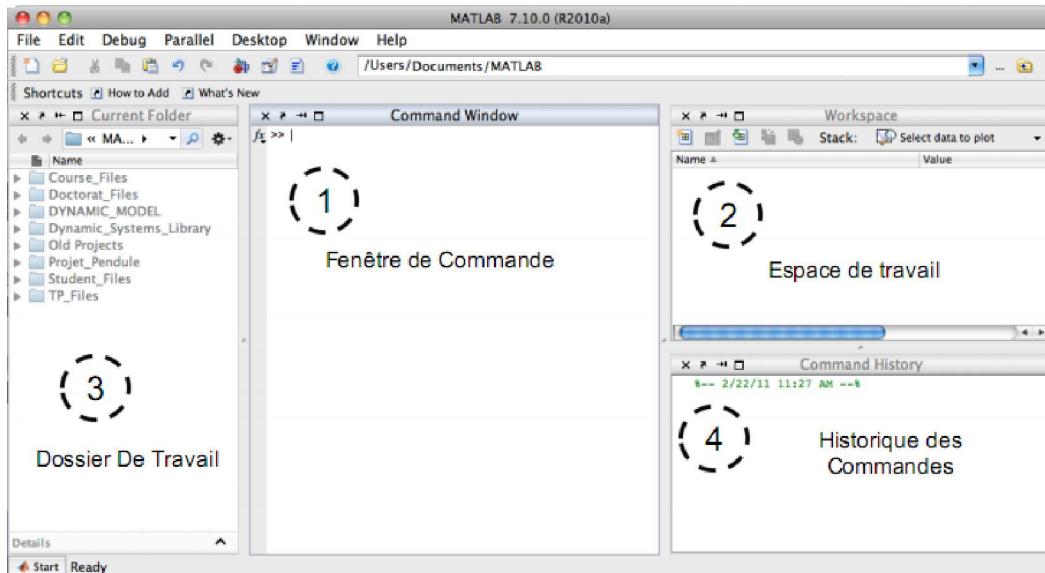


TP Informatique 03

01

1 Prise en main de Matlab

L'interface Matlab se compose d'une fenêtre principale divisée en trois sous-fenêtres comme le montre la figure ci-dessous:



Le **Workspace** permet de gérer les variables utilisées. L'onglet **Command History** est visible par défaut; il indique les dernières commandes exécutées. Le **Current Directory** gère l'emplacement des fichiers. Celui-ci sera utile pour le travail avec les m-files. La fenêtre de commande (**CommandWindow**) est la fenêtre d'interaction avec Matlab, elle permet de faire tout les calculs scientifiques possible.

2. Manipulation de la ligne de commande:

Réaliser quelques opérations simples directement en mode interactif (lignes de commande). Le symbole [**>>**] indique à l'utilisateur où il faut rentrer la commande. On ne peut pas «revenir en arrière».

```
>>2+3
ans=
5
>>3*6
ans=
18
Si on rentre des commandes erronées, Matlab nous l'indique par un message d'erreur.
>>4*
??4*
|Error:Expected a variable, function, or constant, found "end
offline"
>>a
??Undefined function or variable 'a'.
```

Les touches [**↑**] et [**↓**] permettent de naviguer parmi les dernières commandes effectuées, ce qui peut être utile si l'on commet une erreur et qu'on veut éviter de taper à nouveau toute la commande.

3. Utilisation de l'aide en ligne: Une aide est intégrée pour chercher les noms des fonctions et programmes prédefinies dans Matlab. Ainsi, si vous connaissez le nom de la fonction que vous devez utiliser, alors pour en avoir un descriptif et un mode d'emploi (éventuellement des options, les paramètres d'entrée, etc.), il faudra taper:

```
>>help nom_de_la_commande ou >>doc help
nom_de_la_commande
```

4. Syntaxe de base et commandes élémentaires

1. Quels sont résultats fournis par les instructions suivantes:

```
>>x=1.2568
>>y=2.5,
>>z=3.26;
>>x=1.2568 y=2.5,
>>y=2.5 z=3.26;
>>y=2.5; z=3.26;
>>x=1.2568, y=2.5
>>A=5+6i, b=1+3j, d='Salut!'
>>%z=3+4
>>z
```

