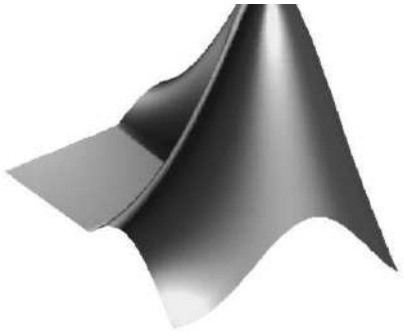
Université de Constantine 1
Département de Sciences et Technologie

Support de cours



MATLAB

PLAN

Présenté par : NOUAR Nabila **Dirigé par : Pr. MESSAI Abderraouf**

1. Prise en main MATLAB

1.1) Introduction

MATLAB est un système interactif de programmation scientifique, pour le calcul numérique et la visualisation graphique, basé sur la représentation matricielle des données. Le nom dérive de cette représentation : MATLAB = MATrix LABoratory. Ce langage contient un minimum de structures de programmation (structure itérative, structure conditionnelle, sous-routine). L'avantage est qu'il est très simple et très rapide à programmer, offrant une grande tolérance (syntaxe simple, pas de définition de types, etc), ce qui permet un gain appréciable en temps de mise au point. L'ingénieur peut par ce moyen être plus efficace dans l'analyse d'un problème.
MATLAB peut donc être utilisé pour la résolution approchée d'équations différentielles, d'équations aux dérivées partielles ou de systèmes linéaires, etc....
L'objectif de ce support de cours est double : la connaissance de ce logiciel est en soi indispensable parce qu'il est de plus en plus utilisé dans l'industrie et les banques pour développer des prototypes de logiciels et la mise en pratique des algorithmes d'analyse numérique étudiés.

1.2) Lancer MATLAB

L'interface MATLAB est composée des fenêtres suivantes : (Voir Figure 1)

A) Command Window: invite de commande permettant de taper des instructions, d'appeler des scripts, d'exécuter des fonctions matlab.

B) Command History : historique des commandes lancées depuis l'invite de commande.

C) Workspace : il liste les variables en mémoire, il permet également de parcourir graphiquement le contenu des variables.

D) Current Directory : un navigateur de fichier intégré à MATLAB pour visualiser le répertoire de travail courant et y effectuer les opérations classiques tel que renommer ou supprimer un fichier.

E) le Help browser : un navigateur permettant de parcourir l'aide de MATLAB. L'aide est un outil précieux pour trouver les fonctions et apprendre leur fonctionnement (notamment le format des données à fournir en entrée ainsi que les valeurs renvoyées par la fonction).

