot(A,P) % par défaut
```



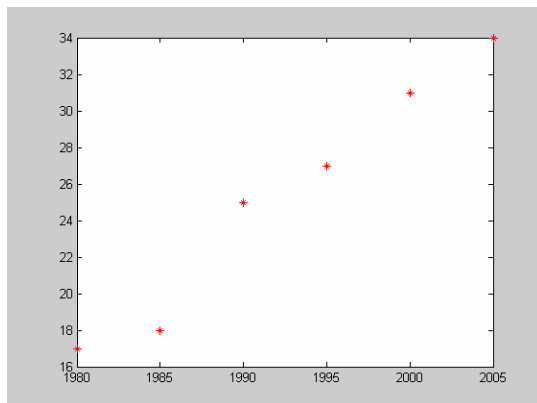
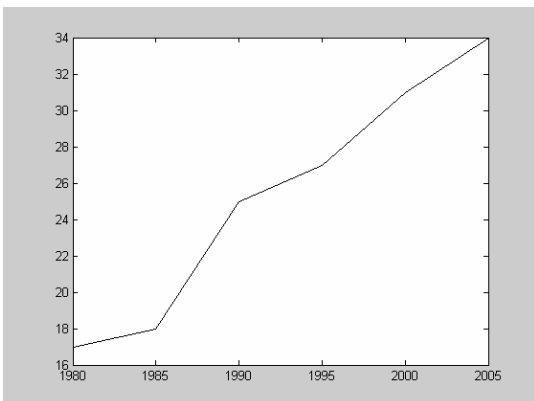
On peut spécifier à MATLAB quelle doit être la couleur d'une courbe, quel doit être le style de trait et/ou quel doit être le symbole à chaque point (x_i, y_i) . Pour cela on donne un troisième paramètre d'entrée à la commande `plot` qui est une chaîne de 3 caractères de la forme 'cst' avec c désignant la couleur du trait, s le symbole du point et t le style de trait. Les possibilités sont les suivantes:

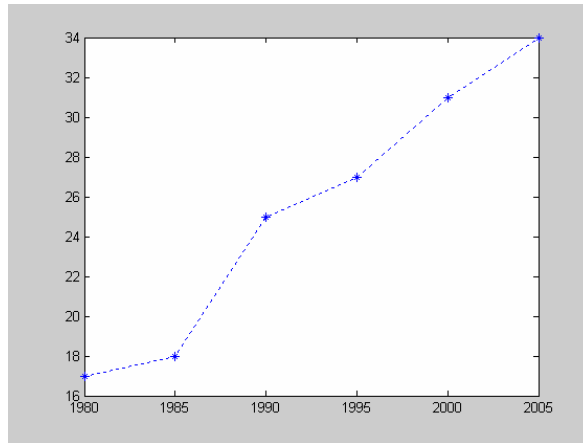
y	jaune	.	point	-	trait plein
m	magenta	o	cercle	:	pointillé court
c	cyan	x	marque x	--	pointillé long
r	rouge	+	plus	-.	pointillé mixte
g	vert	*	étoile		
b	bleu	s	carré		
w	blanc	d	losange		
k	noir	v	triangle (bas)		
		^	triangle (haut)		
		<	triangle (gauche)		
		>	triangle (droit)		
		p	pentagone		
		h	hexagone		

Les valeurs par défaut sont c = b, s = . et t = - ce qui correspond à un trait plein

Exemple 02:

```
>> A=[1980 1985 1990 1995 2000 2005]; P=[17 18 25 27 31 34];  
>> plot(A,P)% par défaut  
>> plot(A,P,'k')% couleur(noir)  
>> plot(A,P,'r*')%couleur(rouge) + Symbole en chaque point (xi,yi) (étoile)  
>> plot(A,P,'b*:')% couleur + Symbole en chaque point (xi,yi) + Style de trait (pointillé court)
```





V.3 Afficher plusieurs courbes dans une même fenêtre

Il est possible d'afficher plusieurs courbes dans une même fenêtre graphique grâce à la commande `hold on`. Les résultats de toutes les instructions graphiques exécutées après appel à la commande `hold on` seront superposés sur la fenêtre graphique active. Pour rétablir la situation antérieure (le résultat d'une nouvelle instruction graphique remplace dans la fenêtre graphique le dessin précédent) on tapera `hold off`.

Voici un exemple d'utilisation de la commande `hold on`. Le résultat est présenté à la figure 10.