2. Que font les instructions **who** et **whos**

5. Vecteurs

Que renvoient les commandes suivantes:

1. **a=[123],**
- b=[1 2 3]**
- c=[1,2,3]**
- d=[1;2;3]**
- e=[1,2,3]'**
- f=[1 2 3]**

```

g=[1;2;3]
h=[1:2:3]
2. v=[2,-3+i,7], v',v.', w=[-3;-3-i;2], v+w, v+w',
v*w, w*v, w'.*v, w'./v, w.^3
3. a=0:10,
b=[0:10]
c=0:1:10
c1=[0:1:10]
f=0.10,
g=[0,10]
h=[0;10],
d=10:0,
e=10:-1:0
k=0.7:3.4
4.a=[0:0.1:1]
b=[0:0.15:1]
c=[1:-0.15:0],
d=linspace(0,1,10),
e=ones(1,4)
f=3*ones(1,6)
g=ones(size(a))
h=zeros(1,3)
h=zeros(4,1)
k=rand(1,5)
z=[10:2:-10]
a=[0:0.1:1]
a(0),
a(1),
A(1),
a(1,1),
a(end),
a(1,end),
a(1)=[]

```

5. Quelle est la taille des vecteurs **a** et **k** ?

6. Expliquer les différences entre les commandes: **[début: fin]**; **[début: pas: fin]**; **linspace(a,b,n)**. Indiquez une méthode pour partager un intervalle borné en 50 intervalles de même longueur.

6. Matrices

Soit **A= 1 2 3** et **u=2**

$$\begin{matrix} 4 & 5 & 6 & & 4 & \\ & & & & & 6 \end{matrix}$$

1. Que reviennent les instructions suivantes:

A1=[111;12], A2=[1 1 1;1 2], A3=[1 2 3

$$\begin{matrix} & & 4 & 5 & 6 \end{matrix}$$

A4=[1 2 3 ...

$$\begin{matrix} 4 & 5 & 6 \end{matrix}$$

B=zeros(2,3)
C=diag([1,2])

D=eye(2,4)

F=[A, B;D,E],

G=[A, zeros(2,3); diag([1,2]), eye(2,4)],
H=[A, B,C,D]

2. Quelle est la taille de **F** ?

3. Quels sont les résultats des instructions suivantes:
F(1,:),
F(:,6),

F(:,end),

A(:),

A([1 3],[2 3]),

A(end)=[],

F([1 3],1:2),

F([1 3],[1,2]),

F([1 3],:),

F([1 3],end),

A',

diag(A),

diag(diag(A)),

triu(A),

tril(A),

rand(size(A)),

eye(size(A))

4. Remplacer la dernière colonne de **F** par le vecteur (7,8,9,10), puis extraire la matrice correspondant aux lignes 2 et 4, et aux colonnes 2 et 6.

5. Quel l'effet de **A(1,2)=0**, puis de **A(4,2)=5** ? Pourquoi **A(10)=6** ?

6. Que donnent:

B=A+2,

C=A*3,

D=A+3*eye(size(A))

7. Comparer: **E=A*A**,

f=A.*A,

G=A³,

I=exp(A),

J=sqrt(A),

K=A*u,

L=A.*A',

M=A*A'

8. Quel est le rôle du point dans la ligne précédente ?

TP Informatique 03
02

Instructions et commandes structurées

Une instruction peut désigner un instruction simple (expression), conditionnelle, une boucle ou une rupture de séquence. Une commande structuré permet de réaliser des itérations ou des sélections. MATLAB dispose des instructions structurées suivantes: *for*, *while*, *if*, et *switch*.

A) Instruction *for*

Syntaxe

for variable=expression

 Instructions

end

Les instructions sont répétées un nombre de fois donné. Les colonnes de "expression" (matrice) sont affectées, l'une parés l'autre, à la variable et les instructions sont exécutées. "expression" est généralement un vecteur ligne de la forme: **debut:pas:fin**.

for variable=**debut:pas:fin**

 Instructions

end

Exemples

```
ex1.m
for x=[1 2 3;4 5 6]
    disp(x)% afficher x
end
ex2.m
for y=[1 2 3 4 5 6]
    disp(y)
end
```

