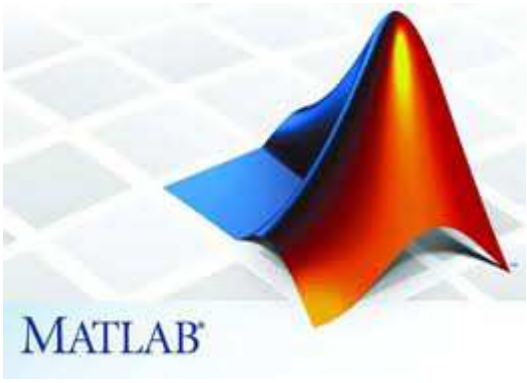
Introduction à MATLAB



Cycle d'ingénieurs et Master

Mohamed ET-TOLBA

Département systèmes de communications

Institut National des Postes et
Télécommunications

ettolba@inpt.ac.ma

Ce document a pour objectif d'aider le lecteur pour apprendre le logiciel MATLAB. Il présente une description de ce logiciel ainsi que des exemples d'illustration pour permettre à l'élève ingénieur d'aborder les travaux pratiques sans difficulté.

Table des matières

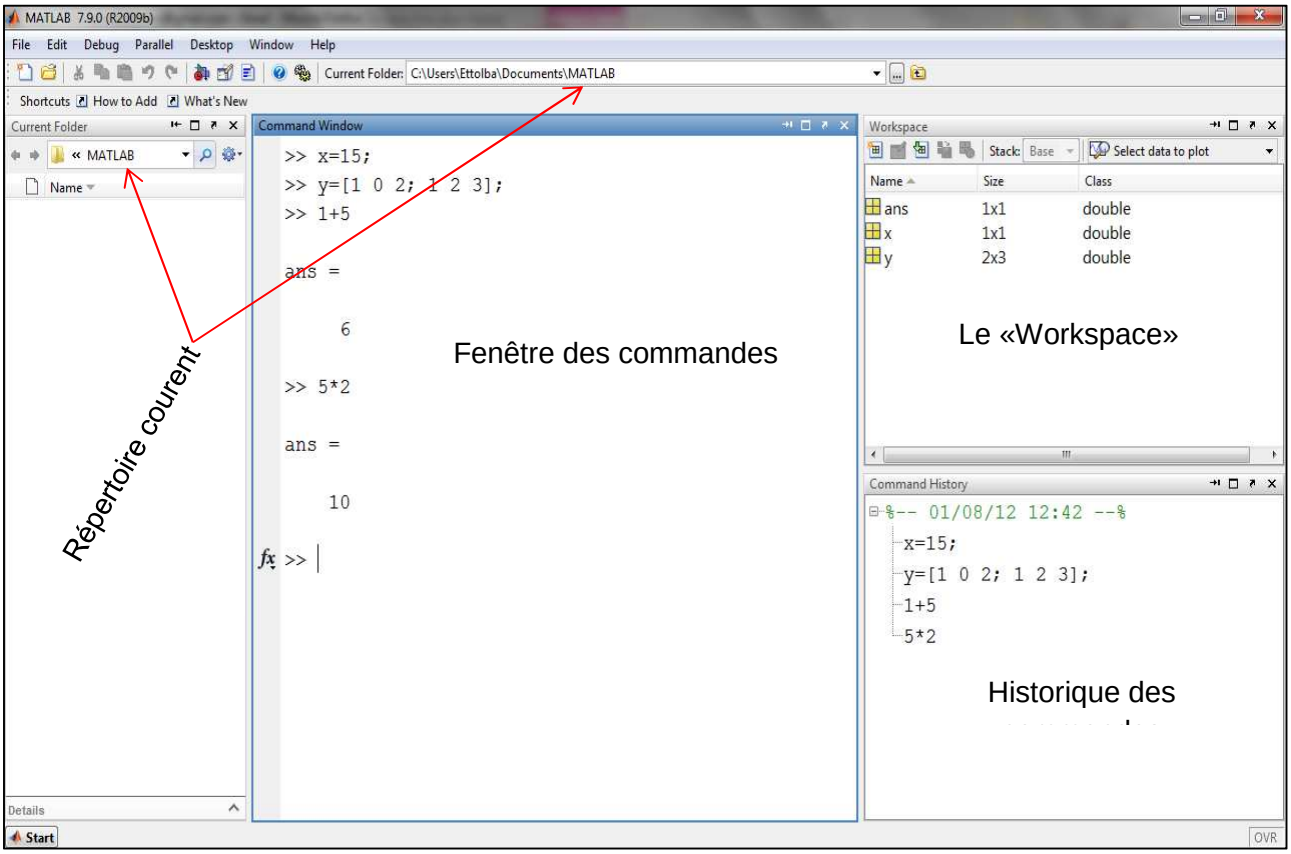
1 Introduction

MATLAB est l'abréviation de *'MATrix LABoratory'*. C'est un logiciel de calcul numérique qui offre un environnement de programmation et de visualisation. Ce logiciel est très puissant pour les calculs matriciels et adapté aux travaux de la recherche scientifique. Il est souvent utilisé dans les disciplines de traitement du signal et d'image, communications, acquisition de données, modélisation et simulation, économie et finance, …etc.

1.1 L'interface graphique de MATLAB

L'utilisateur du logiciel MATLAB dispose d'une interface graphique interactive qui facilite l'utilisation de ce logiciel. L'élève ingénieur doit se familiariser avec quatre fenêtres dans l'interface MATLAB :

- La fenêtre des commandes : elle permet de saisir les commandes et visualiser les résultats
- La fenêtre d'historique des commandes : elle affiche l'historique des commandes saisies.
- La fenêtre '*workspace*' : pour voir l'ensemble des variables crées lors de la session MATLAB.
- La fenêtre du répertoire courant : elle affiche le contenu du répertoire courant.



