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

TP Matlab

RAHAB Hichem

rahab_hichem @yahoo.fr

rahab.e-monsite.com

2017 /2018

Chapitre 1

Introduction au langage MATLAB

1.1 Rappel sur les langages de calculs scientifique

Un langage de calcul scientifique est un langage de programmation destiné à être utilisé par la communité scientifique dans des calculs scientifiques complexes. Et à cette finalité, il est riche en terme fonctions et de bibliothèques facilitant la tâche d'un programmeur dans un domaine de recherche, qui n'a pas nécessairement des compétences de programmation anvancées. On peut classer les langages de calculs scientifiques en langages compilé et langages interprité. Dans un langage de programmation interprété un programme supplémentaire (l'interpréteur) est nécessaire, celui-ci va générer l'exécutable des instructions et les exécuter au fur et à mesur de l'exéécution du programme, donc on n'a pas dans ce cas un code exécutable complet, et à chaque fois on a besoin du code source initiale pour reéxécuter le programme. Par contre un langage compilé va traduire (compilé) le programme en son intégralité vers un code exécutable qui peut être utilisé ultériorment sans avoir besoin du code source initiale.

1.1.1 Langages compilés

On peut citer à titre d'explication quelques langages comiplés :

1. Fortran (FORmula TRANslator) est un langage compilé developpé pr IBM vers 1954, est le plus ancien langage de programmation de haut niveau, c'est un langage de programmation destiné principalement pour le calcul scientifique.
2. Langage C
3. Langage C++
4. etc ...

1.1.2 Langages interprétés

Il y a aussi une varité de langages interprétés dont :

1. Matlab C'est le langage qu'on va étudier le long de ce cours.
2. Le langage R est un langage interprété trés adaptée au calcul scientifique et à la fouille de données. Il possède une large collection d'outils statistiques et graphiques, relayée par une communautée très active. R est à la fois un logiciel de statistique et un langage de programmation. R est un logiciel de traitement statistique des données.
3. Scilab (Scientific Laboratory) est un logiciel libre de calcul numérique multi-plateforme fournissant un environnement de calcul pour des applications scientifiques. Il possède un langage de programmation orienté calcul numérique de haut niveau. Il peut être utilisé pour le traitement du signal, l ?analyse statistique, le traitement d ?images, la modélisation et la simulation. [1] Scilab est disponible pour Windows, Mac OS X, GNU/Linux. La syntaxe

1. https ://fr.wikipedia.org/wiki/Scilab

et les possibilités offertes par Scilab sont similaires à celles de Matlab.

4. etc ...

1.2 Le langage Matlab

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 $n \times 1$. 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.

MATLAB est 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, ...).
On peut utiliser Matlab en deux modes :

Mode ligne de commande

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

Mode script

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. Le mode script ou le mode programmation, quant à lui, permet de développer des applications plus complexes, ainsi que les programmes sont sauvegarder pour faciliter une utilisation ultérieur.

1.3 Installation de Matlab

Bien qu'il y a plusieurs versions de Matlab, on a trouvé que la version 7.0 et la plus convenable pour notre TP, premièrment la taille du programme, moins de 700 Mo, est très acceptable pour la copie et l'installation, aussi son interface est basique et ça ce qu'on veux pour un cours d'initiation au langage. J'ai trouvé utile de présenter en premier l'installation du langage que je recommande à mes étudiants de faire sur leurs ordinateurs personnels.

Commençant avec la modification du thème vers "Windows classique" avec lequel travail cette version.

1. clique droit sur le bureau et choisir Personnaliser, voir Figure 1.1.
2. Choisir le thème Windows Classique, voir Figure 1.2.

Ensuite aller dans le répertoire Matlab7 . Et cliquer sur setup.exe (Figure 1.3)

Dans la Figure 1.4 L'installation commence :

Dans la Figure 1.5 on doit saisir les coordonnées de l'utilisateur et le numéro de série du produit :

Alors comme illustrer dans la Figure 1.6 on va dans le dossier Crack, eton lance le keygen comme dans Figure 1.7 :