```
>> e = exp(1);
>> figure
>> hold on
>> fplot('exp',[-1 1])
>> fplot('log',[1/e e])
>> plot([-1:0.01:e],[-1:0.01:e])
>> grid
>> hold off
```

V.4 LA COMMANDE SUBPLOT

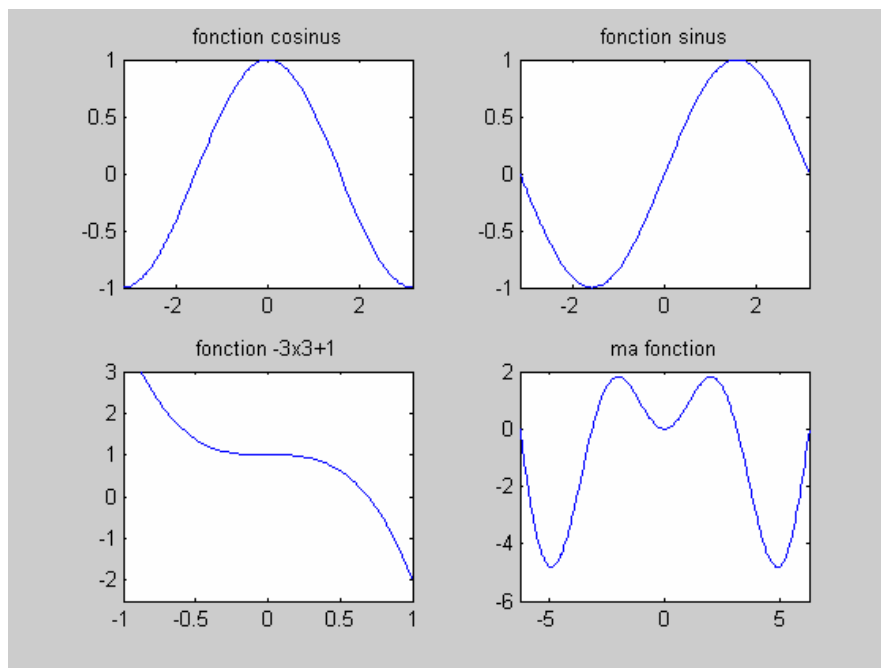
Il est possible de décomposer une fenêtre en sous-fenêtres et d'afficher une figure différente sur chacune de ces sous-fenêtres grâce à la commande `subplot`. La syntaxe est

```
subplot(m,n,i)
```

où

- `m` est le nombre de sous-fenêtres verticalement;
- `n` est le nombre de sous-fenêtres horizontalement;
- `i` sert à spécifier dans quelle sous-fenêtre doit s'effectuer l'affichage. Les fenêtres sont numérotées de gauche à droite et de haut en bas.

```
subplot(2,2,1)
fplot('cos',[pi -pi]),title('fonction cosinus')
subplot(2,2,2)
fplot('sin',[pi -pi]),title('fonction sinus')
subplot(2,2,3)
fplot('-3*x^3+1',[-1 +1 -2.5 3]),title('fonction -3x3+1')
subplot(2,2,4)
fplot('exp',[pi -pi]),title('fonction exp')
```



Quelques fonctions importantes sous MATLAB

FB/MEHDA

Les constantes :

Pi : 3.141592653897

i : $\sqrt{-1}$

j : $\sqrt{-1}$

eps : précision numérique relative

realmin : plus petit nombre à virgule flottante manipulable

realmax : plus grand nombre à virgule flottante manipulable

inf : infini qui est obtenu quand on essaie d'évaluer une expression dont le résultat excède realmax

NaN : not-a-number qui est obtenu quand on essaie d'effectuer une opération non défini comme 0/0

Fonctions portant sur les scalaires :

m et n sont des entiers :

rem(m,n) : donne le reste de la division entière

lcm(m,m) : donne le plus petit multiple commun

gcd(m,n) : donne le plus grand multiple commun

factor(n) : donne les termes de la décompositions en facteurs premier de l'entier n

Les fonctions mathématique incorporées :

x real :

log(x) : logarithme népérien de x

log10(x) : logarithme en base de 10 de x

exp(x) : exponentielle de x

sqrt(x) : racine carrées de x

abs(x) : valeur absolue de x

sign(x) : fonction valant 1 si x est positif ou 0 si non

si z est un complexe alors :

conj(z) : le conjugué de z,

abs(z) : le module de z,

angle(z) : argument de z,

real(z) : partie réelle de z,

imag(z) : partie imaginaire de z.

