جامعـــــة عبـــــاس لغرور خنشلـــــة
UNIVERSITE ABBES LAGHROUR KHENCHELA
ABBES LAGHROUR UNIVERSITY OF KHENCHELA

Support de cours

Analyse numérique 2

D. RAHAB Hichem

http://rahab.e-monsite.com

rahab.hichem@univ-khenchela.dz

Table des matières

Table des figures

Chapitre 1

Introduction au langage MATLAB

1.1 Introduction

MATLAB est un environnement de calcul numérique matriciel, il est basé sur le principe de matrice. Tous les types dans MATLAB sont à la base des matrices, un scalaire est une matrice de dimension 1×1, un vecteur est une matrice de $1 \times n$ ou $n \times 1$, etc. Ce principe est primordial à comprendre pour pouvoir travailler avec MATLAB. MATLAB crée une variable lors de son affectation, de ce fait on n'a pas besoin de déclarer les variables avant leur utilisation.

Le logiciel MATLAB consiste en un langage interprété qui s'exécute dans une fenêtre dite d'exécution. L'intérêt de MATLAB tient, d'une part, à sa simplicité d'utilisation : pas de compilation, pas besoin de déclaration des variables utilisées et, d'autre part, à sa richesse fonctionnelle : arithmétique matricielle et nombreuses fonctions de haut niveau dans divers domaines (analyse numérique, statistique, représentation graphique, ...).

1.2 Utilisation de MATLAB

On peut utiliser MATLAB en deux modes ; mode ligne de commande et mode script.

1.2.1 Mode ligne de commande

c'est-à-dire saisir des commandes dans la fenêtre d'exécution et les exécuter au fur et à mesure. Le mode ligne de commande permet d'obtenir des résultats rapides qui ne sont pas sauvegardés.

1.2.2 Mode script

Le mode script ou programmation, quant à lui, permet de développer des applications plus complexes, ainsi que les programmes sont sauvegardés pour faciliter une utilisation ultérieur. En écrivant dans des fichiers séparés (*.m) l'enchaînement des commandes, ces fichiers s'appellent des scripts et on les construit à l'aide de n'importe quel éditeur de texte (par exemple Notepad++ , ...).

1.3 Opérations sur Les matrices

La manipulation de matrices constitue le point le plus for du langage MATLAB, de ce fait, une large gamme de fonctionnalités sont à la disposition des utilisateurs de MATLAB pour une utilisation efficace et puissante des données matricielles.

1.3.1 La création de matrices

La création de matrices peut se faire de différentes matières et cela pour assurer une rapidité de créer les différents types de matrices.

Matrice 1×1

```
>> x=4

x =

     4
```

Matrice $1 \times n$

```
>> x=[1 2 3 4]

     x =

     1     2     3     4
```

ou bien :

```
>> x=[1, 2, 3, 4]

     x =

     1     2     3     4
```

Matrice $n \times 1$

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

x =

     1
     2
     3
     4
```

Matrice de 1 à n

Exemple : Matrice ligne des éléments de 1 à 10.

```
>> x=[1:10]

x =

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

Matrice de 1 à n avec un Pas

Exemple : Matrice ligne des éléments de 0 à 10 avec un pas de 2.

```
>> x=[0:2:10]

x =

     0     2     4     6     8    10
```

Matrice de $n \times n$

Exemple : Matrice de dimension $n \times n$.

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

x =

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

Remarque

— Le point-virgule (;) dans la matrice marque le retour à la ligne, alors qu'à la fin d'une instruction bloque l'affichage du résultat.

1.3.2 La transposée d'une matrice

Exemple : `x=[0 :2 ;4 :6]`, retourne :

```
x =
0 1 2
4 5 6
```

C'est une matrice à 2 lignes et 3 colonnes.
» `y = x'` retourne la matrice transposée :

```
>> y=x'

y =

     0     3
     1     4
     2     5
```

1.3.3 La taille d'une matrice

La taille de la matrice "y" est donnée par la fonction `size(y)` :

```
>> size(y)

ans =

     3     2
```

La réponse est : 3 lignes et 2 colonnes.

1.3.4 Sélection de lignes ou de colonnes

La sélection des lignes et de colonnes dans une matrice offre aux programmeurs de MATLAB un outil puissant de segmenter une matrices et obtenir des matrices personnalisées plus petites.
La colonne "j" de la matrice "x" est donnée par : `y(:,j)`, on dit qu'on a sélectionné tous les lignes (:) de la colonne "j". Exemple, pour j=2, on a :

```
>> y(:,2)

ans =

     3
     4
     5
```

La ligne i de la matrice x est donnée par : `y(i,:)` , pour i=2, on a :

```
>> y(2,:)

ans =

     1     4
```

Pour la sélection des éléments du diagonale d'une matrice, nous disposons de la commande **diag** :

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

A =

     1     2     3
     4     5     6
     7     8     9

>> B=diag(A)

B =

     1
     5
     9
```

1.3.5 Concaténation de matrices

La concaténation consiste à coller des matrices bout à bout afin d'obtenir une matrice supplémentaire. Cette opération s'effectue entre crochets. A l'intérieur de ces crochets, les différentes matrices doivent être séparées. Nous trouvons une concaténation verticale où les différentes matrices sont séparées par des points-virgules, et aussi une concaténation horizontale dans laquelle les matrices sont séparées par des virgules ou des espaces. Une utilisation combinée des concaténation horizontales et verticales permet d'obtenir des matrices plus importantes à partir de matrices simples.

Exemple 1 : Concaténation verticale

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

A =

     1     2     3

>> B = ones(2,3)

B =

     1     1     1
     1     1     1

>> C = [3 2 1]

C =

     3     2     1

>> X = [A ; B ; C]

X =

     1     2     3
     1     1     1
     1     1     1
     3     2     1
```

Exemple 2 : Concaténation horizontale

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

A =

     1
```

```
     2
     3

>> B = ones(3,2)

B =

     1     1
     1     1
     1     1

>> C = [3;2;1]

C =

     3
     2
     1

>> X = [A,B,C]