Copier le numéro de série voir Figure 1.8

Coller le numéro de série dans l'enmplacement spécifier, Figure 1.9

Accepter le termes de la licence pour continuer 1.10

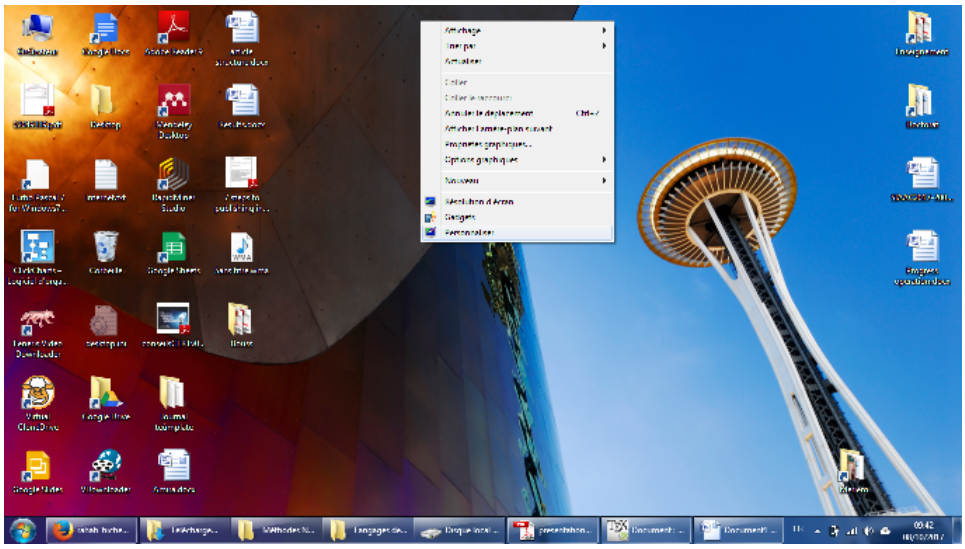


FIGURE 1.1 – Modification de thème windows

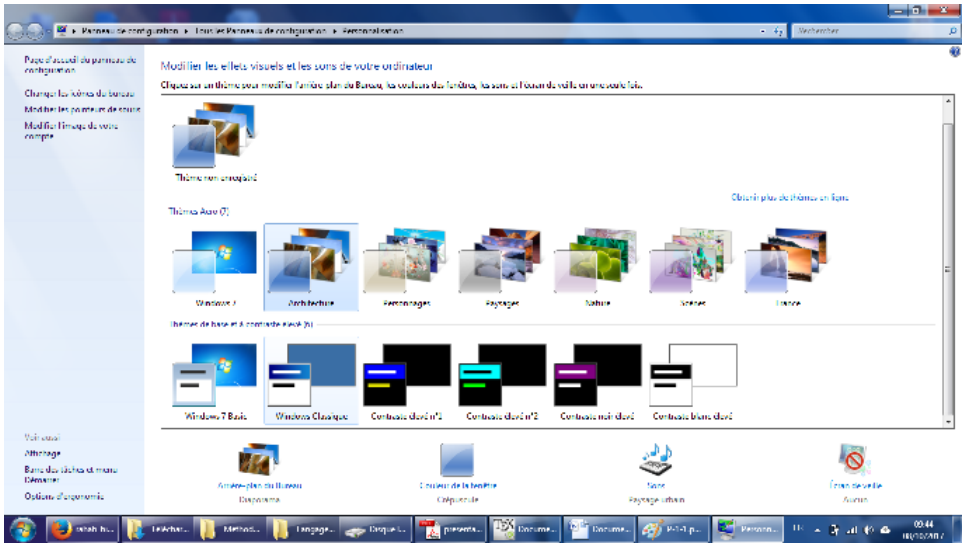


FIGURE 1.2 – Choisir le thème Windows Classique

Choisir installation typique, Figure 1.11

Choisir l'emplacement de l'installation, Figure 1.12

Dans la Figure 1.13 confirmer les options d'installation choisies. Et cliquer Instal.
Dans la Figure 1.15 le CD-2 est demandé alors qu'il n'est pas disponible, donc on clique sur `Skip CD-2`.
Et on poursuit dans Figure 1.16.
L'installation se termine en Figure **??**.
et le programme Matlab est lancer comme illustrer dans Figure 1.17.