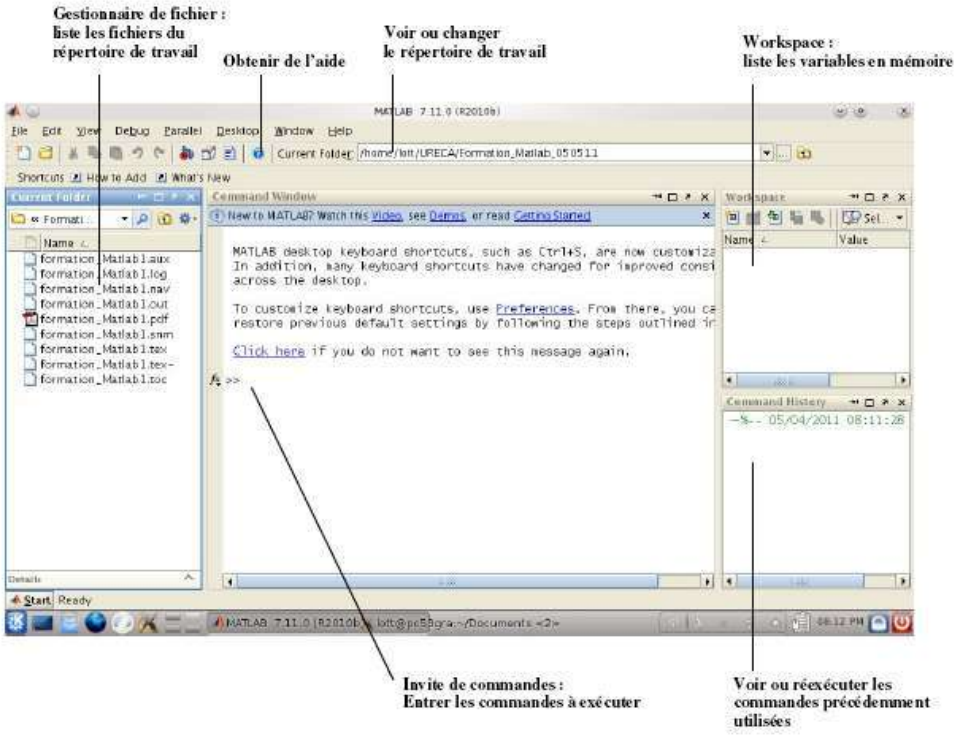


Figure1 : L'interface graphique de l'environnement MATLAB.

- Il existe deux modes de fonctionnement:

1) Mode interactif: MATLAB exécute les instructions au fur et à mesure qu'elles sont données par l'usager.
2) Mode exécutif: MATLAB exécute ligne par ligne un programme MATLAB (d'extension.m). Un 'programme MATLAB ' (ou 'm-file' en anglais) est une suite d'instructions MATLAB écrites dans un éditeur de texte et sauvées dans un fichier avec une extension .m.

2. Notions de base

2.1) Opérateurs et caractères spéciaux

Caractères spéciaux	Opérateurs
( ) parenthèses	**+ (addition)) >> 3+2**
= affectation	**- (soustraction)) >> 3-2**
, virgule	*** (multiplication)) >> 3*2**
; point virgule	**/ (division)) >> 3/2**
% commentaire	**^ (puissance)) >> 3^2**
: « jusqu'à »	

2.2) Syntaxe du langage

On distingue plusieurs types de variables selon les données qu'elles servent à stocker (nombre, caractère alpha-numérique, tableau, matrice, structure) :

A) Nombre

- Les nombres réels peuvent être écrits sous différents formats:

  5 1.0237 0.5245E-12

- Les nombres complexes peuvent être écrits sous forme cartésienne ou polaire:

Forme cartésienne: 0.5 + i*2.7 -1.2 + j*0.789 2.5 + 9.7i

Forme polaire: 1.25*exp(j*0.246)

B) Variable

- lorsqu'aucune variable d'affectation n'est pas spécifiée pour stocker le résultat de l'opération, MATLAB stocke le résultat dans une variable appelée **ans (diminutif pour answer)**, comme suit **:**

Exemple :

```
>> 3+2
ans =
      5
```

- **Sinon,** le nom d'une variable peut être spécifiée pour stocker le résultat de l'opération effectuée :

Exemple :

```
>>x = 3+2
x =
     5
```

- **Cette variable peut être utilisée dans l'opération suivante :**

Exemple :

```
>>x/2
ans =
      2.5000
```

- **Cependant, il existe des variables prédéfinies :**

  - **ans** : résultat de la dernière évaluation
  - **pi** : 3.1416…
  - **inf** : Infini (1/0)
  - **NaN** : "Not a Number" (0/0)
  - **i**, **j** : représentent tous les deux le nombre imaginaire unité (-1)

NB :

- Les noms de variable doivent commencer par une lettre, et sont sensibles à la différenciation des caractères (majuscule/minuscule) et ne peuvent contenir aucun caractère spécial exceptée le tiret bas.
- Il faut éviter d'utiliser comme nom de variable des noms déjà employés comme nom de fonctions (par ex : min, max, exp ...).
- MATLAB générera également une erreur si un des mots-clefs réservé suivant est utilisé : for, end, if, while, function, return.