1.2 L'aide MATLAB

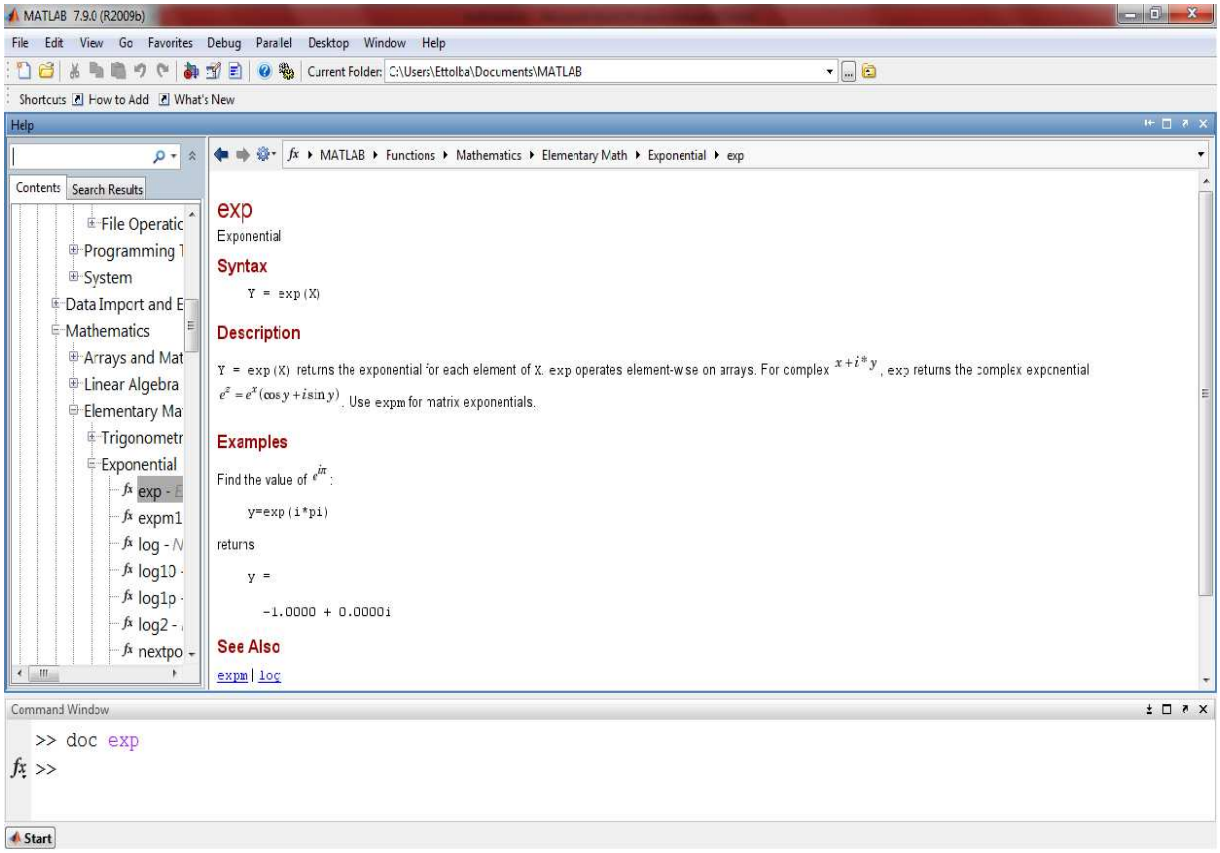
L'aide MATLAB consiste à présenter le fonctionnement d'une commande ou d'une fonction à l'utilisateur. Pour l'obtenir, il suffit de taper $help$, suivie du nom de la commande (ou de la fonction) dans la fenêtre des commandes.

Exemple : La fonction $\exp()$

```
>> help exp

EXP    Exponential.

   EXP(X) is the exponential of the elements of X, e to the X.

   For complex Z=X+i*Y, EXP(Z) = EXP(X)*(COS(Y)+i*SIN(Y)).


   See also expm1, log, log10, expm, expint.
```

En bas de l'aide, on peut facilement avoir accès aux fonctionnements des fonctions associées à $\exp()$.

Pour avoir de l'aide avec une jolie présentation munie d'exemples et facile à lire, on tape doc suivie du nom de la commande (ou de la fonction) dans la fenêtre des commandes.



2 Les variables

MATLAB ne nécessite pas la déclaration et l'initialisation des variables avant leur utilisation.

2.1 Les types de variables supportés par MATLAB

MATLAB est un langage qui supporte plusieurs types de variables. Les plus souvent utilisées parmi ces types de variables sont :

- **double** : ce type est réservé pour les variables numériques. Il occupe une zone mémoire de **64 bits**. C'est le type de variable utilisé dans MALAB par défaut.

```
>>  a= 3.14

 a =

        3.14

>> 3.14

ans

        3.14
```

- **char** : ce type de variable est utilisé pour les caractères. Son occupation de mémoire est de **16 bits.**

```
>>  'm'

ans =

         m

>> c='m'

c =

        m
```

Pour stocker les données en mémoire, MATLAB utilise souvent des variables sous-forme de **vecteurs** ou de **matrices** de type **double** ou **char.**

D'autres types de variables sont supportés par MATLAB : complex, logical, symbolic, … etc

2.2 Création de variables

Pour créer une variable dans MATLAB et lui affecter une valeur on tape l'instruction suivante :

[Non de la variable]=[Valeur de la variable] ;

N.B : les crochets ne font pas partie de la syntaxe de l'instruction.

Exemple : création d'une variable x qui reçoit la valeur 10 et une autre variable MaChaine qui reçoit la chaine de caractère 'mat9ich bladi'

```
>>  x = 10 ;

>> MaChaine= 'mat9ich bladi' ;
```

- ***Le choix des noms de variables***

  - Le premier caractère du nom d'une variable doit être une lettre
  - A partir du deuxième caractère du nom on peut combiner lettres, nombres et _
  - Les noms var1 et Var1 sont différents

- ***Les noms de variables à éviter***
  - **i** et **j** : MATLAB peut les prendre comme des nombres complexes

```
>>  i

 ans=

          0+     1i

>>  j

 ans=

          0+     1i
```

  - **pi** : c'est la valeur 3.141592…

```
>>  pi

ans=

          3.14159265358979
```

  - **ans** : reçoit la dernière valeur non assignée à une variable
  - **Inf** et **–Inf** : sont 'plus l'infini' et 'moins l'infini'
  - **NaN** : représente une quantité qui n'est pas un nombre

2.2.1 Les variables scalaires

Une variable scalaire, dans MATLAB, peut recevoir une valeur

- d'une manière explicite

```
>>  x=10
```

- ou sous forme d'une fonction de valeurs explicites et de variables crées précédemment,

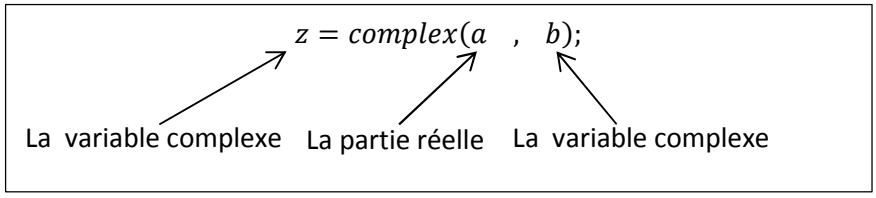
```
>>  y=2*pi - 4*a
```

- Si on ne veut pas afficher la sortie (le résultat), on termine la ligne de commande par un point-virgule comme indiqué auparavant.

```
>>  y=2*pi - 4*a ;
```

2.2.2 Les variables de type complexe

Pour créer une variable complexe dans MATLAB on peut utiliser soit la fonction $complex()$ ou taper directement la variable. Voici la syntaxe de la fonction $complex()$:



Pour créer la variable z, d'une manière directe, on tape l'instruction suivante : $z = a + i * b$;

Exemple :

```
>> z=complex(3 ,  0.5)

z =

          3 +                0.5i

>> z=3+0.5*i

z =

          3 +                0.5i
```

L'utilisateur MATLAB dispose des fonctions qui permettent d'accéder aux parties (réel et imaginaire) et l'argument du nombre complexe :

- La fonction $real()$: retourne la partie réelle d'un nombre complexe
- La fonction $imag()$: retourne la partie imaginaire d'un nombre complexe

- La fonction $angle()$: retourne l'argument d'un nombre complexe

On peut calculer le conjugué et le module d'un nombre complexe en utilisant les deux fonctions $conj()$ et $abs()$ respectivement.

