Université Mohammed V - Agdal

Faculté des Sciences de Rabat
Département de Mathématiques

Cours d'introduction à MATLAB

Année universitaire : 2014/2015

Introduction à MATLAB

MATLAB est un langage de programmation de haut niveau, intégrant plusieurs fonctionnalités, il fournit un environnement interactif pour la création et la gestion de programmes. Dans ce qui suit on introduit quelques fonctionnalités de bases qui nous permettrons d'apprendre à manipuler le logiciel et de nous familiariser avec le langage.

L'élément fondamental chez MATLAB est la matrice, puisque le type de base des données est le type « array ». Scalaires, vecteurs, matrices réelles et complexes sont considérés comme étant des cas spéciaux du type de base.

1. Arithmétique simple

MATLAB permet l'utilisation d'opérateurs arithmétiques classiques que sont $+ \; - \; / \; * \; ^{\wedge}$ pour effectuer des opérations directement sur des nombres ou sur des variables après leur avoir affecté des valeurs.

Exemples :

```
>> 2 + 2

ans =

    4
```

L'addition ci-dessus est effectuée directement sur les nombres et le résultat est stocké dans la variable par défaut 'ans' et affiché.

```
>> x = 2 + 2

x =

    4
```

Le résultat de l'addition est stocké dans la variable 'x', grâce à l'opérateur d'affectation '='.

```
>> y = 3 + 5 - 7;

>> 2*y+2

ans =

    4
```

On remarque que le résultat de la première opération ne s'affiche pas, car un point virgule à la fin de chaque instruction empêche l'affichage, chose très utile pour empêcher l'affichage de résultats intermédiaires.

```
>> 4*(3-2)/7+5
```

ans =

5.5714

Lors de l'existence de plusieurs opérateurs dans une même expression les opérations suivent l'ordre de priorité usuel.

2. Entrées/Sorties et formatage de données

2.1. Entrées/Sorties

MATLAB offre la possibilité de communication avec l'utilisateur grâce aux entrées/sorties de données qui peuvent se faire à travers plusieurs manières différentes, dans cette introduction on ne s'intéressera qu'à la communication par clavier.

Pour instruire au programme d'attendre la saisie d'une valeur pour 'x' on écrit :

```
>> x = input('Entrer une valeur pour x : ')
```

Le résultat est le suivant :

Entrer une valeur pour x : 5

x =

5

Quand à l'affichage de données, au lieu d'écrire le nom de la variable directement il y a d'autre moyen pour afficher sa valeur à l'écran, deux commandes utiles sont 'disp()' et 'fprintf()' :

```
>> disp('la valeur de x : '), disp(x)
```

la valeur de x :

5

```
>> f = 19 ;

>> c = (5/9)*(19-32) ;

>> fprintf('%5.2f en Fahrenheit est égale à %5.2f en Centigrade. \n',f,c)
```

19.00 en Fahrenheit est égale à -7.22 en Centigrade.

2.2. Formatage de données

MATLAB offre plusieurs format qui permettent l'affichage de plus ou moins de 'digits' selon le besoin :

```
>> format rat  %forme rationnelle

>> format long  %forme décimale avec 14 digits
```

>> format long e %forme exponentielle longue

>> format hex %forme hexadécimale telle que représentée en mémoire

>> format short e %forme exponentielle courte

>> format short %forme par défaut à 4 digits

Le signe '%' permet de mettre un commentaire car tout ce qui vient après est ignoré.

3. Matrices et vecteurs

3.1. Initialisation

Les matrices sont saisies sur une seule ligne où les éléments consécutifs des lignes sont séparées par un espace ou une virgule et les lignes sont séparées par un point virgule, le tout doit être entre crochets.

Exemple :

```
>> M = [2 6 3 ; 3 8 1 ; 1 5 3]

M =

   2    6    3

   3    8    1

   1    5    3
```

Cas spéciaux : vecteurs et scalaires

Le vecteur est un cas spécial d'une matrice avec une seule ligne ou une seule colonne, tandis qu'un scalaire est considéré par MATLAB comme une matrice 1x1 et n'a pas besoin de crochets lors de sa saisie.