X =

     1     1     1     3
     2     1     1     2
     3     1     1     1
```

Exemple 3 : Concaténation horizontale et verticale combinées

```
>> Y=[A,B;C]

Y =

     1     1     1
     2     1     1
     3     1     1
     4     5     6
```

1.3.6 La matrice Identité

Pour une matrice carrée A d'ordre n, on a sa matrice identité qui est donnée par la fonction `'eye'` .

Exemple : pour n=3, on a :

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

A =
```

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

>> eye(size(A))

ans =

     1     0     0
     0     1     0
     0     0     1
```

1.3.7 Le produit de matrices

MATLAB offre deux différents produits de matrices, à savoir, le produit matriciel et le produit élément par élément.

A-Produit matricielle

Le produit matricielle c'est le produit connu en algèbre dans les mathématiques où on multiplie à chaque fois la ligne de la prmière matrice avec la colonne correspondante dans la deuxième matrice, ce produit n'est pas alors commutatif. Ce opération est effectuée par le symbole *.

Soient de matrices `"A"` de $n \times m$ et `"B"` de $p \times q$, alors le produit $C = A \times B$ n' est possible que si $m = p$. Dans ce cas, le coefficient c_{11} de cette matrice C s'écrit :

$$c_{11} = a_{11}b_{11} + a_{12}b_{12} + \ldots + a_{1m}b_{p1}$$

A-Produit élément par élément

Dans ce cas le produit n'est possible que si les deux matrices sont de même taille, car dans ce cas, le produit est obtenu en multipliant les éléments des deux matrices un à un. Ce produit est distingué par le point avant le symbole de multiplication (.*).

Exemple :

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

A =

     1     2     3
     4     5     6

>> B=[6 5 4; 3 2 1]

B =

     6     5     4
```

```
     3     2     1

>> A.*B

ans =

     6    10    12
    12    10     6
```

1.3.8 La matrice inverse

Soit A une matrice carrée non nulle. La matrice inverse A^{-1} de A (si elle existe) est telle que :

$$A * A^{-1} = Id$$

Dans MATLAB, cette matrice inverse est donnée par :

`A^(-1) ou bien inv(A).`

Exemple :

```
>> A=[1 3 5;2 -1 0;5 4 3]

A =

     1     3     5
     2    -1     0
     5     4     3
```

La matrice inverse de A est :

```
>> inv(A)

ans =

   -0.0682    0.2500    0.1136
   -0.1364   -0.5000    0.2273
    0.2955    0.2500   -0.1591
```

1.3.9 Autres fonctions

Soit x=[2 15 0] une matrice 1×3 (vecteur ligne).

sort(x) : donne une matrice ligne dont les éléments sont en ordre croissant :

```
>>sort(x)

ans =

      0  2  15
```

sort(x') donne une matrice colonne dont les éléments sont en ordre croissant :

```
>> sort(x')

ans =

      0
      2
      15
```

sum(x) calcule la somme des éléments de la matrice x.

```
>> sum(x)

ans =

      17
```

Pour trouver le maximum et le minimum du vecteur x, on utilise les fonctions max(x) et min(x) :

max(x)

```
>> max(x)

ans =

      15
```

min(x)

```
>> min(x)

ans =

      0
```

flipud(A) : Inverser l'ordre des lignes

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

     1     2     3
     4     5     6
     7     8     9
>> flipud(A)

ans =
```

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

fliplr(A) : Inverser l'ordre des colonnes.

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

     1     2     3
     4     5     6
     7     8     9

>> fliplr(A)

ans =

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

rot90(A,2) : Rotation de 90 degres (sens trigo).

```
>> A=[1 2 3;4 5 6 ;7 8 9];
>> rot90(A,2)

ans =

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

reshape(A,1,9) : changer la forme de la matrice.

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

ans =

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

1.4 Calcul des polynômes

Le calcul des polynômes sur MATLAB consiste à transformer une écriture de polynôme en une formule sur MATLAB par des variables et des coefficients entre ces variables, et avant cela il est nécessaire d'attribuer des valeurs numériques aux différentes variables, cela pour évaluer la formule.

Exemple 1 : Pour évaluer le polynôme suivant : $S = \pi R^2$, avec $R = 4$, on suit les étapes suivants :

```
>> R=4;                % affectation de la valeur 4 à la variable R
>> S=pi*R^2

S =

   50.2655              % Le résultat de calcul
```

Exemple 2 : Pour le deuxième polynôme : $P(x) = \frac{4x^2-2x+3}{x^3+1}$ avec : $x = 2$.

```
>> x=2;
>> p=(4*x^2-2*x+3)/(x^3+1)

p =

    1.6667
```

Opérateurs logiques :

~=	L'opérateur 'NON' (différent)
==	L'opérateur 'égal'
&	L'opérateur 'et'
‖	L'opérateur 'ou'
>	supérieur à
<	inférieur à
>=	supérieur ou égal
<=	inférieur ou égal

1.5 Le teste conditionnel 'if'

Ce test s'emploie, souvent, dans la plupart des programmes, il permet de réaliser une suite d'instructions si sa condition est satisfaisante.
Le test `'if'` a la forme générale suivante : `if-elseif-elseif-...-else-end`

```
if <condition 1>
     <instruction 1.1>
     <instruction 1.2>
     ...
elseif <condition 2>
     <instruction 2.1>
     <instruction 2.2>
     ...
elseif <condition 3>
     <instruction 3.1>
     <instruction 3.2>
     ...
...
```

```
...
else
     <instruction n.1>
     <instruction n.2>
     ...
end
```

Où `<condition 1>, <condition 2>, ...` représentent des ensembles de conditions logiques, dont la valeur est vrai ou faux. La première condition ayant la valeur 1(Vrai) entraîne l'exécution de l'instruction correspondante.

Si la valeur de `<condition k>` est à zéro, les instructions `<instruction k.1>, <instruction k.2>, ...` ne sont pas exécutées et l'interpréteur passe à la suite.

Si toutes les conditions sont fausses, les instructions `<instruction n.1>,<instruction n.2>, ...` sont exécutées.