2.2.3 Les vecteurs

2.2.3.1 Les vecteurs lignes

Pour créer une variable vecteur, dans MATLAB, il faut lui donner un nom et mettre ses éléments entre crochets. Les éléments du vecteur sont séparés par une **virgule** ou le caractère **espace**.

```
>>  ligne=[1 2 3.2 -5.6 4] ;

>>  ligne=[1, 2, 3.2, -5.6, 4] ;
```

2.2.3.2 Les vecteurs colonnes

Dans un vecteur colonne, les éléments sont séparés par un **point-virgule** et placés entre crochets.

```
>>  colonne=[1; 2; 3.2; -5.6; 4] ;
```

Un vecteur colonne peut être construit à partir d'un vecteur ligne en suivant les instructions ci-après

```
>> ligne=[1 2 3 4 5 6]

ligne =

    1    2    3    4    5    6

>> colonne=ligne'

colonne =

    1

    2

    3

    4

    5

    6
```

2.2.3.3 Les fonctions size() et length()

La distinction entre un vecteur ligne et un vecteur colonne se fait par :

- L'affichage du vecteur au niveau de la fenêtre des commandes
- Visualisation du « workspace »
- Utilisation de la fonction MATLAB **size()**

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

>> v=u';

>> size(u)

ans =

    1     6

>> size(v)

ans =

    6     1
```

Pour avoir la taille d'un vecteur (ligne ou colonne), on utilise la fonction MATLAB **length()**

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

>> length(u)

ans =

        6
```

```
>> v=u';

>> length(v)

ans =

        6
```

2.2.4 Les matrices

2.2.4.1 Création d'une matrice par l'opérateur de construction, []

La méthode la plus simple pour créer une matrice dans MATLAB est celle qui utilise l'opérateur de construction des matrices, []. Dans cette méthode on saisit les éléments (séparés par des virgules ou espace) de la première ligne, puis pour créer une nouvelle ligne, on termine la ligne courante par un point-virgule et ainsi de suite :

$$M = [ligne_1; ligne_2; \dots; ligne_n]$$

Exemple : Création d'une matrice de **3 lignes** et de **6 colonnes**

```
>> M= [1 2 3 4 5 6; 7 8 9 10 11 12; 13 14 15 16 17 18]

M =

    1    2    3    4    5    6
    7    8    9   10   11   12
   13   14   15   16   17   18
```

2.2.4.2 Création d'une matrice par fusion

Une matrice peut être construite par fusion d'une autre matrice et d'un vecteur. On distingue deux manières pour fusionner une matrice et un vecteur : fusion horizontale et fusion verticale.

Fusion horizontal

La fusion horizontale d'une matrice M et d'un vecteur u pour construire la matrice N s'exprime par :

$$N = [M, u] \text{ ou } N = [M\ u]$$

Exemple : Fusion horizontale de la matrice M et le vecteur u définis par

$$M = \begin{pmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \\ 0.5 & 0.6 \end{pmatrix} \text{ et } u = \begin{pmatrix} 1 \\ -1 \\ 2.3 \end{pmatrix}$$

```
>> M= [0.1  0.2;  0.3  0.4;  0.5  0.6]

M =

    0.1000    0.2000
    0.3000    0.4000
    0.5000    0.6000

>> u= [1;  -1;  2.3]

u =

    1.0000
   -1.0000
    2.3000
```

```
>> N= [M,  u]

N =

    0.1000    0.2000    1.0000
    0.3000    0.4000   -1.0000
    0.5000    0.6000    2.3000
```

Fusion vertical

Une matrice P peut-être crée par fusion verticale d'une autre matrice M et d'un vecteur v. Cette fusion s'écrit :

$$P = [M, v]$$

Exemple : Fusion de la matrice de l'exemple précédent et le vecteur $v = (-0.54 \quad 3.14)$

```
>> M= [0.1  0.2;  0.3  0.4;  0.5  0.6]

M =

    0.1000    0.2000
    0.3000    0.4000
    0.5000    0.6000

>> v=[-0.54  3.14]

v =

   -0.5400    3.1400

>> P=[M; v]

P =

    0.1000    0.2000
    0.3000    0.4000
    0.5000    0.6000
   -0.5400    3.1400
```

3 Opérations sur les variables et fonctions usuelles

3.1 Opérations sur les scalaires

Pour présenter les opérations usuelles sur les variables scalaires, nous allons créer deux variables scalaires x et y auxquels nous assignons deux valeurs numériques.

```
>> x = 7;

>> y = 3;
```

- Somme de deux variables :

```
>> s = x + y
```

- Différence :

```
>> d = x - y
```

- Produit :

```
>> p = x * y
```

- Division :

```
>> e = x/y
```

- Puissance :

```
>> e = x^y
```

3.2 Opérations sur les vecteurs

- Somme et différence de deux vecteurs :

La somme et la différence de deux vecteurs sont des opérations qui s'effectuent élément par élément. Les deux vecteurs doivent être de même dimension. On ne peut pas faire la somme ou la différence d'un vecteur colonne et d'un vecteur ligne.

```
>> u=[1 2 3 4 5]
u =
   1   2   3   4   5
>> v=[1 1 2 1 2]
v =
   1   1   2   1   2
```

```
>> u+v
ans =
   2   3   5   5   7
>> u-v
ans =
   0   1   1   3   3
```

- Multiplication élément par élément (ou point par point)

Cette opération impose que les tailles des deux vecteurs soient compatibles. Voici la multiplication élément par élément des deux vecteurs u et v de l'exemple précédent

```
>> u.*v
ans =
   1   2   6   4   10
```

- Produit scalaire : c'est le résultat du produit d'un vecteur ligne et un vecteur colonne de même taille. Pour donner un exemple on considère le vecteur colonne w construit à partir du vecteur v :

```
>> w=v'

w =

    1

    1

    2

    1

    2
```

```
>> u*w


ans =


    23
```

- Produit d'un vecteur ligne par un vecteur colonne : le résultat de ce produit est une matrice carrée.