1. Générer un vecteur contenant les racines carrées des nombres entiers allant de 1 à n en utilisant la boucle *for*.

ex3.m

```
clear all;clc
n=input('donnez la valeur de n:'), x=[ ]; for i=1:n, x=[x, sqrt(i)];end
sous une autre forme:
```

exfor1.m

```
n=input('donnez la valeur de n:'),
x=[ ];
for i=1:n
x=[x, sqrt(i)];
end
x
```

exfor2.m

```
n=input('donnez la valeur de n:'),
for i=1:n
y(i)= sqrt(i);
end
y
```

2 .Que retournent les deux programmes suivants:

ex4.m

```
n=input('donnez la valeur de n:');  
x=[ ];  
for i=n:-1:1  
x=[x, sqrt(i)];  
end  
x
```

ex5.m

```
for a = 10:20  
    fprintf(' valeur de a: %d\n', a);  
end
```

B)Instruction while

Syntaxe

```
while expression booléenne  
    instructions  
end
```

Les instructions sont répétées aussi longtemps que "expression" est **différente de zéro** (expression booléenne vraie).

l'expression peut être simple ou composée d'expressions reliées entre elles par des opérateurs relationnels ou logiques ($=$, $<$, $>$, \leq , \geq , $\&$, $\mid\sim=$, etc.). Si "expression" est une variable, les instructions sont répétées si la partie réelle de cette variable a tous ses éléments non nuls.

3. Refaire la question 1) en utilisant l'instruction *while*

4. Afficher le plus grand entier naturel n tel que $2^n < x$

exwhile.m

```
x=15  
while ( )
```

```
end  
disp(['x= ', num2str(x)])  
disp(['n= ', num2str( )])  
disp([2^n ', num2str( )])
```

5. Ecrire un programme Matlab pour calculer la somme: $\sum_{k=1}^n \sqrt{k} = \sqrt{1} + \sqrt{2} + \sqrt{3} + \dots + \sqrt{n}$ en générant

aussi le vecteur contenant les racines carrées des nombres entiers allant de 1 à n en utilisant les boucles **for** et **while** (modifiez le programme **exfor1.m**).

6. Calculer la somme: $\sum_{k=1}^n \sqrt{k}$ en générant aussi le vecteur contenant les racines carrées des nombres

entiers allant de 1 à n **sans utiliser aucune boucle**.

TP Informatique 03

03

A) Instruction de sélection if

Syntaxe 1

```
if expression booléenne
    instructions I1
else
    Instruction I2
end
```

Syntaxe 2

```
if expression booléenne 1
    Instructions I1
elseif expression booléenne 2
    Instruction I2
elseif expression booléenne 3
    Instruction I3
else
    Instructions exécutées si aucune autre expression
    n'est vraie
end
```

Les instructions "I1" sont exécutées si "expression" est **différente de 0** (expression booléenne **vraie**), autrement les instructions "I2" sont exécutées. La partie **else** est facultative. En cas d'instructions conditionnelles imbriquées, un **else** est toujours associé au **if** le plus proche. Dans le cas de sélections multiples, on utilisera l'instruction **elseif**.

1. Ecrire dans M-file un programme qui calcule les racines d'une équation du 2ième degré.

2. Vérifier le résultat avec la fonction prédéfinie **roots()**.

3. Ecrire un programme qui affiche la mention obtenue en fonction de la moyenne de l'étudiant:

```
si la moyenne <10 , l'étudiant est Ajourné
si la moyenne >=10 , la mention est Passable
si la moyenne >=12 , la mention est Assez bien
si la moyenne >=14 , la mention est Bien
si la moyenne >=16 , la mention est Très Bien
```

B) Boucles imbriquées

MATLAB permet également d'utiliser une boucle dans une autre boucle. La syntaxe d'une instruction de boucle imbriquée dans MATLAB est comme suit:

% Pour for :

```
for i = 1:n
    for k = 1:m
        Instructions
    end
end
```