Exemples :

```
>> V_l = [2 6 3]  %vecteur ligne

V_l =

   2    6    3

>> V_c = [6 ; 8 ; 5]  %vecteur colonne

V_c =

   6

   8

   5
```

>> V_c = [6 8 5]' %vecteur colonne également (utilisation de la transposée)

V_c =

6

8

5

Continuation

S'il n'est pas possible d'écrire toute l'instruction sur une seule ligne on peut alors utiliser trois points (…) pour signaler à MATLAB que la suite de cette instruction écrite sur la ligne en cours se fera sur la ligne suivante.

Exemple :

>> M = [4 8 6 ; 4 3 ...

15 ; 0 4 7]

M =

4 8 6

4 3 15

0 4 7

3.2. Appel d'éléments

Une fois qu'une matrice ou qu'un vecteur existent on peut accéder à leurs éléments en spécifiant les indices de leur ligne et de leur colonne. La notation M(i,j) spécifie donc, l'élément de la matrice M existant sur la ligne i et la colonne j, ce qui est une chose commune pour les logiciels de calcul scientifique et les langages de programmation. Cependant, MATLAB permet une plus grande flexibilité au niveau de la manipulation des matrices dans le sens où il permet la spécification d'un ensemble de ligne et de colonne en même temps grâce au caractère ':'.

Exemples :

Vecteurs : V(indice_élément)

>> V = [5 7 9 6 3] ;

>> V(2) %donne le 2ème élément du vecteur

>> V(3:5) %donne les éléments du 3ème jusqu'au 5ème

>> V(:) %retourne le vecteur sous forme de vecteur colonne

>> V(end) %retourne le dernier élément du vecteur

MATRICES : M(indice_ligne, indice_colonne)

```
>> M = [5 7 6 ; 4 9 6 ; 7 8 3] ;
```

```
>> M(3,2) %retourne la valeur 8
```

```
>> M(2:end,1:2) %retourne les éléments sur la 2ème et 3ème ligne et la 1ère et 2ème colonne
```

```
>> M(1,:) %retourne toute la première ligne
```

```
>> M(:,3) %retourne toute la dernière colonne
```

3.3. Matrices et vecteurs commun

MATLAB offre plusieurs commande pour aider à la génération et à la manipulation des matrices, ces commande appelées 'matrices d'utilité' permettent de créer des matrices et vecteurs communs.

Exemples :

```
>> eye(3) % Matrice unité de taille 3x3
```

```
>> zeros(3) % Matrice nulle de taille 3x3
```

```
>> ones(3) % Matrice de 1 de taille 3x3
```

Il existe également des commandes permettant de créer des matrices contenant des variables aléatoires générées selon les lois de probabilité uniforme ou normale 'rand' et 'randn' ainsi que des commandes permettant la création de vecteurs constitués de suite de nombre équidistants où l'on peut préciser le pas.

Exemples :

```
>> V = linspace(1,5,10) % Un vecteur constitué de 10 valeurs équidistantes entre 1 et 5

V =

 Columns 1 through 7

   1.0000    1.4444    1.8889    2.3333    2.7778    3.2222    3.6667

 Columns 8 through 10

   4.1111    4.5556    5.0000
```

```
>> V = 1:7 %Vecteur contenant des valeurs allant de 1 à 7 avec un pas de 1

V =

    1    2    3    4    5    6    7
```

```
>> V = 7:-1:1 %Vecteur contenant des valeurs allant de 7 à 1 avec un pas de -1
```

V =

7 6 5 4 3 2 1

3.4. Opérations sur les matrices :

Opérations usuelles

Les opérations usuelles sur les matrices peuvent être effectuées directement à l'aide des opérateurs arithmétiques ce qui constitue un grand avantage par rapport aux autres langages de programmation.

Exemples :

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

>> d = ones(2,1) ;

>> A'*d

ans =

   5

   7

   9

>> B = [3 4 ; 0 1] ;

>> C = [2 1 ; 0 1] ;

>> C/B

ans =

  0.6667   -1.6667

       0    1.0000
```

Opération élément par élément

MATLAB permet également d'effectuer des opérations élément par élément pour la multiplication, la division et la puissance et ceci en utilisant les opérateurs : .* ./ .^

Exemple :

```
>> V = [ 7 5 9]; V.^2

ans =

  49   25   81
```

4. Fichiers script et fonction

4.1. Scripts

A travers son éditeur MATLAB offre la possibilité de créer des scripts, qui sont des fichiers d'extension '.m' regroupant une suite d'instruction. Un script peut être exécuté en écrivant son nom dans le 'prompt', son exécution est équivalente à la saisie des instructions une par une dans la fenêtre de commande.

Un fichier script peut contenir un nombre quelconque de commande ainsi que des appels à des fonctions déjà existantes ou écrites par l'utilisateur.

4.2. Fonctions

4.2.1. Fichier fonction

Un fichier fonction est également un fichier d'extension '.m' il commence par une ligne de définition ayant la syntaxe suivante :