C) Vecteurs et matrices

Les vecteurs et matrices peuvent être construits directement selon la syntaxe suivante

- Ils sont délimités par des crochets
- Les éléments sont entrées ligne par ligne
- Les éléments appartenant à la meme ligne sont séparés par des virgules ou des espaces
- Les différentes lignes doivent comporter le même nombre d'éléments et sont séparées par des point-virgule.

1) Vecteurs

On peut définir un vecteur x en donnant la liste de ses éléments:

a) Un vecteur ligne est un vecteur dont les éléments sont séparées par des virgules

```
>> x=[0.5, 1.2, -3.75, 5.82, -0.735]
   x =
0.5000 1.2000 -3.7500 5.8200 -0.7350
```

b) Un vecteur colonne est un vecteur dont les éléments sont séparés par des points-virgule

```
>> y=[0.5;1.2;-3.75;5.82;-0.735]
y =
0.5000
1.2000
-3.7500
5.8200
-0.7350
```

- **Des vecteurs définis au préalable peuvent être utilisés pour générer d'autres vecteurs**

Exemple

```
>> a=[1 2 3];
>> b=[4 5 6];
>> c=[a b]

   c =
   1 2 3 4 5 6
```

- **Si l'on veut déterminer la longueur d'un vecteur, on utilisera la commande "length" :**

```
>> l=length(c)
   l =
      6
```

2) Matrices

On définit aussi une matrice A en donnant ses éléments:

```
>> A=[0.5 2.7 3.9 4;4.5 0.85 -1.23 3;-5.12 2.47 9.03 2]
A =
 0.5000 2.7000 3.9000 4.0000
 4.5000 0.8500 -1.2300 3.0000
-5.1200 2.4700 9.0300 2.0000
```

Pour en connaître la dimension, on utilise la commande "**size**" :

```
>> [l c]=size(A)
   l =
      3
   c =
      4
```

Certaines matrices sont **prédéfinies** dans matlab. Ainsi "**zeros(m,n)**" crée une (m,n) matrice de zéros; "**ones(m,n)**" crée une (m,n) matrice de uns et "**eye(n)**" crée une (n,n) matrice identité.

Exemple

```
>> B=eye(4)
B =
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
```

NB :

Les éléments d'un vecteur ou d'une matrice peuvent être adressés en utilisant les indices sous la forme suivante:

- **t(10)** donne l' élément no. 10 du vecteur t

- **A(2,9)** donne l' élément se trouvant à ligne 2, colonne 9 de la matrice A
- **B(:,7)** donne la colonne 7 de la matrice B
- **C(3,:)** donne la ligne 3 de la matrice B

L'opérateur deux points (**:**) permet également de créer un vecteur ligne de la manière suivante :
<debut>:<incrément>:<fin>
si l'incrément n'est pas spécifié, il sera pris égal à 1.

3) Opérations matricielles (Voir Glossaire)

Voici quelques opérations matricielles exécutées par MATLAB :

B = A' La matrice B est égale à la matrice A transposée
E = inv(A) La matrice E est égale à la matrice A inversée
C = A + B Addition
D = A - B Soustraction
Z = X*Y Multiplication
X = B/A Division (Équivalent à B*inv(A))

[X = A\B Équivalent à inv(A)*B]

Note: Attention aux dimensions des multiplications et divisions de matrices.

4) Les opérations «ÉLÉMENT PAR ÉLÉMENT» :

Les opérations «élément par élément» des vecteurs et des matrices sont effectuées en ajoutant un point (.) avant les opérations * / ^

Exemple :

```
>> A=[1 2 3 4 5];
>> B=[6 7 8 9 10];
Essayez de faire C = A*B; et justifiez ce qui se passe;
puis essayez la multiplication 'élément par élément'
>> C=A.*B
C =
6 14 24 36 50
>> D=A./B
D =
0.1667 0.2857 0.3750 0.4444 0.5000
```

D) Fonctions

1) Expressions mathématiques (Voir GLOSSAIRE)

On écrit les expressions mathématiques de la façon habituelle:

```
z = 5*exp(-0.4*x).*sin(7.5*y);
```

2)Fonctions mathématiques (Voir GLOSSAIRE)

3) Fonctions trigonométriques (Voir GLOSSAIRE)

3) Gestion des données

3.1) Lecture et écriture au clavier et des fichiers

On peut avoir à lire des données numériques ou alphanumériques dans un fichier, par exemple les conditions initiales d'un calcul ou un maillage qui a été généré par un autre programme.