Les fonctions d'arrondis

round(x) : entier le plus proche de x

floor(x) : arrondi par défaut

ceil(x) : arrondis par excès

fix(x) : arrondi par défaut un réel positif et par excès un réel négatif

cos : cosinus
acos : cosinus inverse(arccos)
sin : sinus
asin: sinus inverse(arcsin)
tan: tangente
atan: tangente inverse (arctan)
cosh: cosinus hyperbolique(ch)
acosh: cosinus hyperbolique inverse(argch)
sinh: sinus hyperbolique(sh)
asinh: sinus hyperbolique inverse(argsh)
tanh: tangente hyperbolique (th)
atanh: tangente hyperbolique inverse(argth)

Fonctions portants sur les vecteurs :

x et y sont des vecteurs

cross(x,y) : permet de calculer le produit vectoriel des deux vecteurs x et y
sum(x.*y) : donne le produit des 2 vecteurs
sum(x) : donne la somme des éléments du vecteur x
prod(x) : donne le produit des éléments du vecteur x
max(x) : donne le plus grand élément du vecteur x
min(x) : donne le plus petit élément du vecteur x
mean(x) : donne la moyenne des éléments du vecteur x
sort(x) : ordonne les éléments du vecteur x par ordre croissant
fliplr(x) : échange la position des éléments de x
all(x) : donne 1 si tous les éléments de x sont différents de 0 et 0 si au moins un élément vaut 0
any(x) : retourne 1 si au moins un élément de x est différent de 0 et 0 si x est composé seulement de 0

Fonctions portants sur les matrices :

A est une matrice

eig(A) : renvoie les valeurs propres de la matrice carrée A
[V,D]=eig(A) : renvoie une matrice diagonale D formée des valeurs propres de A et une matrice V dont les vecteurs colonnes sont les vecteurs propres correspondant.
inv(A) : renvoie l'inverse de la matrice carrée A
rank(A) : renvoie le rang de la matrice carrée A
trace(A) : renvoie la trace de la matrice A
expm(A) : renvoie l'exponentielle matricielle de a

On peut obtenir les différentes normes d'une matrice A grâce à la commande **norm**.

norm(A) : renvoie la norme 2 de la matrice A.

norm(A,2) : même chose que **norm(A)**.

norm(A,1) : norme 1 de la matrice A, $\|A\|_1 = \max_{1 \leq j \leq n} \sum_{1 \leq i \leq n} |a_{ij}|$.

norm(A,inf) : norme infini de la matrice A, $\|A\|_\infty = \max_{1 \leq i \leq n} \sum_{1 \leq j \leq n} |a_{ij}|$.

Principales instructions graphiques :

Instruction	Description
<code>plot(x,y)</code>	tracé de la courbe passant par les points (x,y)
<code>loglog(x,y)</code>	idem avec échelle logarithmique sur les deux axes
<code>semilogx(x,y)</code>	idem avec échelle logarithmique sur l'axe Ox
<code>semilogy(x,y)</code>	idem avec échelle logarithmique sur l'axe Oy
<code>plotyy(x,y,x,z)</code>	courbe (x,y) avec l'axe Oy à gauche, et courbe (x,z) avec l'axe Oz à droite
<code>xlabel('label')</code>	légende pour l'axe Ox
<code>ylabel('label')</code>	légende pour l'axe Oy
<code>title('label')</code>	titre au dessus du graphique
<code>legend('lab1','lab2','lab3',...)</code>	légende avec une chaîne de caractères pour chaque courbe
<code>text(x,y,'label')</code>	chaîne de caractères à la position x,y
<code>plot3(x,y,z)</code>	tracé de la surface passant par les points (x,y,z)
<code>hold on, hold off</code>	active/désactive la conservation de la fenêtre graphique
	à l'appel de la fonction <code>plot</code>

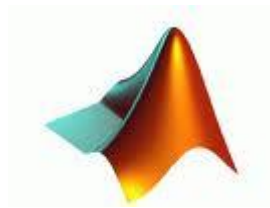
Tableau 1: Notations

<code>[]</code>	énumération d'éléments
<code>:</code>	descripteur d'éléments de vecteur/matrice
<code>()</code>	ensemble d'arguments
<code>,</code>	séparateur d'arguments
<code>;</code>	séparateur des lignes dans les matrices supression du résultat de l'évaluation d'une instruction
<code>'</code>	transposition de matrice
<code>.</code>	force l'opérateur à s'appliquer sur chaque élément du vecteur/matrice
<code>%</code>	délimiteur de commentaire
<code>...</code>	continuation de l'instruction sur la ligne suivante

FB/MEHDA

Voir aussi

Interfaces graphiques sous Matlab



<http://blogmatlab.blogspot.com/search/label/ebooks>

Et visiter le site :



<http://blogmatlab.blogspot.com/>