% Pour while :

```
while exp. bool. 1
while exp. bool. 2
    Instructions
end
end
```

```
% Génération d'un carré magique de taille 3x3
M = magic(3);
% Affichage de la matrice
disp('Carré magique de taille 3x3 :')
disp(M)
% Utilisation de boucles imbriquées pour afficher chaque élément
disp('Éléments du carré magique :')
for i = 1:size(M, 1) % Parcourir les lignes
    for j = 1:size(M, 2) % Parcourir les cols.
        fprintf('M(%d, %d) = %d\n', i, j, M(i, j));
    end
end
Carré magique de taille 3x3 :
8 1 6
3 5 7
4 9 2
Éléments du carré magique :
M(1, 1) = 8    M(2, 1) = 3    M(3, 1) = 4
M(1, 2) = 1    M(2, 2) = 5    M(3, 2) = 9
M(1, 3) = 6    M(2, 3) = 7    M(3, 3) = 2
```

4. En utilisant les instructions **for** et **while** construire un script qui en fonction de l'entier n affiche la matrice carrée:

$$A = (a_{i,j}), 1 \leq i, j \leq n \quad \text{où } a_{i,j} = \frac{1}{i^2 + j^2 + 1}$$

C) Instructions de rupture de séquence

break: termine l'exécution d'une boucle (*for* ou *while*). Si plusieurs boucles sont imbriquées, **break** permet de sortir de la boucle la plus proche.

continue: L'instruction **continue** est utilisée pour terminer l'itération en cours et force l'itération suivante d'une boucle **for** ou **while**.

ex_break.m

```
a = 10;  
while (a < 20 )  
    fprintf('valeur de a: %d\n', a);  
    a = a+1;  
    if( a > 15)  
        break; % termine la boucle en utilisant break  
    end  
end
```

valeur de a: 10
valeur de a: 11
valeur de a: 12
valeur de a: 13
valeur de a: 14
valeur de a: 15

ex_continue.m

```
a = 10;  
while a < 20  
    if a == 15  
        a = a + 1;  
        continue; % terminer cette itération  
    end  
    fprintf('valeur de a: %d\n', a);  
    a = a + 1;  
end
```

valeur de a: 10
valeur de a: 11
valeur de a: 12
valeur de a: 13
valeur de a: 14
valeur de a: 16
valeur de a: 17
valeur de a: 18
valeur de a: 19

D) Fichiers de Fonctions

Les fichiers de fonctions fournissent une extensibilité Matlab. Vous pouvez créer de nouvelles fonctions spécifiques à votre domaine de travail qui auront le même statut que toutes les autres fonction MATLAB.

Les variables dans les fonctions sont par défaut locales. Il est possible de définir des variable globales en utilisant le mot réservé **global** (exemple: **global x y**).

Définition et appel d'une fonction

function [y1,y2,.....,ym]=nom(x1,x2,....xn)

Corps de la fonction

où:

y_1, y_2, y_m : sont les arguments de sortie (retour).

x_1, x_2, x_m : sont les arguments d'entrée (d'appel).

Le fichier de la fonction doit être enregistré sous **nom.m**.

5. Ecrire une fonction MATLAB qui reçoit en entrée les coordonnées cartésiennes (x,y) d'un point et qui retourne en sortie ses coordonnées polaires (r,θ) .

$$r = \sqrt{x^2 + y^2}, \theta = \text{atg}\left(\frac{y}{x}\right)$$

6. Refaire les questions 1, 3 et 4 en transformant les fichiers scripts en fichiers de fonctions.

TP Informatique 03

04

1. Les polynômes

Matlab représente un polynôme sous forme d'un tableau de ses coefficients classés dans l'ordre des puissances décroissantes.

1.1 Saisie d'un polynôme

Le polynôme P d'expression: $P(x) = x^2 - 6x + 9$ est représenté par le tableau à 1 dimension suivant:

```
>>P=[1 -6 9]
```

```
P= 1 -6 9
```

Le nombre d'éléments du tableau est égal au degré du polynôme +1.

```
>>Q=[1 2 0 -3] %
```

```
Q= 1 2 0 -3
```

1.2 Racines d'un polynôme

La fonction *roots* permet de déterminer les racines d'un polynôme.

```
>>roots(P) %  $P(x) = (x-3)^2$ 
```

```
ans= 3
      3
```

```
>>roots(Q) %  $Q(x) = (x-1) (x^2 + 3x + 3)$ 
```

```
ans= -1.5000 + 0.8660i
      -1.5000 - 0.8660i
      1.0000
```

1.3 Evaluation de polynômes

La fonction *polyval* permet d'évaluer un polynôme en un point ou en un ensemble de points.