Exemple 1

```
>> V=268.0826

V =

  268.0826

>> R=4

R =

     4

>> if V>150, surface=pi*R^2, end

surface =

   50.2655
```

Exemple 2 Pour calculer les racines d'un trinôme $ax^2 + bx + c$, on peut utiliser les instructions suivantes :

```
if a ~= 0
    sq = sqrt(b*b - 4*a*c);
    x(1) = 0.5*(-b + sq)/a;
    x(2) = 0.5*(-b - sq)/a;
elseif b ~= 0
    x(1) = -c/b;
elseif c ~= 0
    disp('Equation impossible');
```

```
else
    disp(' L''equation est une egalite');
end
```

Remarques

— La double apostrophe sert à représenter une apostrophe dans une chaîne de caractères. Ceci est nécessaire car une simple apostrophe est une commande MATLAB.
— La commande disp(' ') affiche simplement ce qui est écrit entre crochets.

1.6 Les boucles

1.6.1 La boucle for

Une boucle `for` répète des instructions pendant que le compteur de la boucle balaie les valeurs rangées dans un vecteur ligne.

Exemple : Pour calculer les 6 premiers termes d'une suite de Fibonacci

$$f_i = f_{i-1} + f_{i-2}, \qquad avec \qquad f_1 = 0 \qquad et \qquad f_2 = 1$$

On peut utiliser les instructions suivantes :

```
>> f(1) = 0; f(2) = 1;
>> for i = 3:6
f(i) = f(i-1) + f(i-2);
end
>> f

f =

     0     1     1     2     3     5
```

Remarques

— L'utilisation du point-virgule (;) permet de séparer plusieurs instructions MATLAB entrées sur une même ligne.
— On pourrait remplacer la seconde instruction par : `» for i = [3 4 5 6]`
— MATLAB n'exécute l'ensemble du bloc de commandes qu'une fois taper `'end'`.

1.6.2 La boucle 'while'

La boucle `while` répète un bloc d'instructions tant qu'une condition donnée est vraie.

Exemple : Les instructions suivantes ont le même effet que les précédentes :

```
>> f(1) = 0; f(2) = 1; k = 3;
>> while k <= 6
f(k) = f(k-1) + f(k-2); k = k + 1;
end
```

Remarques

— Le compteur 'k' est ajouté ici, ce compteur doit être initialisé et incrémenté pour assurer la condition d'arrêt de la boucle `while`.

1.7 Écriture de programmes MATLAB

Un nouveau programme MATLAB doit être placé dans un fichier, appelé m-fichier, dont le nom comporte l'extension `.m`.
Les programmes MATLAB peuvent être des scripts ou des fonctions.

1.7.1 Les scripts

Un script est simplement une collection de commandes MATLAB, placée dans un m-fichier et pouvant être utilisée interactivement.

Exemple : Pour le polynôme g(x) :

$$g(x) = \frac{2x^3 + 7x^2 + 3x + 1}{x^2 - 3x + 5.e^{-x}}$$

On peut écrire un script, qu'on choisit d'appeler TP, comme suit :

```
g=(2*x^3+7*x^2+3*x-1)/(x^2-3*x+5*exp(-x));
```

Ce script est enregistré dans le fichier TP.m.
Pour le lancer, on va taper simplement l'instruction `TP` après le prompt » MATLAB, bien sur après l'attribution d'une valeur à la variable x.

```
>> x=3;

>> TP

g =

    502.1384
```

1.7.2 Les fonctions

Une fonction est aussi définie dans un m-fichier, mais une fonction commence obligatoirement par une ligne de la forme :
`function [out1,... ,outn]=name(in1 ,... ,inm)` .
Où :

— `out1,...,outn` sont les variables de sortie sur lesquelles les résultats de la fonction sont retournés ;
— `in1,...,inm` sont les variables d'entrée, qui sont nécessaires à la fonction pour accomplir ses calculs.

Exemple 1 On définit une fonction `determ` , qui calcule le déterminant d'une matrice d'ordre 2 :

```
function det=determ(A)
[n,m]=size(A);
if n==m
  if n==2
    det = A(1 ,1)*A(2,2)-A(2 ,1)*A(1 ,2);
  else
    disp(' Seulement des matrices 2x2 ');
  end
else
   disp(' Seulement des matrices carrées ');
end
return
```

Cette fonction possède un seul paramètre de sortie `det` et un seul paramètre d'entrée `A`. Elle doit être enregistré dans un fichier intitulé `determ.m`.

Exemple 2 La fonction suivante permet de trouver les solutions d'une équation de $2^{ème}$ degré :

```
function [x1,x2]= degre2(a,b,c)
delta=b^2-4*a*c;
if delta < 0
    disp ('Pas de solution ...')
else
        x1= (-b-sqrt(delta))/(2*a);
        x2= (-b+sqrt(delta))/(2*a);
end
return
```

Cette fonction doit être enregistrer sur le fichier : degre2.m . La fonction `degre2` possède deux paramètres de sortier et trois paramètres d'entrée.
Voici deux exécutions en ligne de commande :

```
>> [r1,r2]=degre2(1,3,1)

r1 =
   -2.6180

r2 =
   -0.3820

>> [r1,r2]= degre2 (1,1,1)

Pas de solution ...
```

Remarque

L'instruction `return` peut être utilisée pour forcer une interruption prématurée de la fonction (quand une certaine condition est satisfaite).

1.8 Le graphisme

Pour tracer la courbe d'une fonction sous MATLAB on a besoin de :

— Définir le vecteur x, représentant les valeurs pour lesquels la fonction va être appliquée.

— Définir la fonction $y = f(x)$ comme la représentation du vecteur x par la fonction f.

— Appliquer la fonction `plot(x,y)`.

MATLAB permet d'ajouter un titre, des labels dans les axes x-axis et y-axis, et des lignes de grillage et aussi d'ajuster les axes pour convenir au graphe.

— `title` : Permet d'ajouter un titre au graph

— `xlabel` : Permet de générer une étiquette pour l'axe x.