  Les deux vecteurs doivent être de même taille.

```
>> w*u

ans =

    1    2    3    4    5

    1    2    3    4    5

    2    4    6    8   10

    1    2    3    4    5

    2    4    6    8   10
```

3.3 Opérations sur les matrices

Les opérations usuelles sur les matrices doivent respecter la compatibilité entre les dimensions.

Considérons deux matrices A et B. Les opérations usuelles sur les matrices s'expriment :

- Somme de deux matrices : $M1 = A + B$

  Les matrices A et B doivent avoir les mêmes dimensions

- Différence de deux matrices : $M2 = A - B$
  Les matrices A et B doivent avoir les mêmes dimensions

- Produit de deux matrices : $M3 = A * B$

Ce produit impose que le nombre de colonnes de la matrice A doit être égal au nombre de lignes de la matrice B.

- Produit élément par élément de deux matrices : $M4 = A.* B$
  Les matrices A et B doivent être de même dimension.
- Transposé conjugué d'une matrice : $M7 = A'$
- Transposé réel d'une matrice : $M8 = A.'$
  Dans cette opération, les éléments de A ne sont pas transformés en leurs conjugués.

```
>> A=[i 1; 1+i 2];

>> A'

ans =

                0 -             1i              1 -             1i

                1                               2

>> A.'

ans =

                0 +             1i              1 +             1i

                1                               2
```

- Puissance d'une matrice : $M5 = A\hat{}n$
  La matrice A doit être carrée.
- Puissance élément par élément : $M6 = A.\hat{}n$
- Somme d'une matrice et un scalaire : $N1 = A + \alpha$

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

A =                                ans =

   1    2    3                          4    5    6

   4    5    6                          7    8    9
```

- Différence d'une matrice et un scalaire : $N2 = A - \alpha$

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

```
>> A-3
 ans =
      -2   -1    0
       1    2    3
```

- Multiplication d'une matrice par un scalaire : $N3 = \alpha * A = \alpha .* A$

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

```
>> 0.5*A
ans =
       0.5     1     1.5
         2     2.5     3
```

3.4 Les fonctions usuelles dans MATLAB

MATLAB possède une grande bibliothèque de fonctions préprogrammées qui sont exploitées par un simple appel en passant les paramètres entre parenthèses. Voici quelques exemples de fonctions usuelles dans MATLAB :

```
>> sqrt(2)
>> log(2)
>> atan(-0.7)
>> floor(4.3)
```

```
>> log10(2.3)
>> cos(1.2)
>> exp(1+3*i)
>>ceil(5.28)
```

3.5 Initialisation automatique des variables

- La fonction **zeros()** : cette fonction est utilisée pour initialiser un vecteur ou une matrice par des zéros.
  L'initialisation par des zéros d'une matrice $\boldsymbol{M}$ de L lignes et C colonnes se fait selon la syntaxe suivante :

M=**zeros**(L,C) ;

```
>> M=zeros(2,3)
M =
     0    0    0
     0    0    0
```

Pour initialiser un vecteur ligne par des zéros il suffit de prendre $L = 1$

ligne=**zeros**(1,C) ;

```
>> ligne=zeros(1,4)

ligne =

     0     0     0     0
```

Un vecteur colonne est initialisé par des zéros en prenant $C = 1$

colonne=**zeros**(L,1) ;

```
>> colonne=zeros(3,1)

colonne =

     0

     0

     0
```

- La fonction **ones()** : permet d'initialiser un vecteur ou une matrice par des uns. Sa syntaxe est la même que celle de la fonction zeros.

  Exemple :

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

M =

     1     1

     1     1

     1     1
```

```
>> colonne=ones(3,1)

colonne =

     1

     1

     1
```

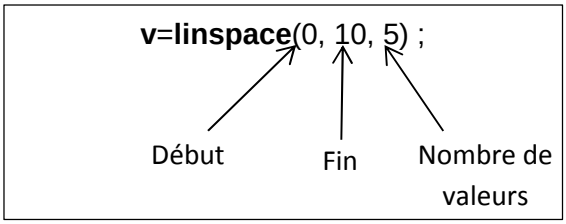
- La fonction **rand() :** initialise un vecteur ou une matrice, d'une manière aléatoire, par des valeurs comprises entre 0 et 1 (distribution uniforme sur $[0, 1]$)

Exemple :

```
>> ligne=rand(1,2)

ligne =

    0.933993247757551    0.678735154857773
```

```
>> M=rand(2,2)

M =

    0.757740130578333    0.392227019534168

    0.743132468124916    0.655477890177557
```

- La fonction **linspace() :** permet d'initialiser (ou de générer) un vecteur avec des valeurs équidistantes. Pour générer un vecteur commençant par 0, finissant par 10 et contenant 5 valeurs, on utilise la syntaxe suivante :



Exemple :

```
>> v=linspace(0, 5, 6)

v =

   0    1    2    3    4    5
```

- Initialisation par l'opérateur (**:**) : On peut initialiser un vecteur dans MATLAB en utilisant l'opérateur (**:**).
  Exemple : t= 0 : 0.5 :10 génère un vecteur t qui commence par 0 et finit par une valeur inférieure ou égale à 10 avec un pas de 0.5. La valeur finale dépend du choix du pas.

```
>> t=0 : 1.5 : 5

t =

     0              1.5              3              4.5

>> t=0 : 2.5 : 5

t =

     0              2.5              5
```

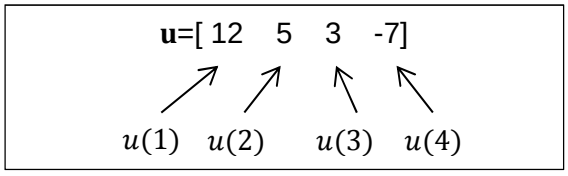
Lorsque la valeur du pas n'est pas spécifiée, le pas est fixé à 1 par défaut.

```
>> t=1:10

t =

   1   2   3   4   5   6   7   8   9   10
```

3.6 Accès aux éléments des variables matrices et vecteurs

Avant de commencer cette partie, il est important de mentionner que les indices dans MATLAB sont soumis à quelques règles :

- Les indices commencent toujours par 1 et non par 0 comme en langage C par exemple.