```
function [ output_arguments ] = function_name( input_arguments )
```

Il faut faire attention à ce que le nom de la fonction '`function_name`' soit le même que le nom du fichier dans lequel la fonction est enregistrée.

L'appel de la fonction peut se faire à travers la saisie de son nom et des arguments correspondants. Les arguments de sorties, quand ils ne sont pas importants, peuvent être omis lors de l'appel.

Exemple :

```
function [a,p] = APDisque(r)
%---fonction calculant l'aire et le périmètre d'un disque de rayon r---%

a = pi.*r^2;
p = 2.*pi.*r;

end
```

L'appel à la fonction dans le 'prompt' se fait par :

```
>> [a,p] = APDisque(2);
```

4.2.2. Fonctions anonymes

Dans le cas d'une fonction ayant une structure simple, au lieu de créer un fichier on peu utiliser un 'handle' pour créer une fonction anonyme sur une seule ligne, sa syntaxe est la suivante :

```
handle = @(arglist)anonymous_function
```

'handle' est le nom qui sert à l'appel et à la manipulation de la fonction.

Exemple :

```
>> f  = @(x, y) x.^2 - sin(y) +exp(x^y)

f =

   @(x,y)x.^2-sin(y)+exp(x^y)

>> f(5,2)

ans =

  7.2005e+10
```

5. Boucles et Contrôle de flux

5.1. Boucle : *for indice = i_début : pas : i_fin – end*

La boucle 'for' permet une exécution répétitive d'un bloc d'instruction pour un nombre prédéfini de fois, elle possède la syntaxe suivante :

```
for i = indice_début : pas : indice_fin
    %%bloc d'instructions
end
```

Dans MATLAB il est d'usage d'initier l'indice dans les boucles à '1' et non pas à '0' pour éviter les erreurs lors de la manipulation des vecteurs et matrices puisque l'appel des premiers éléments de ces derniers se fait à travers l'indice '1'.

On peut forcer l'arrêt de n'importe quelle boucle en utilisant l'instruction break.

Exemple :

```
V = [1 4 7 10 13 16];
S = 0;

for i = 1 : length(V)
    S = S + V(i);
end
S
```

La boucle ci-dessus calcul la somme des éléments du vecteur 'V'.

5.2. Boucle : *while condition – end*

La boucle 'while' permet de répéter un bloc d'instruction tant qu'une condition prédéfinie est vraie, sa syntaxe est la suivante :

```
while condition
    %%bloc d'instructions
End
```

Exemple :

```
V = [1 4 7 10 13 16];
S = 0;
```

```
i = 1;

while i < 7
    S = S + V(i);
    i = i+1;
end
S
```

Dans l'exemple ci-dessus la boucle 'while' effectue la même opération que pour la boucle 'for' à savoir, le calcul de la somme des éléments du vecteur 'V'.

5.3. *if condition – elseif condition – else - end* et *switch – case – end*

Ayant une condition basée sur un opérateur logique ou relationnel, l'instruction conditionnelle if est utilisée pour ajuster l'ordre dans lequel les blocs d'instructions devront être exécutés. Quant au bloc switch case il est utile pour remplacer un if – elseif – elseif… – end multiple d'une manière habile, leur syntaxe est la suivante :

- Choix simple

```
if condition
    %%bloc d'instructions
End
```

- Choix multiple

```
if condition1
    %bloc d'instructions
elseif condtion2
    %bloc d'instruction
elseif condition3
    %bloc d'instructions
else
    %bloc d'instructions
end

switch expression
    case cas1, %bloc d'instructions
    case cas2, %bloc d'instructions
    case cas3, %bloc d'instructions
    otherwise %bloc d'instructions
end
```

Exemples :

Bloc de choix simple

```
t = 0;
if t>0
    signe = +1;
else
    signe = -1;
end
signe
```

Bloc de choix multiples

```
point = 8
if point >= 9, apprec = 'Excellent'
elseif point >= 8, apprec = 'Très bien'
elseif point >= 7, apprec = 'bien'
elseif point >= 6, apprec = 'assez bien'
else apprec = 'moyen'
end
```

Bloc switch

