Chapitre III: Programmer avec Matlab

1. Présentation de Matlab

MATLAB = « MATrix LABoratory »
(Laboratoire des Matrices)

Matlab est un logiciel pour le calcul scientifique, orienté vers les vecteurs et les listes de données. Matlab est un langage interprété, chaque ligne d'un programme Matlab est lue, interprétée et exécutée. Procurant ainsi un environnement flexible pour le calcul technique.

Matlab se présente avec un environnement interactif et un prompt ou invite de commandes (généralement >>) dans lequel on peut introduire des commandes. Par exemple, pour sortir de Matlab tapez la commande:

```
>> quit
```

Matlab est basé sur le calcul matriciel numérique. Tous les objets utilisés dans Matlab sont donc définis au travers de matrices ou vecteurs dont les valeurs sont, par définition, des grandeurs complexes. Il existe un très grand nombre d'opérateurs et fonctions distribués dans le logiciel de base et dans des boîtes à outils spécialisées. A ceci peut s'ajouter un outil de programmation graphique, **Simulink**, essentiel pour la simulation de systèmes dynamiques non linéaires.

L'environnement Matlab se présente sous la forme d'un espace de travail dans lequel un interpréteur de commandes exécute les opérations demandées.

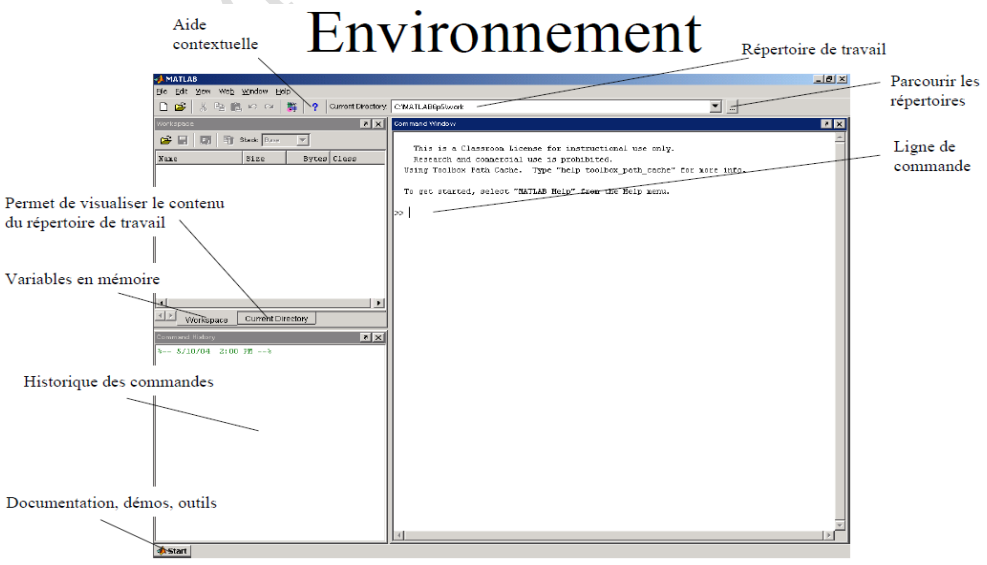

Figure 1- Environnement de Matlab.

L'affectation des valeurs à des variables et l'effectuation des opérations sur ces variables ce fait comme suit:

```
>> x = 4
x =
        4

>> y = 2
y =
        2

>> x + y
ans =
        6

>> x * y
ans =
        8
>>
```

Remarque

Lorsqu'on ne fixe pas un variable de sortie, Matlab place le résultat d'aune opération dans "**ans**".

Pour connaitre les variables utilisées et leur type, on utilise la commande "**whos**".