— `ylabel` : Permet de générer une étiquette pour l'axe y.

— `legend` : Lorsque il s'agit de tracer plusieurs fonctions dans un même graphe, on peut mettre une légende pour les distinguer.

— `grid on/off` : quadrille ou non le graphique.

— `axis equal` : Uniformiser la distance dans les deux axes.

— `clf` : efface la figure en cours d'utilisation.

— `ginput(n)` : récupère les coordonnées de n points cliqués à la souris dans la figure en cours.

— `figure` :Permet de créer un nouveau graph.

L'exemple suivant va expliquer ce principe. Commençant par tracer la courbe de la fonction $y = x$ pour la plage de valeur de x de 0 à 100, avec un pas de 5. On va créer un script sur lequel on met :

```
x =[0:5:100];
y = x;
plot(x, y)
```

En exécutant le script la courbe suivante est obtenue :

Prenant un autre exemple pour tracer la courbe de la fonction $y = x^2$. Créer un script avec le code suivant :

```
x =[-100:40:100];
y = x.^2;
plot(x, y)
```

Changeons le code précédent, par la réduction du pas à 5 :

```
x =[-100:5:100];
y = x.^2;
plot(x, y)
```

MATLAB trace un graphe plus fine.

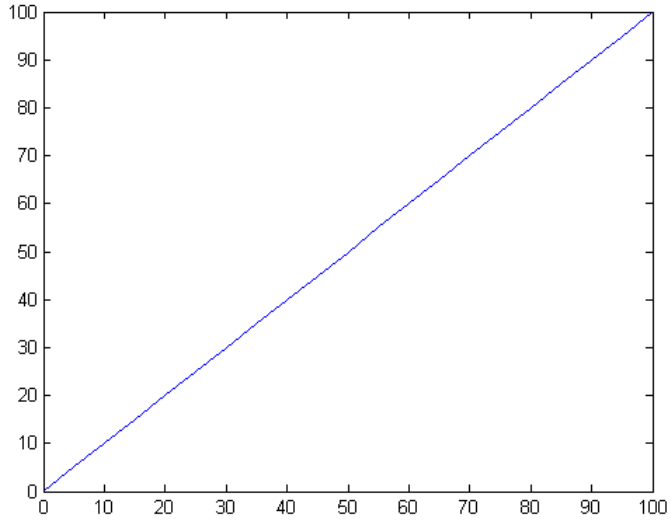

FIGURE 1.1 – Y=x

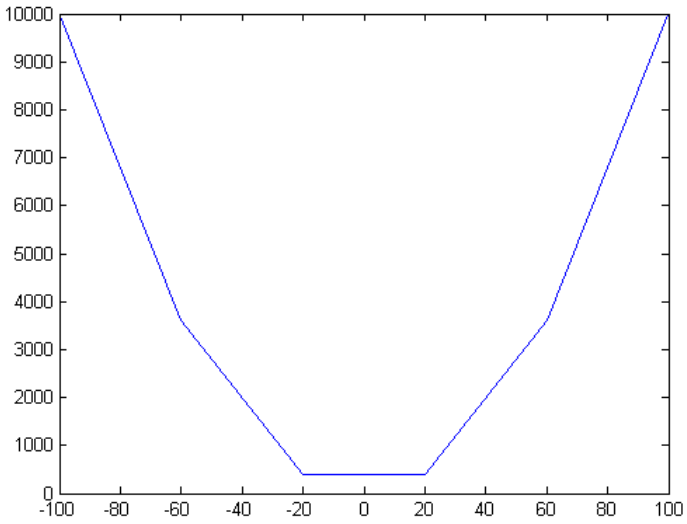

FIGURE 1.2 – Y=x

1.8.1 Graphe d'une fonction

Pour tracer le graphe d'une fonction dans un intervalle donnée on utilise la commande `fplot`.

Syntaxe : `fplot('fonction',[`$x_{min}, x_{max}, y_{min}, y_{max}$`]`, où :

- fonction = On donne le nom d'une fonction Matlab tel que `sin`, ou bien on définie une fonction donnée.
- x_{min}, x_{max} : C'est l'intervalle des abscisses dans lequel le graphe sera tracer.
- y_{min}, y_{max} : C'est un paramètre facultatif, par défaut il prend les valeurs minimums et maximums de la fonction sur l ?intervalle $[x_{min}, x_{max}]$, comme on peut préciser l'intervalle des ordonnées qu'on veut.

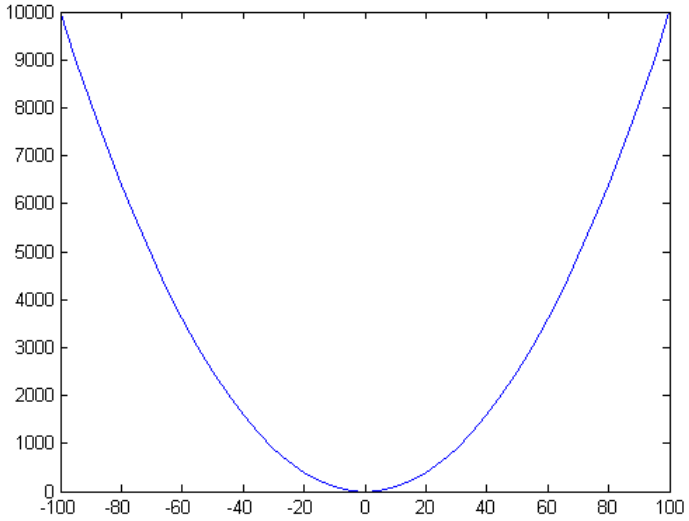

FIGURE 1.3 – Y=x

Exemple 1 : La fonction *sinus* :

```
>> fplot('sin',[0,2*pi])
```

La figure 1.4 montre le graphe correspondant, les valeur de y_{min}, y_{max} sont pris comme $[0, 1]$.