Exemple 1 :

M = dlmread('NomDeFichier','Delimiteur')
Lit des données numériques du fichier ASCII NomDeFichier, chaque nombre est séparé du suivant par le caractère Délimiteur ou par un retour _a la ligne. La virgule et l'espace sont des délimiteurs par défaut.
[A,B,C,...] = textread('NomDeFichier','format')

Lit les variables avec le format spécifiée jusqu'à épuisement du fihier. (Le nombre de variables à lire et le nombre de données dans le fichier doivent correspondre.)

Exemple 2 :

fid=fopen('NomDeFichier') % ouvre le fichier
A = fscanf(fid,format)
[A,count] = fscanf(fid,format,size)

Lit les données avec le format spécifiée. Un format est une chaine de caractères spécifiant en lecture le type de données à lire, c'est-à-dire :

'%d' pour un entier
'%f' pour un réel
'%c'pour un caractère

NB : Si on veut un format plus lisible, en particulier, afficher plusieurs variables sur la même ligne, on peut utiliser la commande **fprintf**.

Exemple :

a=1.5;
b=2;
fprintf('a = %f et b= %d',a,b);
a =1.5 et b=2

3.1)Enregistrer les variables de l'espace de travail dans un fichier

Pour enregistrer les variables de l'espace de travail dans un fichier, on utilise les instructions suivantes:
>> save
Enregistrer toutes les variables dans un fichier matlab.mat. Dans une session ultérieure, taper >> **load** pour ramener l'espace de travail enregistré .Si vous voulez enregistrer dans un fichier de nom choisi:
>>save fichier1.mat x y z A X
Enregistre les variables x, y, z, A, X dans le fichier fichier1.mat.
Dans une session ultérieure, taper **>> load fichier1.mat** pour ramener les variables x, y, z, A, X dans l'espace de travail.
Note: Si vous n'êtes pas dans un répertoire dans lequel vous avez le droit d'écrire, MATLAB refusera d'exécuter save

Pour effacer toutes les variables de l'espace de travail, on utilise la commande :

>> **clear**

3.2) Chargement de fichiers de données

A) Si des variables ont été sauvées en format Matlab dans le fichier fichier1.mat. Dans la session actuelle ou dans une session ultérieure, vous pouvez charger ces données en tapant:
>> load fichier1.mat (si vous êtes dans les répertoire contenant fichier1.mat) .Les données sont récupérées avec le nom des variables qui avaient été sauvées.
B) Si des données sont sous forme de matrices régulières dans un fichier (ex file500.txt), vous pouvez charger ces données en tapant:
>> load file500.txt

la matrice de données est alors disponible dans l'espace matlab sous le nom file500 (pas d'extension).
C) Si des données sont dans un fichier excel (ex sarrefl_TD.xls'), vous pouvez charger ces données en tapant:

>> [data,texte]=xlsread('sarrefl_TD.xls')

4) Programmer avec MATLAB

4.1) Fichiers scripts

Un fichier script est un fichier externe contenant une suite d'instruction MATLAB. Les fichiers de script ont une extension de nom de fichier .m. Les M-files peuvent être des scripts qui exécutent simplement une suite d'instructions ou peuvent ^être des fonctions
Exemple : créer le script "test script" (soit vous tapez >>edit test_script.m, soit vous faites 'File'->'New'->'Script' puis 'Save As' en spécifiant "test script.m" comme nom) avec la suite d'instructions suivante :
clear all %efface toutes les variables du workspace

a = 1
b = 2;
c = 3, d = 4;
e = a*b/(c+d),
scal = 11;

Il s'agit de sauvegarder puis exécutez le script (menu Debug->Save&Run ou Fleche verte ou F5). la sortie est affichée sur la ligne de commande .