Exemple

Pour les manipulations suivantes:

```
>> whos
Name          Size                Bytes  Class
ans           1x1                     8  double array
x             1x1                     8  double array
y             1x1                     8  double array
Grand total is 3 elements using 24 bytes
>>
```

La solution de l'opération "x + y" n'est pas affichée (variable perdue). Il est donc préférable de toujours donner des noms aux variables de sortie. Si on ne veut pas afficher la valeur d'une variable en ajoute un point-virgule (**;**):

```
>> x = 4;
>> y = 2;
>> a = x + y
a =
        6
```

```
>> b = x * y
b =
        8
>> whos
Name          Size                    Bytes  Class
a             1x1                         8  double array
b             1x1                         8  double array
x             1x1                         8  double array
y             1x1                         8  double array
Grand total is 4 elements using 32 bytes
>>
```

Pour écrire le résultat d'un programme dans une phrase, on peut convertir les nombres en chaine de caractères en utilisant la commande "**num2str**". Les commentaires sont écrits dans un programme en utilisant le signe "**%**" au début de la ligne du commentaire.

Exemple

Programme de calcul de la racine carrée d'un nombre:

```
a = input ('Entrez un nombre:  '); % utilisation de input, l'utilisateur
                                   % doit entrer un nombre.
b = sqrt(a);

str = ['La racine carrée de ' num2str(a) ' est ' num2str(b)];
                                   % composition de la phrase de sortie
disp(str)                  % utilisation de display pour afficher le résultat à l'écran
```

Le résultat serait le suivant:

```
Entrez un nombre: 23          % 23 est entrée par l'utilisateur
La racine carrée de 23 est 4.7958       % sortie à l'écran
```

2. Operations mathématiques avec Matlab

Scalaires, vecteurs, matrices

L'élément de base de Matlab est la matrice. C'est-à-dire:

- un scalaire est une matrice de dimension **1x1**;
- un vecteur colonne de dimension **n** est une matrice **nx1**;
- un vecteur ligne de dimension **n** est une matrice **1xn**.

Les **scalaires** sont déclarés directement, par exemple:

```
>> x = 0;
>> a = x;
```

Les **vecteurs lignes** sont déclarés comme suit:

```
>> V_ligne = [0 1 2]
V_ligne =
        0        1        2
```

Pour les **vecteurs colonnes**, on sépare les éléments par des points-virgules:

```
>> V_colonne = [0;1;2]
V_colonne =
        0
        1
        2
```

- ***Le transposé d'un vecteur:*** se fait à l'aide de la commande **transpose** ou avec l'apostrophe ('):

```
>> V_colonne = transpose (V_ligne)
V_colonne =
        0
        1
        2

>> V_colonne = V_ligne'
V_colonne =
        0
        1
        2
```

- ***L'incrémentation***: se fait à l'aide de double point (**:**). Pour créer par exemple un vecteur ligne des valeurs de 0 à 1 par incrément 0.2:

```
>> V = [0:0.2:1]
V =
    Columne 1 through 6
        0        0.2000        0.4000        0.6000        0.8000        1.0000
```

Par défaut l'incrémentation est de 1. Par exemple: pour un vecteur ligne de 0 à 5 par incrémentation 1:

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

- ***Les opérations sur les vecteurs:*** l'addition, la soustraction, la multiplication par scalaire:

```
>> V1 = [1 2]
>> V2 = [3 4]
>> V = V1 + V2        % addition de vecteurs
V =
        4        6

>> V2 – V1            % soustraction de vecteurs
V=
        2        2

>> V = 2*V1           % multiplication par un scalaire
V =
        2        4
```

- ***La multiplication et la division de deux vecteurs:*** ici il faut faire attention aux dimensions des vecteurs en cause. Pour la multiplication et la division élément par élément, on ajoute un point devant l'opérateur (**.*** et **./**). Par exemple:

```
>> V = V1.*V2         % multiplication élément par élément
V =
        3        8

>> V = V1./V2         % division élément par élément
V =
        0.3333        0.5000
```

Si les dimensions des vecteurs ne concordent pas, Matlab lancera un message d'erreur. Les messages d'erreur peuvent être très utiles pour corriger les programmes:

```
>> V = [1 2 3]
V =
        1        2        3
>> V = V1.*V3
??? Error using ==> .*  Matrix dimensions must agree.
```

La multiplication de deux vecteur est donnée par (*). Ici l'ordre a de l'importance:

```
>> V1= [1 2];         % vecteur 1x2
>> V2 = V1' ;         % vecteur 2x1
>> V = V1*V2
V =
        5
```

```
>> V = V2*V1
V =
     1     2
     2     4
```