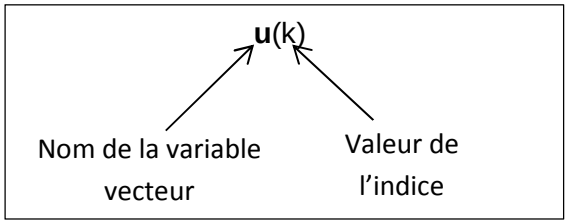


- La valeur de l'indice est entière et positive

3.6.1 Accès aux éléments d'un vecteur

- *Accès individuel*

Pour accéder à $k^{ème}$ élément d'un vecteur, on utilise la syntaxe suivante :



Exemple :

```
>> u=[ 10 5 -2 1];

>> u(1)

ans =

      10
```

```
>> u(3)

ans =

      -2
```

Le dernier élément du vecteur u est accessible soit par la plus grande valeur de l'indice soit par **end :**

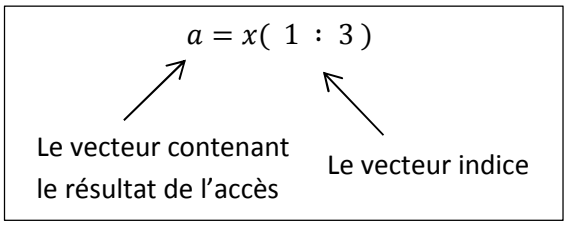
```
>> u(4)

ans =

     1
```

```
>> u(end)

ans =

     1
```

- Accès à un groupe d'éléments successifs

Dans un vecteur on peut accéder à plusieurs éléments successifs. Dans ce cas l'argument indice est un vecteur. Le résultat de cet accès est un vecteur ayant la même taille que le vecteur indice.

Exemple : l'accès à trois éléments successifs du vecteur $x = [1 \ \ -5 \ \ 1.5 \ \ 0.1 \ \ 2]$ se fait en utilisant la syntaxe suivante :




```
>> x=[1   -5   1.5   0.1   2];

>> a=x(1:3)

a =

    1          -5          1.5
```

```
>> b=x(2:4)

b =

    -5          1.5          0.1
```

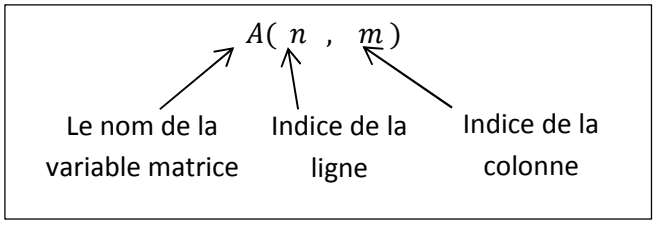
3.6.2 *Accès aux éléments d'une matrice*

- Accès individuel

Les éléments d'une matrice sont accessibles de deux manières :

- ✓ En utilisant les indices des lignes et des colonnes

  Pour accéder à l'élément situé à la $n^{ème}$ ligne et $m^{ème}$ colonne d'une matrice A, on utilise la syntaxe suivante :



Exemple :

```
>> A=[ 0.5   0.3   1;   2   0.34   0.4] ;

>> A(1,1)

ans =

        0.5

>> A(2,3)

ans =

        0.4
```

- ✓ En considérant la matrice comme un vecteur

Dans ce type d'accès les éléments d'une matrice, A (à deux lignes et trois colonnes par exemple), sont indexés de la manière suivante :

$$A = \begin{bmatrix} A(1) & A(3) & A(5) \\ A(2) & A(4) & A(6) \end{bmatrix}$$

Exemple :

```
>> A=[ 0.5   0.3   1;   2   0.34   0.4] ;

A =

     0.5              0.3              1

       2             0.34             0.4

>> A(4)

ans =

      0.34
```

```
>> A(1)

ans =

        0.5

>> A(2)

ans =

          2
```

- ➢ Accès aux sous-matrices d'une matrice

Pour extraire une **sous-matrice**, N (de p lignes et de q colonnes), d'une matrice M de m lignes et n colonnes, on utilise la syntaxe suivante :

$$N = A(\underbrace{1 : p}_{\text{Parcours des lignes de } A}, \underbrace{1 : q}_{\text{Parcours des colonnes de } A})$$

```
>> M = [10   11   12; 13   14   15]

M =

     10   11   12

     13   14   15
```

```
>> N = M(1  :  2 , 2  :  3)

N =

     11   12

     14   15
```

On peut spécifier les indices des lignes et des colonnes à mettre dans la sous-matrice comme le montre l'exemple suivant :

```
>> M=[10   11   12;  13   14   15] ;

>> a = M([2   1]  ,  [3   2   1])

a =

    15   14   13

    12   11   10
```

- Accès aux vecteurs d'une matrice

MATLAB offre la possibilité d'extraire une ligne ou une colonne d'une matrice.

L'accès à l'ensemble des éléments de la $k^{ème}$ligne d'une matrice A se fait par la syntaxe suivante :

$$ligne = A(k\ ,:)$$

L'ensemble des éléments de la $l^{ème}$ colonne de la matrice A est accessible de la manière suivante :

$$colonne = A(:\ ,l\ )$$

Exemple :

```
>> A = [10   11   12; 13   14   15]
A =
      10    11    12
      13    14    15
```

```
>> ligne=A(2 , : )
ligne =
        13    14    15
>> colonne=A(:,3)
colonne =
              12
              15
```

4 Programmation dans MATLAB

4.1 Les instructions de contrôle

4.1.1 Instructions conditionnelles if, else et elseif

- Structure if

```
if   condition
   Bloc d'instructions
end
```

- Structure if avec else

```
if  condition
    Bloc d'instructions 1
else
   Bloc d'instructions 2
end
```

- Structure elseif

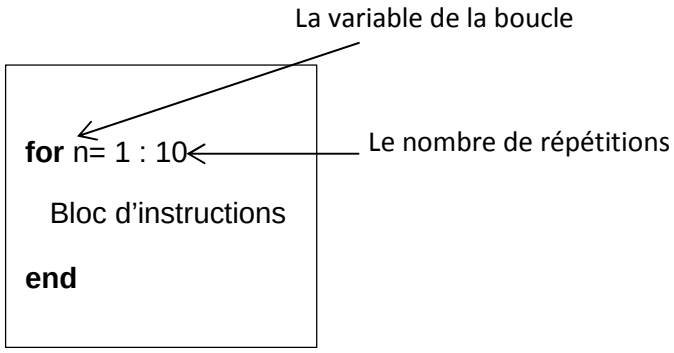
```
if   condition 1
   Bloc d'instructions 1
elseif    condition 2
   Bloc d'instructions 2
Else
   Bloc d'instructions 3
end
```

La condition peut être satisfaite (vraie) ou non (fausse). S'elle est vérifiée, le bloc d'instruction qui en dépend est exécuté, sinon on passe à la suite.