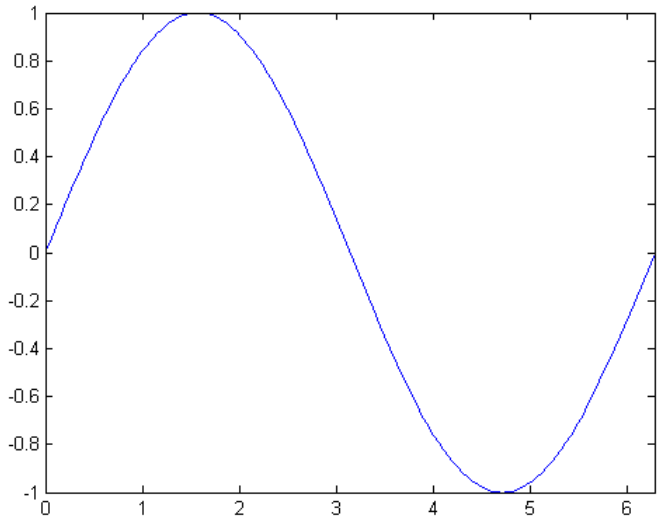

FIGURE 1.4 – Y=sin(x)

Exemple 2 : La fonction : $x.sin(x)$, voir Figure 1.5

```
>> fplot('x*sin(x)',[-2*pi,2*pi,-5,5])
```

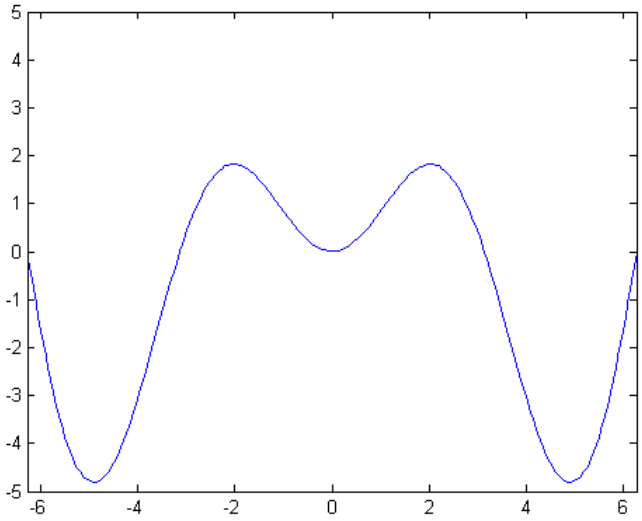

FIGURE 1.5 – Y=x.sin(x)

1.8.2 Les titres sur les graphes

Soit la courbe de la fonction

```
>> x =[0:0.01:10]; y = sin(x);
>> plot(x, y),
```

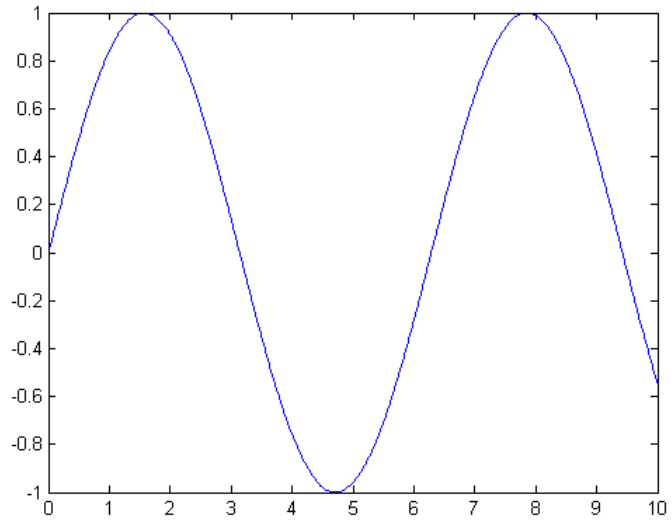

FIGURE 1.6 – Y=sin(x)

Pour ajouter les étiquetes et le titre.

```
>> xlabel('x'), ylabel('Sin(x)'), title('Graph de Sin(x) '),
grid on, axis equal
```

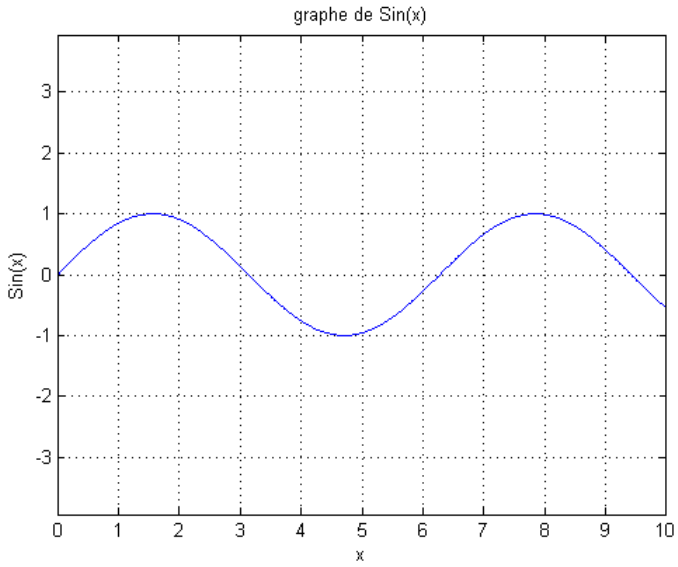

FIGURE 1.7 – Grapbe de : Y=sin(x)

1.8.3 Tracer plusieurs fonctions sur le même graphe

Un graphe peut être utiliser pour représenter plusieurs fonction :

Exemple :

```
x =[0:0.01:10];
y = sin(x);
g = cos(x);
plot(x, y, x, g)
legend('Sin(x)','Cos(x)')
```