```
point = 7.5

switch floor(point) %entier inférieur ou égal à point
    case 9, apprec = 'Excellent'
    case 8, apprec = 'Très bien'
    case 7, apprec = 'bien'
    case 6, apprec = 'assez bien'
    otherwise apprec = 'assez bien'
end
```

6. Opérateurs relationnels et logiques

6.1. Opérateurs relationnels

MATLAB propose six opérateurs relationnels qui ont comme résultat un vecteur ou une matrice, ayant la même taille que les opérandes, contenant la valeur 1 quand la relation est vraie et 0 quand elle est fausse.

Ces opérateurs sont :

```
<  inférieur
<= inférieur ou égal
>  supérieur
>= supérieur ou égal
== égal
~= différent
```

6.2. Opérateurs logiques

Il y'en a quatre, ces opérateurs fonctionnent de la même manière que les opérateurs relationnels et produisent également des vecteurs et matrices contenant des 1 quand la relation est vraie et 0 quand elle est fausse.

Ces opérateurs sont :

```
&   ET logique
|   OU logique
~   Négation
xor OU exclusif
```

En addition à ces opérateurs MATLAB propose également plusieurs fonctions logiques telles que :

```
all  retourne 1 si tous les éléments d'un vecteur satisfont une
condition
any  retourne 1 si l'un des éléments d'un vecteur satisfait une condition
find retrouve les indices d'éléments non nuls d'une matrice
```

7. Graphiques

Pour la visualisation de données, MATLAB propose de très bon outils qui vont d'un simple graphe en 2D jusqu'à des illustrations tridimensionnelles animées, la manipulation des graphiques est simple et intuitive, nous présentons dans ce qui suit les outils de base pour les utiliser correctement sur MATLAB.

7.1. Graphiques 2D

La commande de base pour un graphique bidimensionnel sur MATLAB est la commande 'plot', elle s'écrit :

plot(valeur_X, valeur_Y, 'option_de_style')

Les options de style sont résumées dans le tableau suivant :

Options de couleur	Option de ligne	Options de marqueur
y : Jaune m : magenta c : cyan r : rouge g : vert b : bleu w : blanc K : noir	- solide -- hachurée : pointillé -. tiret-point	+ : singe plus O : cercle * : astérisque x : marque x . : point ^ : triangle s : carré d : diamand

Titres, labels, échelles et légendes :

L'annotation d'un graphique peut se faire grâce aux commandes 'xlabel', 'ylabel', 'title' et 'text' qui écrivent du texte respectivement sur l'axe des abscisses, l'axe des ordonnées, au dessus du graphique et sur un point dont les coordonnées sont donnés en arguments.

La légende peut être produite à travers la commande 'legend' qui est plutôt versatile, dans le sens où elle peut prendre une grande quantité d'arguments.

Contrôle des axes :

En utilisant la commande :

axis([xmin xmax ymin ymax])

On arrive à changer les limites des axes, on peut alors utiliser cette commande pour un zoom in ou un zoom out sur une partie du graphique.

Un graphique, plusieurs fonctions :

MATLAB offre trois manières de tracer plusieurs fonctions sur un seul graphique, on n'utilisera ici que la commande 'plot' qui est la plus intuitive, sa syntaxe pour plusieurs fonctions est la suivante :

plot(x1, y1, 'options', x2, y2, 'options', x3, y3, 'options')

Partition des fenêtres :

Sur MATLAB on a la possibilité de créer plusieurs graphiques sur une seule fenêtre puisque la fenêtre graphique est séparée en n fenêtre verticale et m fenêtre horizontales, on utilise la notation suivante :

subplot(m, n, q), plot(x,y)

Où q est le graphique en cours.

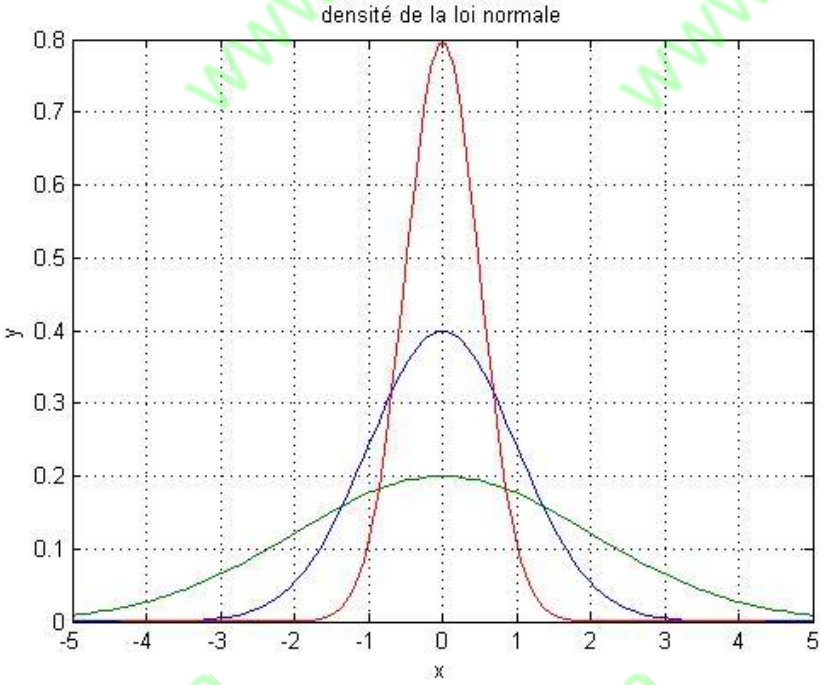
Exemple :