4.2) Fonctions ou macros function

Les fonctions sont des enchainements de commandes Matlab regroupées sous un nom de fonction permettant de commander leur exécution.
On peut mettre dans une fonction un groupe de commandes destiné à être exécuté plusieurs fois au cours du calcul avec éventuellement des valeurs de paramètres différents.

A) M-files functions :

Dès que la fonction nécessite plusieurs instructions, il vaut mieux la définir dans un fichier à part à l'aide de l'éditeur de texte.
De manière générale, la syntaxe de définition d'une fonction externe est
function [y_1,...,y_m]=toto(x_1,....,x_n)

où toto est le nom de la fonction, x1,…,xn, les n arguments d'entrée et [y1,….,ym] les m arguments de sortie.
Les points verticaux symbolisent les instructions effectuées à l'appel de la fonction.

Exemple :

Prenons l'exemple d'une fonction **angle**, qui doit calculer l'angle formé par le segment d'extrémités (0,0) et (x,y) avec l'horizontale et aussi le dessiner. On peut créer le fichier angle.m contenant les lignes :

function [s]=angle(x,y)
s=180*atan(y/x)/pi;
patch([x,0,x],[0,0,y],'y')
axis equal

puis dans la fenêtre de commandes on tape
angle(4,5)

NB :
S'il n'y a qu'un résultat comme dans l'exemple de la fonction angle, on peut se dispenser de le récupérer dans une variable. En revanche s'il y a plus d'un paramètre de sortie, il faut récupérer leurs valeurs dans des variables dans le script d'appel.

Regardons par exemple l'utilisation de la fonction polaire définie ci-dessous et sauvegardée dans le fichier polaire.m

Exemple:

```
function [r,theta]=polaire(x,y)
r=sqrt(x^2+y^2);
theta=atan(y/x);
```

Pour l'utiliser _a partir de la fenêtre Matlab on tape les instructions suivantes

```
>>r=polaire(2,3) % ou bien
>>polaire(2,3) % Si seulement le rayon nous intéresse
ans =
3.6055513
>>[r,t]=polaire(2,3) %Si on veut récupérer a la fois le rayon et l'angle
t =
    .9828
r =
    3.6056
```

B) Inline functions

Une fonction ne comportant qu'un petit nombre d'instructions peut être définie directement dans la fenêtre de commandes de la manière suivante :

```
>>angle=inline('180*atan(y/x)/pi')
angle =
Inline function:
angle(x,y) = atan(y/x)
>>angle(5,4)
ans =0.6747
```

Les arguments de la fonction angle sont normalement fournis à l'appel dans l'ordre d'apparition dans la définition de la fonction. On peut aussi spécifier les arguments d'appel explicitement :

Exemple :

```
>>f = inline('sin(alpha*(x+y))','x','y','alpha')
f =
Inline function:
f(x,y,alpha) = sin(alpha*(x+y))
>>f(0.2,0.3,pi)
ans =1
```

C) Fonctions outils

Certaines commandes spéciales ne peuvent s'utiliser qu'en relation à une fonction :

nargin : donne le nombre d'arguments entrée passés à l'appel de la fonction

Exemple :

```
function c = testarg1(a,b)
if (nargin == 1)
c = 2*a;
elseif (nargin == 2)
c = a + b;
end
```

Cette commande peut être utilisée pour connaitre le nombre (prévu) d'arguments d'entrée

Exemple :

```
>> nargin('testarg1')
ans =
        2
```

4.3) Instructions de contrôle

A) Boucle For

Il y a deux types de boucles en MATLAB : les boucles while et les boucles for. La boucle for parcourt un vecteur d'indices et exécute à chaque pas toutes les instructions délimitées par l'instruction end.

```
>>x=1; for k=1:4,x=x*k, end
x =
     1
x =
     2
x =
     6
x =
    24
```