- ***Concaténation des vecteurs (****mettre bout à bout****):***

Cas des vecteurs lignes:

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

Cas des vecteurs colonnes:

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

Création de matrices à partir de vecteurs:

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

qui n'est pas équivalant à:

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

- ***Les matrices:*** peuvent être construites directement comme suit:

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

- ***Accès à un élément d'une matrice:***

```
>> m21 = M(2,1)        % 2ème ligne, 1ère colonne
m21 =
        3
```

On peut aussi compter les éléments d'une ligne (de gauche à droite) avant d'accéder à la ligne suivante:

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

>> a5 = A(5)            % la valeur du 5ème élément de la matrice A
a5 =
        5
```

- ***Stockage d'une ligne d'une matrice dans un vecteur:*** si on veut stocker la 2ème colonne de la matrice A:

```
>> V = A(: , 2)         % ici, (:) signifie toute les lignes
V =
        2
        5
        8
```

- ***Stockage de plusieurs lignes d'une matrice:*** si on veut stocker les lignes 2 et 3 de la matrice A:

```
>> M2 = A(2:3, :)       % (2:3) signifie lignes 2 à 3
                        % et (:) signifie toutes les colonnes
M2 =
        4       5       6
        7       8       9
```

- ***Inverse et transposé d'une matrice:*** à l'aide des commandes "**inv**" et "**transpose** ou (')" respectivement:

```
>> invM = inv (M)
invM =
        -2.0000         1.0000
         1.5000        -0.5000

>> transpM = M'
transpM =
        1       3
        2       4
```

- ***Operations sur les matrices:*** Addition, Soustraction Multiplication et Division (attention aux dimensions):

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

>> C = A + B          %       addition
C =
        5       5
        5       5

>> D = A – B          % soustraction
D =
       -3      -1
        1       3

>> C = 3 * A          % multiplication par un scalaire
C =
        3        6
        9       12

>> C = A * B          % multiplication matricielle
C =
        8        5
       20       13

>> D = A/B            % division matricielle
D =
        1.5000         -2.5000
        2.5000         -3.5000


>> C = A .*B          % multiplication élément par élément
C =
        4       6
        6       4

>> D = A ./B          % division élément par élément
D =
        0.2500         0.6667
        1.5000         4.0000
```

Les matrices spéciales

```
>> I = eye(3)            % matrice identité
I =
     1     0     0
     0     1     0
     0     0     1

>> V_nul = zeros(1,2)    % un vecteur de 1 ligne, 2 colonnes de 0
V_nul =
     0     0

>> M_nul = zeros(2,2)    % une matrice 2x2 de 0
M_nul =
     0     0
     0     0

>> V_un = ones(1,2)      % un vecteur de 1 ligne, 2 colonnes de 1
V_un =
     1     1

>> M_un = ones (2,2)     % une matrice 2x2 de 1
M_un =
     1     1
     1     1
```

Dimension d'une matrice et longueur d'un vecteur

Par les commandes **size** et **length** respectivement:

```
>> V = [0:0.1:10];      % utilisation de length – vecteur 1x101
>> n = length(V)
n =
     101

>> M = [1 2 3; 4 5 6; 7 8 9];          % utilisation de size – matrice 2x3
>> [n,m] = size(M)
n =
     2
m =
     3

>> dim = length (M)          % utilisation de length sur une matrice
dim =
     3      % donne la plus grande dimension, ici nombre de colonnes
```

3. Les types de données en Matlab

Le type courant

Le courant des variables en Matlab est le type "réel", il n'y a pas de type "entier" proprement dit. Ce type est affecté automatiquement à partir des valeurs affectées à la variable.

Le type complexe

L'unité imaginaire est désignée par "***i***" ou "***j***". Les nombres complexes peuvent être écrits sous la forme cartésienne "***a + ib***" ou sous forme plaire "re^{it}".

Les différentes écritures possibles sont:

```
a+ib
a+i*b
a+bi
a+b*i
r*exp(it)
r*exp(i*t).
```

Exemple

```
>> z = [1+i, 2 , 3i]
z =
        1.0000 + 10000i        2.0000        0 + 3.0000 i
```

Les commandes ***real*** et ***imag*** permettent d'afficher la partie réelle ou imaginaire d'un nombre complexe respectivement.