4.1.2 Instructions répétitives for et while

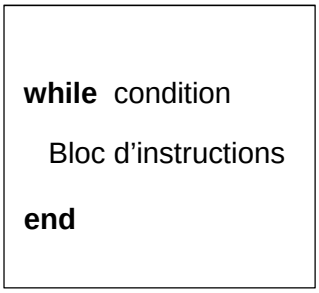
✓ La boucle for

La boucle for est utilisée lorsque le nombre d'itérations (répétitions) est connu à l'avance. Sa syntaxe dans MATLAB est



✓ La boucle while

C'est la structure la plus générale d'une boucle. Dans une boucle while, on n'a pas besoin de connaitre le nombre d'itération. Voici la syntaxe MATLAB pour la boucle while :



Le bloc d'instructions est exécuté tant que la condition est satisfaite. Il faut prendre toutes les précautions afin d'éviter une boucle infinie.

4.1.3 Opérateurs relationnels

La partie condition d'une structure de contrôle (conditionnelle ou répétitive) est un test qui met en jeu des opérandes et des opérateurs relationnels que l'utilisateur doit avoir en tête pour utiliser les instructions de contrôle. Ces opérateurs relationnels sont présentés sur le tableau suivant :

Opérateur	signification
==	Egal
~=	Différent
>	Strictement supérieur
>=	Supérieur ou égal
<	Strictement inférieur
<=	Inférieur ou égal.

4.2 Programmation de fonctions

L'utilisateur a la possibilité de programmer ses propres fonctions et les ajouter à la bibliothèque MATLAB. Ces fonctions peuvent faire appel à des fonctions préprogrammées dans MATLAB comme zeros(), ones(), size(), cos(), exp() …etc.

La programmation d'une nouvelle fonction se fait dans un fichier portant le même nom que la fonction et ayant l'extension '.m'. La définition d'une fonction dans ce fichier doit être précédée par sa déclaration dont la syntaxe est la suivante :

function [Arguments de sorties]=**NomDeLaFonction**(Arguments d'entrée)

- function : c'est un mot clé réservé à la programmation des fonctions. La déclaration d'une nouvelle fonction doit commencer par le mot '**function**'
- Arguments de sorties : ils contiennent les résultats retournés (calculés) par la fonction. Lorsqu'on a plus d'un argument de sortie, ils sont placés entre crochets et séparés par des virgules
- NomDeLaFonction : La fonction doit avoir le même nom que le fichier '.m' dans lequel elle est définie.
- Arguments d'entrée : ils sont déclarés après le nom de la fonction, séparés par des virgules et placés entre parenthèses.

Voici la structure générale de la définition d'une fonction :

function [Arguments de sorties]=**NomDeLaFonction**(Arguments d'entrée)

Le corps de la fonction

On peut marquer la fin d'une fonction par le mot '**end**' mais c'est optionnel.

Exemple 1 : une fonction qui convertit une température de degré Celsius en Fahrenheit et Kelvin :

```
function  [F, K]=TempConversion(C)

    F= C*(9/5) + 32 ;

    K= C + 273.15 ;
```

Exemple 2 : une fonction qui fait le passage des coordonnées polaires aux coordonnées cartésiennes (On considère que les arguments d'entrée, r et theta, sont des vecteurs) :

```
function  [x, y]=PolaireAcartesienne(r,theta)

    x= r.*cos(theta);

    K= r.*sin(theta);
```

Exercice :

Ecrire une fonction qui convertit les coordonnées cartésiennes en coordonnées polaires ?

4.3 Affichage graphique

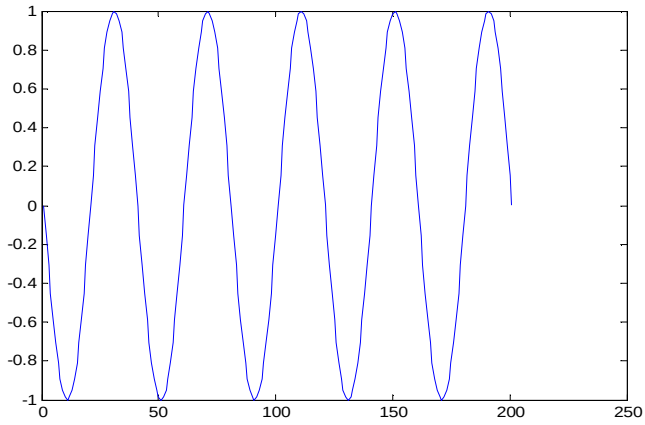
4.3.1 La fonction plot ()

L'affichage graphique à deux dimensions se fait à l'aide de la fonction MATLAB ***plot().***

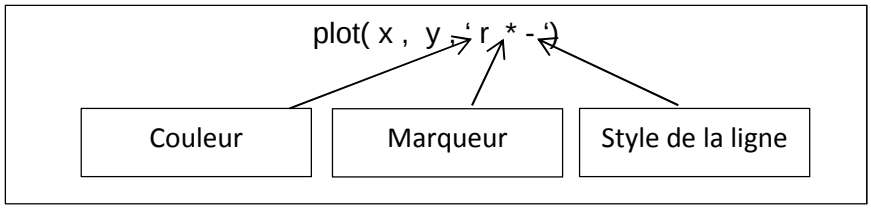
$plot(x, y)$ permet de tracer un vecteur y en fonction d'un autre vecteur . La fonction $plot()$ impose que les deux vecteurs x et y aient la même longueur.

Exemple : tracé du signal sinusoïdal $y = \sin(2\pi f_0 t)$

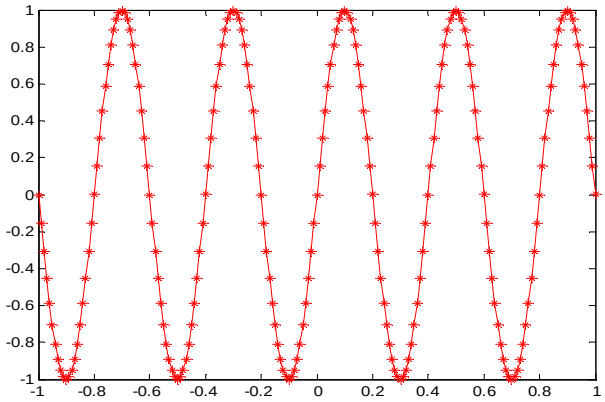
```
>> t=-1 : 0.01 : 1;

>> y=sin(2*pi*2.5*t);

>> plot(y)
```



Des options peuvent être ajoutées à la fonction $plot()$: l'utilisateur a la possibilité de choisir la couleur de la courbe, le type du marqueur des points de la courbe, et le style de la ligne reliant ces points. Voici la syntaxe d'une fonction $plot()$ muni des options



```
>> t=-1 : 0.01 : 1;

>> y=sin(2*pi*2.5*t);

>> plot(t,y,'r * - ')
```