La boucle for peut parcourir un vecteur (ou une matrice) en prenant comme valeur à chaque pas les éléments (ou les colonnes) successifs.

```
>>v=[-1 3 0]
v =
-1 3 0
>>x=1; for k=v, x=x+k, end
x =
     0
x =
     3
x =
     3
```

B)Boucle While

Il arrive que nous souhaitions répéter une suite d'instructions jusqu'_a qu'une condition soit satisfaite. Si l'on ne connait pas le nombre d'itérations nécessaire à l'avance, une boucle while est préférable par rapport à une boucle for.

La boucle while exécute une suite de commandes jusqu'à ce qu'une condition soit satisfaite.

Exemple

```
>>x=1
 while x<14
x=x+5
end
x =
     6
x =
    11
x =
    16
```

C) Instruction conditionnée IF

Syntaxe

```
if expression
instructions I1
```

else
instructions I2
end

Exemple

if moy >= 5.5
mention = ‘très bien’;
elseif moy >= 5
mention = ‘bien’;
elseif moy >= 4.5
mention = ‘Assez bien’;
elseif moy >= 4
mention = ‘Passable’**;**
else
mention = ‘Ajourné(e)’;
end

D) Instructions de rupture de commande :

break : Termine l’exécution d’une boucle. Si plusieurs boucles sont imbriquées, break permet de sortir de la boucle la plus proche.
return : Cette instruction permet de revenir au fichier M ayant appelé le programme courant ou à la ligne de commandes MATLAB.
error(‘message’) : Affiche le message spécifié, émet un “bip” et interrompt l’exécution du programme.

Colours		Line Styles	
y	yellow	.	point
m	magenta	o	circle
c	cyan	x	x-mark
r	red	+	plus
g	green	-	solid
b	blue	*	star
w	white	:	dotted
k	black	-.	dashdot
		--	dashed

4.4) Tests

Un test est une alternative entre deux commandes (ou groupe de commandes) Matlab sélectionnées suivant
le résultat d'une opération logique. Le résultat d'une opération logique est une variable logique ou booléenne qui
vaut 1 pour VRAI et 0 pour FAUX.
Dans MATLAB on dispose du classique if-else agrémentée du elseif parfois bien utile. La syntaxe est :

Exemple :
>>x=16
x =
16
>>if x>0, y=-x, else y=x, end
y =
- 16

5) Représentation graphique sous MATLAB

Pour représenter des courbes du type y = f(x) ou des surfaces z = f(x; y), les données x,y,z doivent être des vecteurs colonnes (x et y) ou des matrices (z) aux dimensions compatibles. L'instruction de dessin correspondante (par exemple plot(x,y) pour tracer des courbes planes) est alors utilisée et éventuellement complétée par des arguments optionnels (couleur, type de trait, échelle sur les axes, etc...). La visualisation du résultat s'effectue dans une fenêtre graphique (avec possibilité de zoom, de rotation, d'impression).

5.1) Exemple de représentation graphique en deux dimensions (2D)

A) Fonction élémentaire :

La commande `plot` dont la syntaxe est la suivante :

plot(x,y,s)

permet de tracer des graphiques (courbes ou nuages de points) de vecteurs de dimensions compatibles (y en fonction de x).

B) Styles de lignes et couleurs:

Par défaut, les lignes sont bleues en trait plein. Il est possible de modifier le style et la couleur de la ligne en rajoutant un 3 ème argument à la fonction plot( ). Par exemple, pour afficher notre courbe avec un point vert par donnée, nous utiliserons l'option 'g.'

```
>>plot(x,y,'g.')
```

NB : Le type de tracé est par défaut le trait continu. De même, MATLAB fixe une couleur par défaut si elle n'est pas spécifiée.
Les styles de lignes disponibles ainsi que les codes associés sont résumés dans le tableau ci-dessous :