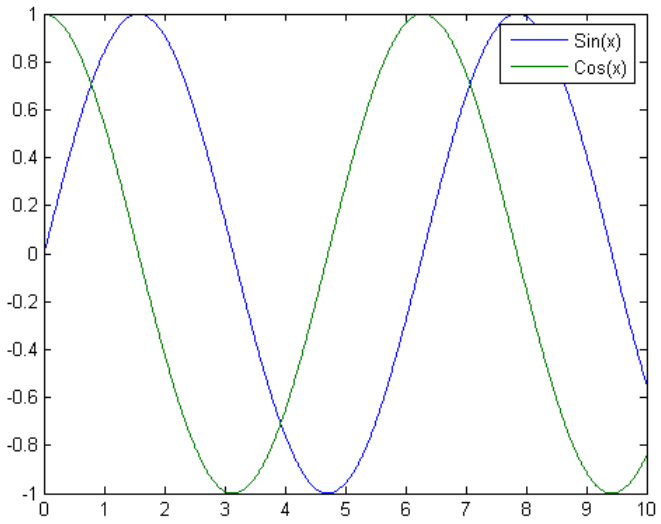

FIGURE 1.8 – Graphe de : sinus et cosinus

On peut personnaliser la couleur des courbes, exemple on mis le sinus en rouge `red` et le cosinus en blue `bl` :

```
figure,plot(x, y,'r', x, g,'b')
```

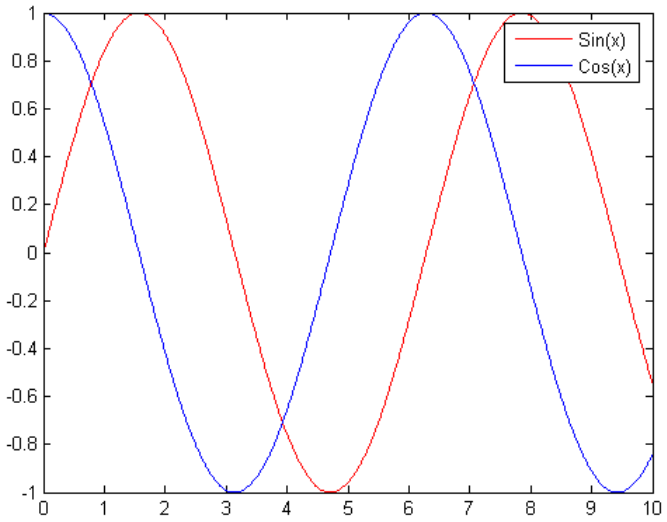

FIGURE 1.9 – Graphe de : sinus et cosinus

1.8.4 Options de couleur et style des graphiques

Le tableau suivant résume les codes de couleurs de MATLAB(Valeur par défaut : 'b-.' = bleu, trait plein, point) :

Couleur	code	Style	code	Symbole	code
Blanc	w	trait plein	-	point	.
Noir	k	pointillé court	:	Cercles	o
Blue	b	pointillé long	–	croix	x
Rouge	r	pointillé mixte	-.	plus	+
Cyan	c	pas de ligne	none	étoile	*
Vert	g			carré	s
Magenta	m			losange	d
Jaune	y			triangle (bas)	v
				triangle (gauche)	<
				triangle (droite)	>
				pentagone	p
				hexagone	h
				aucun	none

Exemple :

```
A=[1980 1985 1990 1995 2000 2005]; P=[17 18 25 27 31 34];
subplot(1,3,1)
plot(A,P)% par défaut
```

```
subplot(1,3,2)
plot(A,P,'k')% couleur(noir)
subplot(1,3,3)
plot(A,P,'r*')%couleur(rouge) + Symbole en chaque point (xi,yi) (étoile)
subplot(1,3,1)
plot(A,P,'b*:')% couleur+Symbole en chaque point (xi,yi) + Style de trait(pointillé co
```

1.8.5 Plages de valeurs des axes

On peut paramétrer les valeurs des axes en donnant les valeurs minimum et maximum.

```
axis ([xmin xmax ymin ymax])
```

Exemple :

```
x =[0:0.01:10];
y = exp(-x).* sin(2*x +3);
plot(x, y), axis([0 10-1 1])
grid on
```

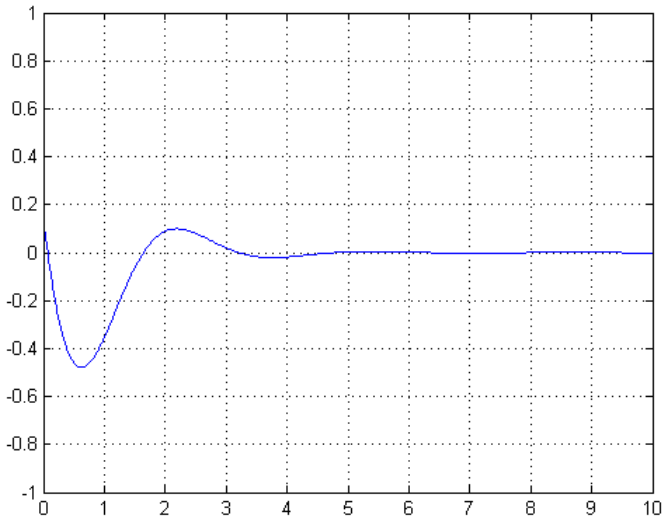

FIGURE 1.10 – Axes prédéfinies

1.8.6 Sous Graphes

On peut mettre plusieurs sous graphes fonctions sur le même graphe, pour comparer plusieurs phénomènes par exemple :

```
x =[0:0.01:5];
y = exp(-1.5*x).*sin(10*x);
subplot(1,2,1)
plot(x,y),
```

```
xlabel('x'),ylabel('exp(-1.5x)*sin(10x)'),axis([0 5 -1 1])
y = exp(-2*x).*sin(10*x);
subplot(1,2,2)
plot(x,y),xlabel('x'),ylabel('exp(- 2x)*sin(10x)'),axis([0 5 -1 1])
```