Le type chaîne de caractères

Une donnée de type chaîne de caractère (***char***) est représentée sou la forme d'une suite de caractères encadrée d'apostrophes simples ('). La concaténation de chaîne de caractères s'effectue de la même façon pour les règles de manipulation des tableaux.

```
>> ch1 = 'bon'
ch1 =
        bon
>> ch2 = 'jour'
ch2 =
        jour

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

```
>> ch = [ch1; ch2]
ch =
        bonjour
```

4. Operateurs logiques

Ce type d'operateurs permet de comparer des valeurs entre elles, les résultats prennent les valeurs 1 (vrais) et 0 (faux).

Tableau 1- Operateurs de comparaisons

Opérateur	Description
~ a	NOT - Retourne 1 si a égal 0, et 1 si a égal 0
a == b	Retourne 1 si a égal b, 0 autrement
a < b	Retourne 1 si a est inferieur à b, 0 autrement
a > b	Retourne 1 si a est supérieur à b, 0 autrement
a <= b	Retourne 1 si a est inferieur ou égal à b, 0 autrement
a >= b	Retourne 1 si a est supérieur ou égal à b, 0 autrement
a ~= b	Retourne 1 si a est différent de b, 0 autrement

Exemple

```
>> a = sin(2*pi);
>> b= cos(2*pi);
>> bool = (a>b)
bool =
        0

>>a
a =
        -2.4493e-016            % ici a devrait égaler 0, la précision est limitée!
>>b
b =
        1
```

Remarque

L'emploi de l'opérateur '==' est très risqué lorsqu'on compare deux valeurs numériques, du fait que la précision de l'ordinateur est limitée. Il est préférable donc d'utiliser une condition sur la différence comme dans l'exemple suivant:

```
if abs(a-b) < eps       % eps est la précision machine (2.2204e-016)
        bool = 1;
else
        bool = 0;
end
```

Il est aussi possible de lier entre les conditions par les opérateurs "et" (**&**) et "ou" (|).

Tableau 2 – Table de vérité – opérateur ET

P	Q	P et Q
1	1	1
1	0	0
0	1	0
0	0	0

Tableau 3 – Table de vérité – opérateur OU

P	Q	P ou Q
1	1	1
1	0	1
0	1	1
0	0	0

5. Les structures répétitives et conditionnelles

Les structures *if – elseif – else*

Ce type de structures de programmation est très utile pour vérifier des conditions. Elle est présentée en Matlab de la manière suivante:

```
if CONDITON1
        ACTION1;
elseif CONDITION2
        ACTION2;
else
        ACTION3;
```

La boucle *for*

Cette boucle est très utile dans la plupart des applications mathématique). La syntaxe de cette boucle en Matlab est comme suit:

```
for i= 0:valeur_finale
        ACTION1;
        ACTION2;
        ACTION3;
        …..
        ACTIONn;
end
```

Remarque

Dans les as courants: l'incrémentation ce fait par 1, sauf autres indications. Par exemple si l'on veut calculer les carrés des nombres pairs entre 0 et 10;

```
for i=0:2:10
        carre = i^2
end
```

La boucle *while*

La boucle while permet de répéter une opération tant qu'une condition n'est pas remplie (ou le contraire: tant qu'une condition est vérifiée). On écrit ce type de boucle de la manière suivante:

```
While CONDITION
        ACTION1;
        ACTION2;
        ACTION3;
        …..
        ACTIONn;
end
```

La boucle *switch*

Cette boucle permet par fois de remplacer les boucles if-elseif-else.

```
switch (CAS)
case {CAS1}
        ACTION1
case {CAS2}
        ACTION2
otherwise
        ACTION3
end
```

Exemple

Détermination de l'exponentielle ou le logarithme en bas e d'un nombre entré par l'utilisateur.

```
operation = input('Opération: (1) exp ; (2) log ?');

nombre = input ('Valeur: ');

switch operation
```

```
        case 1
                b = exp (nombre)
        case 2
                b = log (nombre)
        otherwise
                disp('mauvais choix – operation')
end
```