Exemple :

```
plot(x,cos(x),x,sin(x),x,exp(-x))  % Matlab va automatiquement utiliser des couleurs différentes pour chaque courbe%

plot(x,cos(x),'o-r',x,sin(x),'x-b',x,exp(-x),'*-g') % pour spécifier le type de symbole et la couleur à utiliser pour chaque courbe%

legend('cos(x)','sin(x)','exp(-x)')     % pour rajouter une légende
```

NB : Pour superposer des courbes par des appels successifs à cette fonction, il faut auparavant avoir utilisé la commande hold on, comme suit :

```
hold on
plot(x,cos(x),'o-r')
plot(x,sin(x),'x-b')
plot(x,exp(-x),'*-g')

legend('cos(x)','sin(x)','exp(-x)')
```

Le résultat de ce script est illustré par la Figure 2.

C) Affichage de plusieurs graphes côte à côte

Une fenêtre d'un graphique peut être divisée en sous-fenêtres suivant un tableau de dimension (mxn). On peut alors afficher dans chaque sous-fenêtre une ou plusieurs courbes :

```
>>subplot(2,2,1), plot(x,sin(3*pi*x))
>> ylabel('sin 3 pi x')
>>subplot(2,2,2), plot(x,cos(3*pi*x))
>> ylabel('cos 3 pi x')
>>subplot(2,2,3), plot(x,sin(6*pi*x))
>> ylabel('sin 6 pi x')
>>subplot(2,2,4), plot(x,cos(6*pi*x)), hold on, plot(x(1:10:end),cos(6*pi*x(1:10:end)),'r.')
>> ylabel('sin 6 pi x')
```

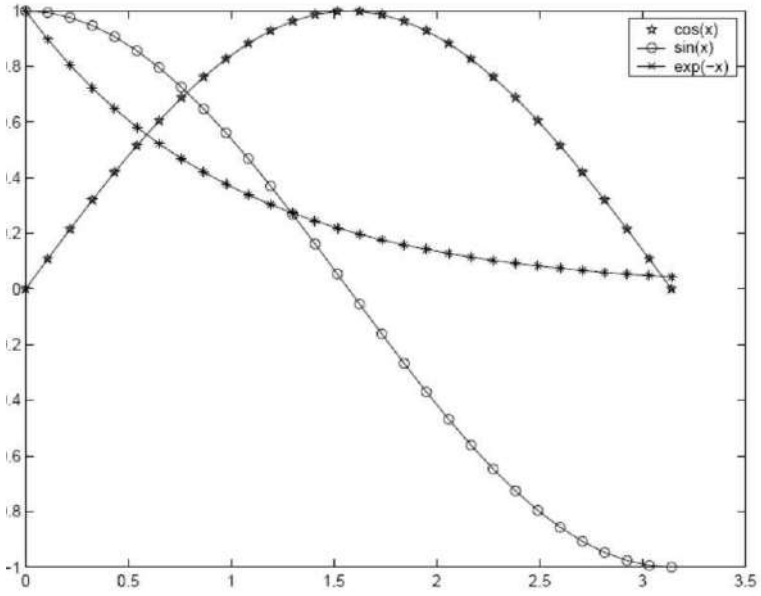


Figure2 : Exemple d'appels successifs à la fonction « plot ».

GLOSSAIRE

Fonction	Description
ones(i,j)	crée un tableau de i lignes j colonnes contenant des 1
zeros(i,j)	crée un tableau de i lignes j colonnes contenant des 0
eye(i,j)	crée un tableau de i lignes j colonnes avec des 1 sur la diagonale principale et 0 ailleurs
toeplitz(u)	crée une matrice de Toeplitz symétrique dont la première ligne est le vecteur u
diag(u)	crée une matrice carrée avec le vecteur u sur la diagonale et 0 ailleurs
diag(U)	extrait la diagonale de la matrice U
triu(A)	renvoie la partie supérieure de A
tril(A)	renvoie la partie inférieure de A
linspace(a,b,n)	crée un vecteur de n composantes uniformément réparties de a à b
A\b	résolution du système linéaire `Ax=b`
cond(A)	conditionnement d'une matrice (norme euclidienne)
det(A)	déterminant d'une matrice
rank(A)	rang d'une matrice
inv(A)	inverse d'une matrice
pinv(A)	pseudo inverse d'une matrice
svd(A)	valeurs singulières d'une matrice
norm(A)	norme matricielle ou vectorielle
u'	prend le transposé de u
u*v	multiplication matricielle
u+v	addition matricielle
u-v	soustraction matricielle
u.* v	multiplication des tableaux u et v terme à terme
u./v	division du tableau u par le tableau v terme à terme
find(C(A))	indices des composantes du tableau A vérifiant la condition C(A)