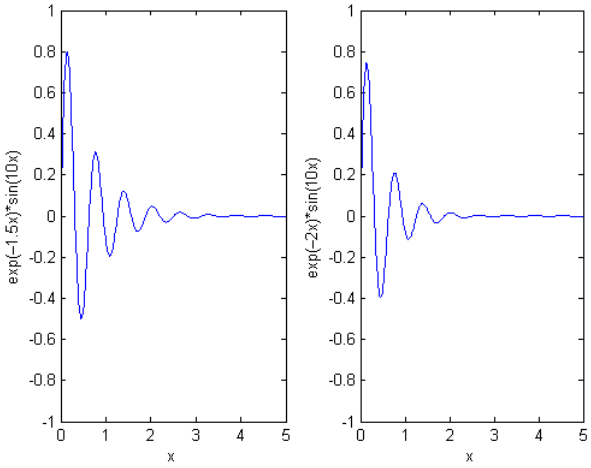

FIGURE 1.11 – Sous graphes

Exemple 2 :

```
subplot(2,2,1)
fplot('cos',[pi -pi]),title('fonction cosinus')
subplot(2,2,2)
fplot('sin',[pi -pi]),title('fonction sinus')
subplot(2,2,3)
fplot('-3*x^3+1',[-1 +1 -2.5 3]),title('fonction -3x3+1')
subplot(2,2,4)
fplot('exp',[pi -pi]),title('fonction exp')
```

1.8.7 Les histogrammes

Un histogramme est un graphique permettant de représenter la répartition d'une variable continue en la représentant avec des colonnes verticales.

Exemple : supposant les notes de 10 étudiants représenter dans le vecteur x comme suit :

```
x =[1:10];
y =[15  11.5  18  17.5  10  17  18.5  15  12  19];
bar(x,y)
```

Le fonction `pie(x)`

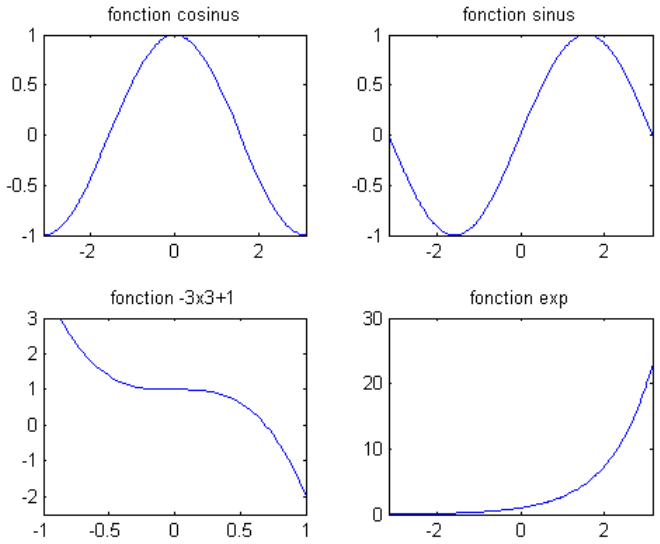

FIGURE 1.12 – Sous graphes(Exemple 2)

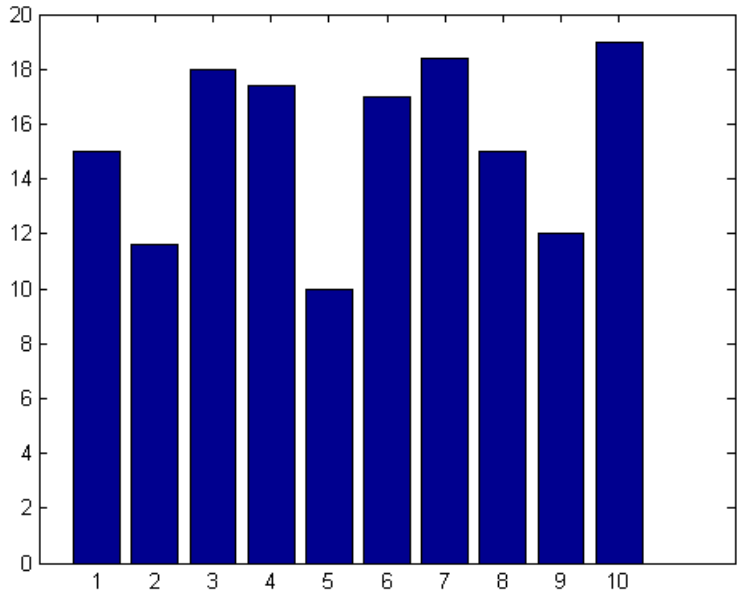

FIGURE 1.13 – Histogramme

Exemple 2 :

```
SurveyData = [8, 7, 6; 13, 21, 15; 32, 27, 32];
bar(SurveyData)
```

1.8.8 Graphe 3D

Exemple : La fonction $f(x, y) = x.e^{-x^2-y^2}$ dans l'intervalle 3D , [-2,2,-2,2].

```
>> [X,Y] = meshgrid(-2:0.2:2,-2:0.2:2);
```