%Evaluer le polynôme P en 1: $P(1) = 1^2 - 6 * 1 + 9 = 4$

```
>>polyval(P,1) %
```

```
ans= 4
```

%Evaluer le polynôme Q en 0:

```
Q(0) = (0 - 1)(0^2 + 3 * 0 + 3)
```

```
>>polyval(Q,0)
```

```
ans= -3
```

Détermination d'un polynôme à partir de ses racines: **poly**

On cherche le polynôme qui a pour racines: 1, 2 et 3.

```
>>r=[1 2 3]
```

```
r=1 2 3
```

Le polynôme recherché est alors:

```
>>k=poly(r)
```

```
k=1 -6 11 -6
```

qui correspond à: $K(x) = x^3 - 6x^2 + 11x - 6$

On vérifie bien que les racines du polynôme K sont 1, 2 et 3.

```
>>racines=roots(K)
```

```
racines= 3.0000
```

```
2.0000
```

```
1.0000
```

1.4 Multiplication et division de polynômes

Soit deux polynômes $P1$ et $P2$ définis par:

```
 $p_1(x) = x + 2, \quad p_2(x) = x^2 - 2x + 1$ 
```

```
>>P1=[1 2]
```

```
P1=1 2
```

```
>>P2=[1 -2 1]
```

```
P2= 1 -2 1
```

```
>>P3=conv(P1,P2)% multiplication de P1 par P2
```

```
P3=1 0 -3 2
```

```
>>[Q,R]=deconv(P2,P1)% divison de P2 par P1
```

```
Q= 1 -4
```

```
R= 0 0 9
```

En divisant $P3$ par $P1$, on retrouve le polynôme $P2$ (Le reste R est nul car la divsion est exacte).

```
>>[Q,R]=deconv(P3,P1)
```

```
Q= 1 -2 1
```

```
R= 0 0 0 0
```

2. Représentation graphique

2.1 Représentation graphique du polynôme $K(x)$

$x=0:0.01:5;$ % Intervalle de la variable x

```
k=[1 -6 11 -6];
```

$y=\text{polyval}(k,x);$ % évaluation du polynôme $K(x)$ à chaque point de $\%x$, cette instruction est similaire à : $y=x.^3-6*x.^2+11*x-6;$

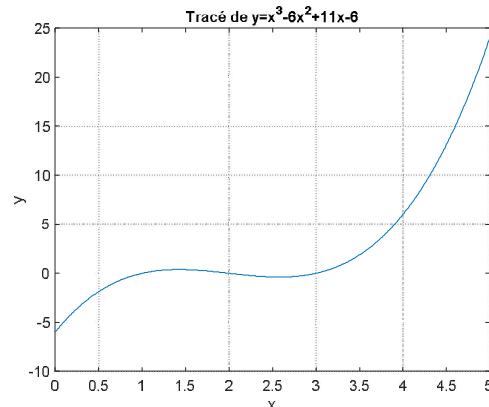
```
plot(x,y) % tracé de la fonction  $y=K(x)$ 
```

```
grid% Affiche le quadrillage
```

```
title('tracé de  $y=x^3-6x^2+11x-6$ ')%attribue un titre au %graphique
```

```
xlabel('x')% attribue un texte à l'axe des abscisses
```

```
ylabel('y')% attribue un texte à l'axe des ordonnées
```



plot(x,y,s): permet de tracer des graphiques de vecteurs de même taille (y en fonction de x). Le choix du type de la couleur du tracé peut se faire avec le paramètre facultatif 's' qui est une chaîne composée de 1 à 3 caractères. Pour plus de détail, tapez `>>doc plot`

Exemples: $t=0:10$; $y=t.^2$; figure(1), plot(t,y)

```
figure(2),plot(t,y,'k-')
figure(3), plot(t,y,'k.-')
figure(4),plot(t,y,'mo')
figure(5),plot(t,y,'m*:')
```

Echelles des axes: la fonction `axis()`

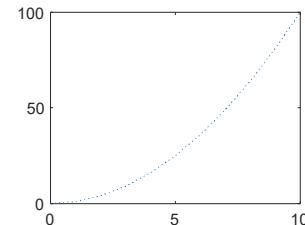
permet de fixer des intervalles de visualisation des différentes axes: `axis([xmin xmax ymin ymax])`