Principales opérations matricielles

sin	cos	tan	asin	acos	atan	atan2
sinh	cosh	tanh	asinh	acosh	atanh	

Fonctions trigonométriques

min valeur minimale	max valeur maximale	mean valeur moyenne	std écart type	cov covariance
sum somme	abs valeur absolue; module (nb complexe)	angle argument (nb. complexe)	real partie réelle	imag partie imaginaire
conj conjuguée (nb. complexe)	round arrondir	fix arrondir (vers zéro)	floor arrondir (vers $-\infty$)	ceil arrondir (vers ∞)
sqrt racine carrée	rem reste	exp exponentielle	log logarithme base e	log10 logarithme base 10

Fonctions mathématiques

Opérateurs et caractères spéciaux

Instruction	Description
`plot(x,y)`	tracé de la courbe passant par les points `(x,y)`
`loglog(x,y)`	idem avec échelle logarithmique sur les deux axes
`semilogx(x,y)`	idem avec échelle logarithmique sur l'axe Ox
`semilogy(x,y)`	idem avec échelle logarithmique sur l'axe Oy
`plotyy(x,y,x,z)`	courbe (x,y) avec l'axe 0y à gauche, et courbe (x,z) avec l'axe 0z à droite
`xlabel('label')`	légende pour l'axe 0x
`ylabel('label')`	légende pour l'axe 0y
`title('label')`	titre au dessus du graphique
`legend('lab1','lab2','lab3',...)`	légende avec une chaîne de caractères pour chaque courbe
`text(x,y,'label')`	chaîne de caractères à la position x,y
`plot3(x,y,z)`	tracé de la surface passant par les points `(x,y,z)`
`hold on, hold off`	active/désactive la conservation de la fenêtre graphique à l'appel de la fonction plot

Principales instructions graphiques

`dlmread('NomDeFichier','delimiteur')`	lecture du fichier
`dlmwrite('NomDeFichier',M,'delimiteur')`	écriture de M dans le fichier
`textread('NomDeFichier','format')`	lecture du fichier
`fid=open('NomDeFichier')`	ouverture du fichier NomDeFichier
`[A,count]=fscanf(fid,'format')`	lecture du fichier ouvert par open
`fprintf(fid,'format',données)`	écriture des données avec un format
`close(fid)`	fermeture
`fprintf('format',données)`	écriture des données avec un format
`fprintf('format',données)`	écriture des données avec un format

Les commandes d'entrée-sortie et les accès fichiers

Instruction	Description
`nargin`	nombre d'arguments d'entrée d'une fonction
`nargout`	nombre d'arguments de sortie d'une fonction
`error`	interrompt l'exécution de la fonction, affiche le message d'erreur et retourne dans le programme appelant.
`warning`	imprime le message mais ne retourne pas dans le programme appelant
`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 requièrant normalement l'intervention de l'utilisateur.
`break`	sort d'une boucle `while` ou `for`.
`return`	retourne dans le programme appelant sans aller jusqu'à la fin de la fonction.

Commandes de contrôle

français	test `Matlab`
et	&
ou	|
non	~
égal	==
différent	~=
plus petit que	<
plus grand que	>
plus petit ou égal à	<=
plus grand ou égal à	>=

Les opérateurs logiques dans les tests