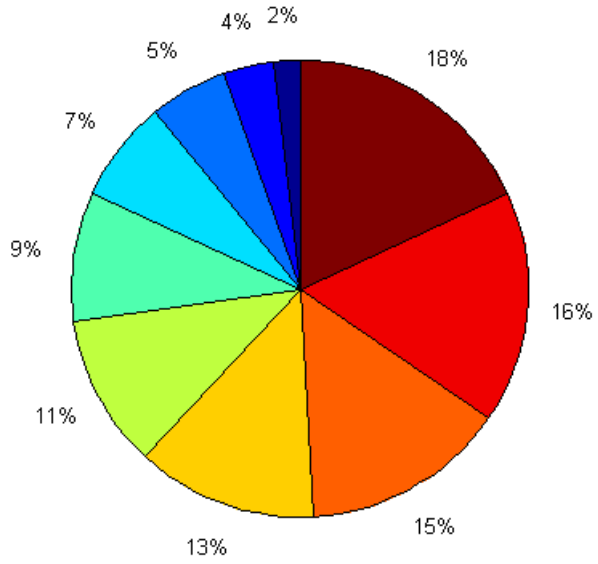

FIGURE 1.14 – Proportions

```
>> Z = X.*exp(-X.^2-Y.^2);
>> mesh(X,Y,Z)
```

Voir Figure 1.15

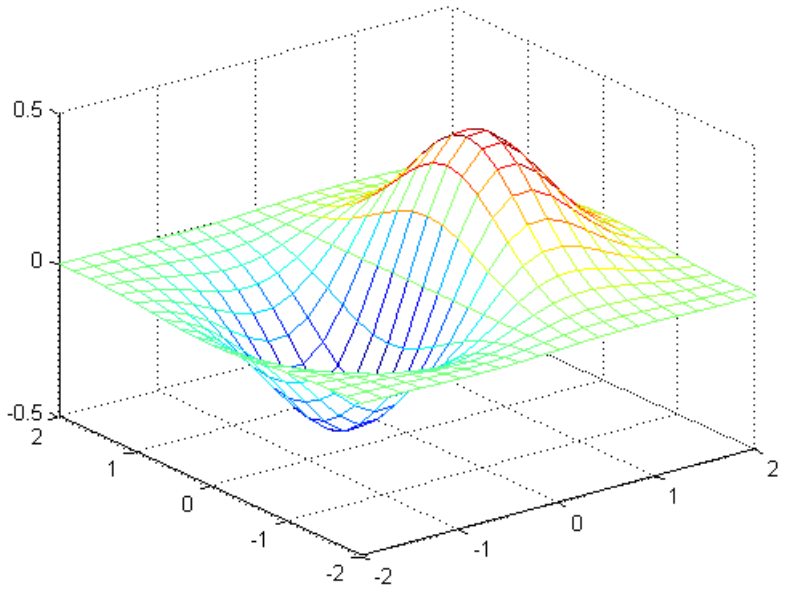

FIGURE 1.15 – Le fonction $f(x, y) = x.e^{-x^2-y^2}$ en 3D

Exemple 2 :

```
x=0:1/1000:10;y=-sin(x);
z=cos(x).^2;
plot3(x,y,z)
xlabel('x');ylabel('y');zlabel('z');
```

1.9 Exercices

1.9.1 Exercice : Construction de matrice tridiagonale

1. Construire la matrice T tridiagonale suivante :

$$T = \begin{pmatrix} 1 & 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix}$$

2. Extraire de T les deux premières colonnes.
3. Extraire de T les éléments des colonnes et des lignes 2 à 4.
4. Créer une matrice T2 où la ligne 1 est échangée avec la ligne 3 puis la colonne 2 est remplacée par les valeurs de la colonne 4.

1.9.2 Exercice : Création de différents vecteurs

Donnez le code MATLAB qui permet de :

1. Créez un vecteur colonne `vec` de 5 éléments linéairement espacés entre 2 et 3.
2. Ajoutez deux lignes à la fin de ce vecteur avec la valeur 0.
3. Ajoutez 1 au deuxième et sixième éléments de ce vecteur.
4. Créez un second vecteur `vec2` colonne de même dimension que `vec` contenant les entiers pairs supérieurs ou égaux à 6.
5. Définir un vecteur `SumVec` comme la somme des deux vecteurs `vec` et `vec2`.
6. Définir un vecteur `ProdVec` comme le produit termes à termes des deux vecteurs `vec` et `vec2`.
7. Quel est la somme des éléments de `Prodvec` ?
8. Quel est le plus grand élément du vecteur `Prodvec` ?

1.9.3 Exercice : Création de matrices en ligne de commande

Créer les matrices suivantes avec la ligne de commande :

$$A = \begin{pmatrix} 5 & 2 & 0 \\ 3 & 4 & 1 \end{pmatrix}, B = \begin{pmatrix} 3 & 3 \\ 3 & 3 \\ 3 & 3 \end{pmatrix}, C = \begin{pmatrix} 5 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 5 \end{pmatrix}, D = \begin{pmatrix} e^1 & 0 & 0 & 0 \\ 0 & e^2 & 0 & 0 \\ 0 & 0 & e^3 & 0 \\ 0 & 0 & 0 & e^4 \end{pmatrix}$$

$$E = \begin{pmatrix} 1 & 2 & ... & 10 \\ 11 & 12 & ... & 20 \end{pmatrix}, F = \begin{pmatrix} 1 & 2 & ... & 10 \\ 10 & 20 & ... & 100 \\ 100 & 200 & ... & 1000 \end{pmatrix}$$

1.9.4 Exercice : Test sur Matrices

Écrire un script qui permet de lire une matrice saisie par l'utilisateur et l'informe si elle est carrée.

1.9.5 Exercice : Calcul de fonctions

Écrire un script **MATLAB** qui calcul y1 et y2 en fonction de x1 et x2.

$$y1 = \begin{cases} \frac{1-cos(2x_1)}{\sqrt{1+4x_1^2-4}} & ; x_1 < 0 \\ (1-\frac{x_1}{2})^{\frac{2}{x_1}} & ; x_1 \in]0,1] \\ \frac{sin(\pi x_1)}{1-x_1} & ; x_1 > 1 \end{cases}$$

$$y2 = \begin{cases} cos(2x^2+1) & ; |x_2| > 2 \\ \sqrt{x_2^2+2|x_2|+2} & ; |x_2| \leq 2 \end{cases}$$

1.9.6 Exercice : Calcul de factoriel

Écrire un script **MATLAB** qui calcul Y en fonction de x.

$$Y = \frac{1!}{1-x} + \frac{2!}{1+x^2} + \frac{3!}{1-x^3} + ...$$

Remarque : Pour calculer le factoriel de n, on utilise la fonction factorial de **MATLAB**.

1.9.7 Exercice : Fonctionnalités graphiques

Utilisant les fonctionnalités graphiques de MATLAB, Tracer les courbes suivantes (Utiliser `plot` puis `fplot`) :

1. La fonction `sin(x)` dans l'intervalle $[-\pi, \pi]$ avec un pas de $\frac{\pi}{100}$.
2. La fonction `cos(x)` dans l'intervalle $[-\pi, \pi]$ avec un pas de $\frac{\pi}{5}$.
3. La fonction `cos(x)+1` dans l'intervalle $[-\pi, \pi]$ avec un pas de π.

Pour chaque graphe :
- Créer le quadrillage
- Créer les titres sur le graphe.
- Uniformiser la taille des axes.