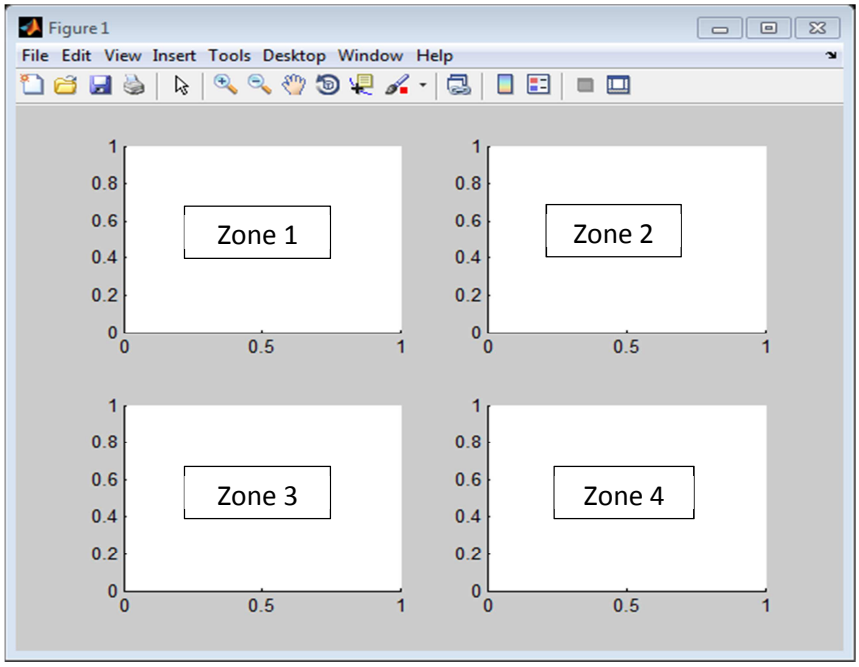
4.3.2 Multiple affichages graphique sur une figure : subplot()

Pour plusieurs affichages graphiques sur une seule figure, on utilise la fonction MATLAB $subplot()$.

$subplot(n\,,m\,,p)$ divise la fenêtre réservée à la figure en $(n \times m)$ zones et en sélectionne la zone numéro p pour l'affichage qui suit la fonction (n, m, p) .

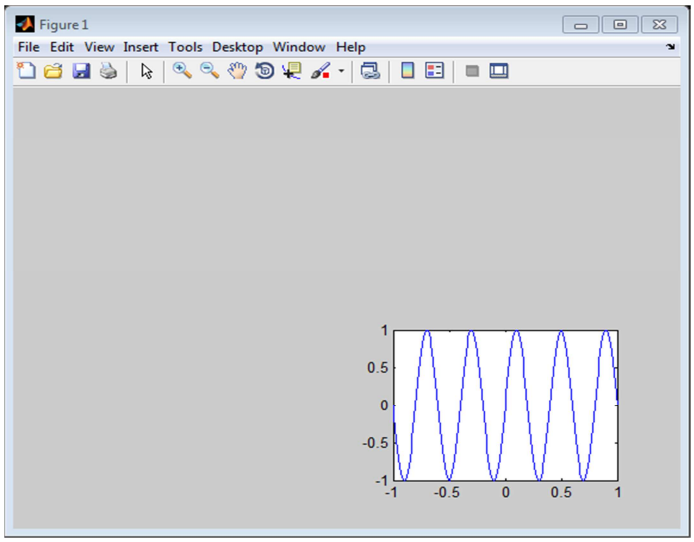
Exemple :

```
>> subplot(2,2,1)

>> subplot(2,2,2)

>> subplot(2,2,3)

>> subplot(2,2,4)
```



Pour afficher le signal sinusoïdal précédent dans la zone 4, on utilise les commandes suivantes :

```
>> t=-1 : .01 : 1;

>> y=sin(2*pi*2.5*t);

>> subplot(2,2,4);

>> plot(t,y)
```



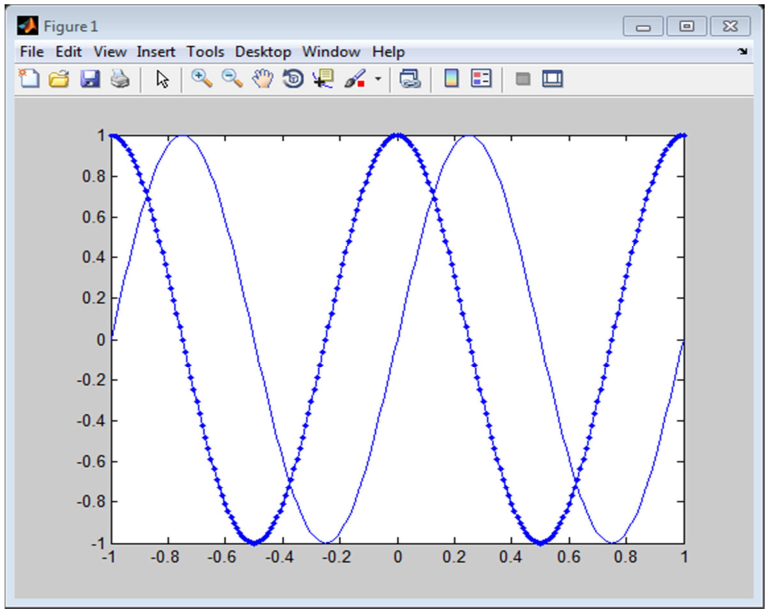
4.3.3 La commande hold et ses variétés hold on et hold off

La commande $hold$ permet de contrôler si MATLAB efface le graphe courant lorsqu'un nouveau graphe sera affiché, ou faire l'affichage des deux graphes à la fois. Voici les variétés les plus courantes de la commande $hold$:

- $hold\ on$: permet de retenir le graphe courant lorsqu'on affiche un nouveau graphe.
- $hold\ off$: annule l'effet de la commande $hold\ on$