Subdivision de la fenêtre graphique

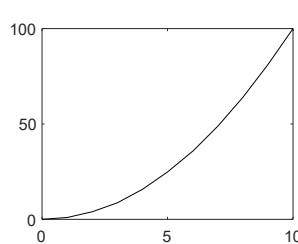
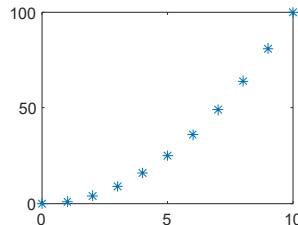
`subplot(m,n,p)`: divise la fenêtre graphique courante en $m * n$ zones graphiques (m: lignes; n: colonnes) et trace le graphique qui suit cette instruction dans la zone de numéro p.

Exemple

```
t=0:10; y=t.^2;
subplot(2,2,1) % ou subplot(221), emplacement 1
plot(t,y,'')
subplot(2,2,2) % emplacement 2
plot(t,y,'*')
subplot(2,2,4) % emplacement 4
```



```
plot(t,y,'k')
```



2.2 Graphe d'une fonction

Définition des fonctions: inline et function

Instruction de tracage: plot et fplot

Exemple: soit la fonction à représenter graphiquement:

$$f(x) = e^x - 3x^2$$

Méthode 1: (fplot)

```
fplot(@(x) exp(x)-3*x.^2)
grid on
```

Méthode 2: (inline + plot)

```
f=inline('exp(x)-3*x.^2')% ou f=inline('exp(x)-3*x.^2','x')
x=-10:0.1:10;
%évaluer la fonction f dans tous les points du vecteur x
ff=feval(f,x);
plot(x,ff)
axis([-5 5 -40 50]), grid
>>f
f= Inline function:
f(x) = exp(x)-3*x.^2
```

Méthode 3: (inline + plot)

```
f=inline('exp(x)-3*x.^2')
x=-10:0.1:10;
plot(x,f(x))% plot(x,feval(f,x))
axis([-5 5 -40 50]), grid
```

Méthode 4: (function + fplot)

```
fplot('fct',[-5 5]) %de la forme: fplot('fct',[xmin xmax])
grid
```

avec fct est une fonction définie comme suit:

```
fct.m
function y =fct(x)
y=exp(x)-3.*x.^2;
```

Méthode 5: (function + plot)

```
x=-10:0.1:10;
plot(x,fct(x))
axis([-5 5 -40 50]), grid
```

Méthode 6: (Une boucle pour évaluer la fonction à chaque valeur de x + plot)

```
y=[];
for x=-10:0.1:10
y=[y, exp(x)-3*x^2];
end
xx=-10:0.1:10;
plot(xx,y)
axis([-10 10 -10 100]), grid
```

Méthode 7: (Représentation vectorielle de la fonction + plot)

```
x=-10:0.1:10;
y=exp(x)-3.*x.^2;
plot(x,y)
axis([-10 10 -10 100]), grid
```

Méthode 8

% Tracer la fonction 'exp(x)-3*x^2' entre -5 et 5 en rouge

```
fplot('exp(x)-3*x^2',[-5 5],'r')
```

3. Question: Que fait le programme suivant:

```
x=-pi:0.1:3*pi;
figure(1),plot(x,y,x,sin(x))
y=x.*sin(x);figure(2), plot(x,y)
hold on,% Ajoute sur la même figure un autre graphe
plot(x,sin(x),'r')
figure(3),plot(x,y)
plot(x,sin(x),'r')
close all% ferme toutes les fenêtres graphiques
figure(4)
fplot(@(x) sin(x+pi/5),'LineWidth',2);
hold on
fplot(@(x) sin(x-pi/5),'--or');
fplot(@(x) sin(x),'-.*c')
hold off% annulation de l'effet de hold
figure(5)
fplot(@(x) sin(x+pi/5),'LineWidth',2);
hold on
fplot(@(x) sin(x-pi/5),'--or');
fplot(@(x) sin(x),'-.*c')
hold off
fplot(@(x) sin(x)),
```