```
>> f1 = @(x) (1./(sqrt(2*pi))).*exp(-(x.^2)./2) ;

>> f2 = @(x) (1./(2*sqrt(2*pi))).*exp(-(x.^2)./8);

>> f3 = @(x) (1./(0.5*sqrt(2*pi))).*exp(-(x.^2)./(2*0.5^2));

>> plot(x1, f1(x1), x1, f2(x1), x1, f3(x1)), xlabel('x'), ylabel('y'), grid on, title('densité de la loi normale')
```



L'exemple ci-dessus illustre quelques une des fonctions de base citées dans ce cours.

7.2. Graphique 3D

Les possibilités qu'offre MATLAB au niveau de la visualisation 3D sont hors de la portée de ce cours cependant pour donner une simple idée sur l'allure d'un graphique en 3D créé sur MATLAB on procède directement par un exemple utilisant les commandes 'mesh' et 'meshgrid'.

Exemple :

```
>> x = linspace(0,5,6) ;

 >> y = linspace(0,5,6) ;

>>[X Y] =  meshgrid(x,y) ;

>> Z = X.^2 .* Y.^2;

>> mesh(Z)
```

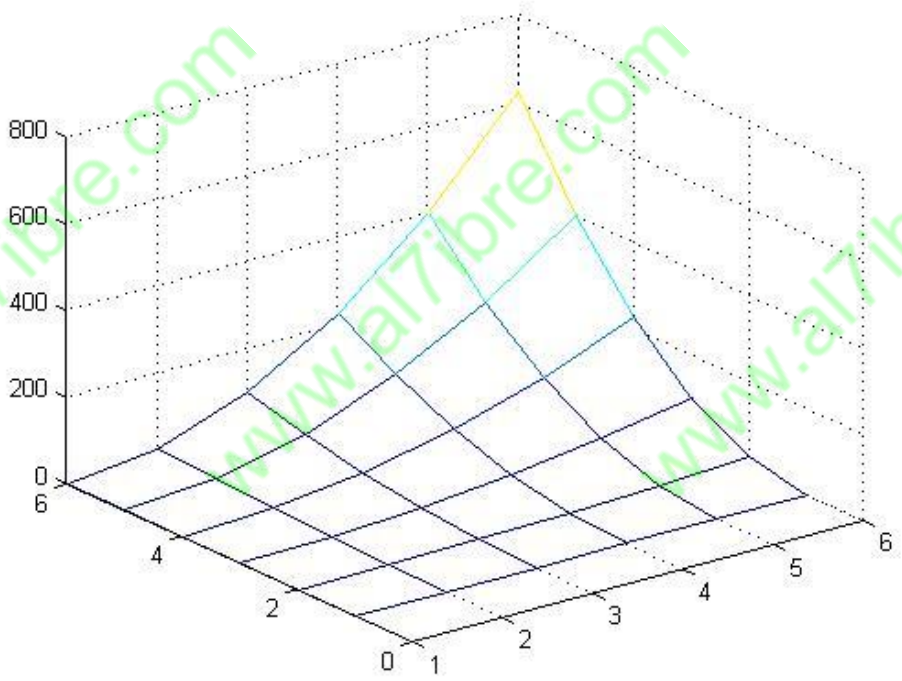


Table des matières