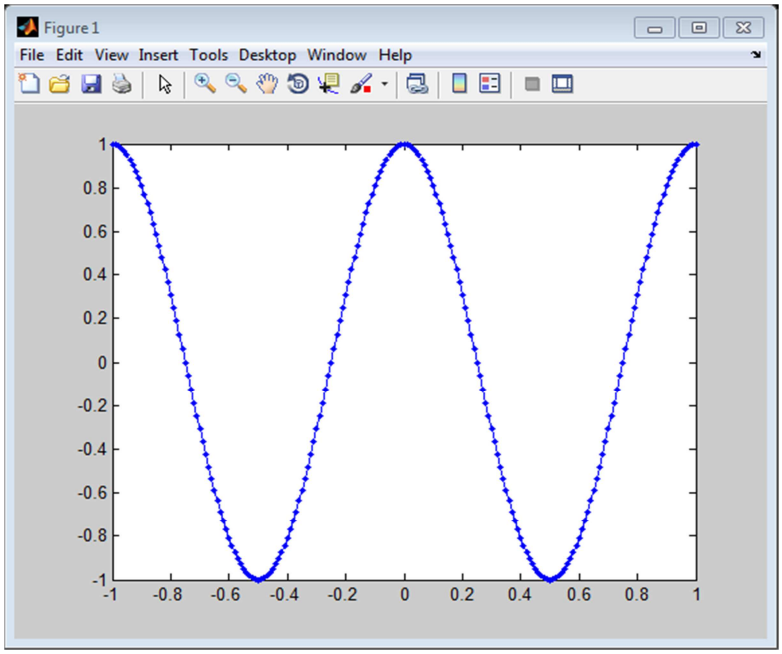
Exemple : l'effet de $hold\ on$ et celui de $hold\ off$

```
>> t=-1 : .01 : 1;

>> y=sin(2*pi*t);

>> y1=sin(2*pi*t);

>> y2=cos(2*pi*t);

>> plot(t,y1);

>> hold on;

>> plot(t,y2,'.-');
```



Pour voir l'effet de $hold\ off$, on ajoute les deux instructions suivantes :

```
>> hold off;

>> plot(t,y2,'.-') ;
```



4.4 Les entrées-sorties dans MATLAB

4.4.1 Entrée au clavier

On a la possibilité de saisir des données au clavier grâce à la fonction MATLAB $input()$. Voici un exemple de saisie d'une valeur numérique au clavier :

```
>> y=input(' Entrer la valeur de y : ');

 Entrer la valeur de y : 15

>> y

y =

    15
```

Pour saisir une chaine de caractères, on procède de la manière suivante :

```
>> Chaine=input('Quel est votre prénom ?','s');

Quel est votre prénom ? Nabil

>> Chaine

Chaine =

          Nabil
```

4.4.2 Affichage à l'écran

L'affichage à l'écran peut se faire soit par la fonction MATLAB $disp()$, $sprintf()$, ou $fprintf()$.

- La fonction $disp()$: permet d'afficher le contenu d'une variable de type chaine de caractère, vecteur, matrice, …etc.

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

>> disp(M);

     1    2    3

     4    5    6
```

- La fonction $fprintf()$: affiche les données à l'écran avec un format spécifié.

```
>> x=10;

>> fprintf('la valeur de x est : %1.3f \n',x);

la valeur de x est : 10.000

>> fprintf('la valeur de x est : %1.2f \n',x);

la valeur de x est : 10.00
```

- La fonction $sprintf()$: utilisée pour l'affichage à l'écran des chaines de caractères. Son fonctionnement est exactement le même que celui de la fonction $fprintf()$.

4.4.3 Lecture et écriture dans un fichier

L'utilisateur de MATLAB peut sauvegarder des données dans un fichier avec la commande $save$ (voir l'aide). Il a également la possibilité d'écrire n'importe quel type de texte. Voici les étapes d'écriture dans un fichier :

- Ouvrir le fichier avec la fonction $fopen()$
- Ecrire dans le fichier à l'aide de la fonction $fprintf()$
- Fermer le fichier par la fonction $fclose()$

Pour plus de détails sur ces fonctions, utiliser l'aide MATLAB.

5 Quelques fonctions élémentaires

5.1 Fonctions élémentaires pour les matrices

Fonction	rôle
real(X);	*partie réelle des éléments de X*
imag(X);	*partie imaginaire des éléments de X*
size(A)	*taille d'une matrice*
A.'	*transposée d'une matrice*
A'	*transposée conjuguée d'une matrice (transposée hermitienne)*
conj(A)	*conjugué d'une matrice*
sqrt(A)	*racine carré des éléments de la matrice*
min(A)	*valeur et position du plus petit élément de A*
max(A)	*valeur et position du plus grand élément de A*
sum(A,1)	*somme des lignes de la matrice*
sum(A,2)	*somme des colonnes de la matrice*
abs(A)	*valeur absolue des éléments de A*
sign(A)	*prend le signe des éléments de A*
rem(x,y)	*reste après la division de x par y*
fix(x)	*rend la partie entière la plus proche de zéro*
mod(x,y)	*fonction x modulo y*
floor(A)	*arrondi vers l'entier inférieur*
ceil(A)	*arrondi vers l'entier supérieur*
round(A)	*arrondi vers l'entier le plus proche*
fliplr(A)	*permutation de gauche à droite des éléments d'une matrice*
flipud(A)	*permutation de haut en bas des éléments d'une matrice*
find(A<0.5)	*recherche des éléments de A inférieurs à 0.5*
sort(X)	*tri des éléments de X par ordre croissant*
mean(X)	*valeur moyenne sur X*
std(X)	*écart type sur X*
prod(X)	*produit des éléments de X*
cumsum(X)	*somme cumulée des éléments de X*
eig(A)	*valeurs propres de la matrice A*

Il existe d'autres fonctions mathématiques élémentaires pour les matrices : **sin, cos, tan, cot, sinc, sinh, cosh, tanh, coth, exp, log, log10, log2, erfc, ...**

5.2 Fonctions élémentaires pour les représentations graphiques et les figures

Fonction	rôle
figure(n)	*ouvre ou sélectionne la fenêtre n pour affichage d'une figure*
plot(X,'x')	*trace le vecteur X avec un x pour chaque point*
hist(X)	*tracé d'un histogramme des valeurs de X*
semilogy(X)	*trace un graphe en semilog (utile pour les probabilités d'erreurs)*
clf	*efface la figure courante*
close	*fermeture de la fenêtre courante*
close all	*fermeture de l'ensemble des fenêtres associées aux figures*
subplot( )	*permet de diviser une fenêtre en plusieurs figures*
axis([min_x max_x min_y max_y])	*permet de redéfinir la fenêtre de visualisation*
hold	*maintien d'une figure pour superposition d'une autre figure*
grid	*positionnement d'une grille*
xlabel('texte')	*permet de labelliser l'axe des abscisses*
ylabel('texte')	*permet de labelliser l'axe des ordonnées*
title('texte')	*permet de donner un titre à la figure*

5.3 Autres fonctions élémentaires

Fonction	rôle
rand	*variables uniformement distribuées entre 0 et 1*
randn	*variables distribuées selon une loi normale N(0,1)*
fft	*transformée de Fourier rapide*
ifft	*transformée de Fourier inverse rapide*
conv	*convolution de deux vecteurs*
filter	*filtrage numérique avec traitement des effets de bords*
freqz	*trace la fonction de transfert d'un filtre numérique*
psd	*estimation de la densité spectrale de puissance*
roots	*racines d'un polynome*
abs	*module*
angle	*phase*