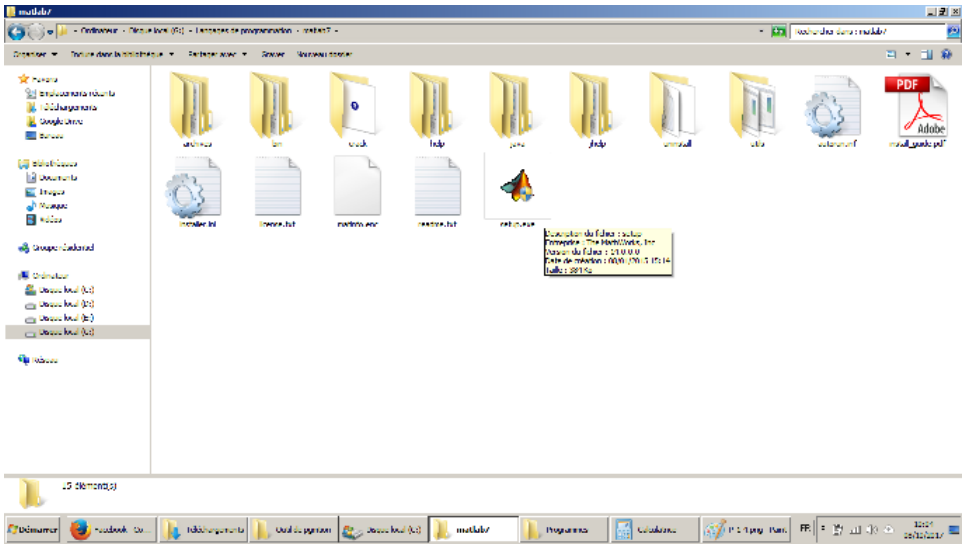


FIGURE 1.3 – Cliquer sur setup.exe

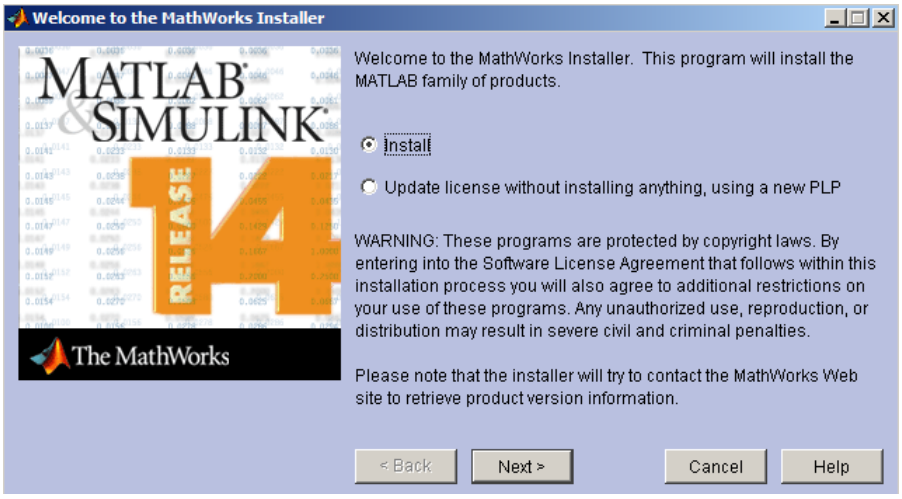


FIGURE 1.4 – L'installation commence

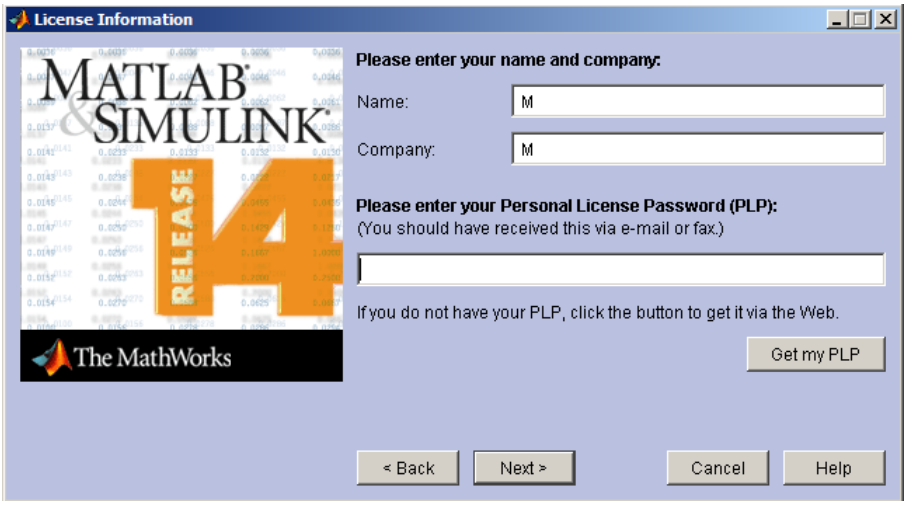


FIGURE 1.5 – Demande de numérod de série

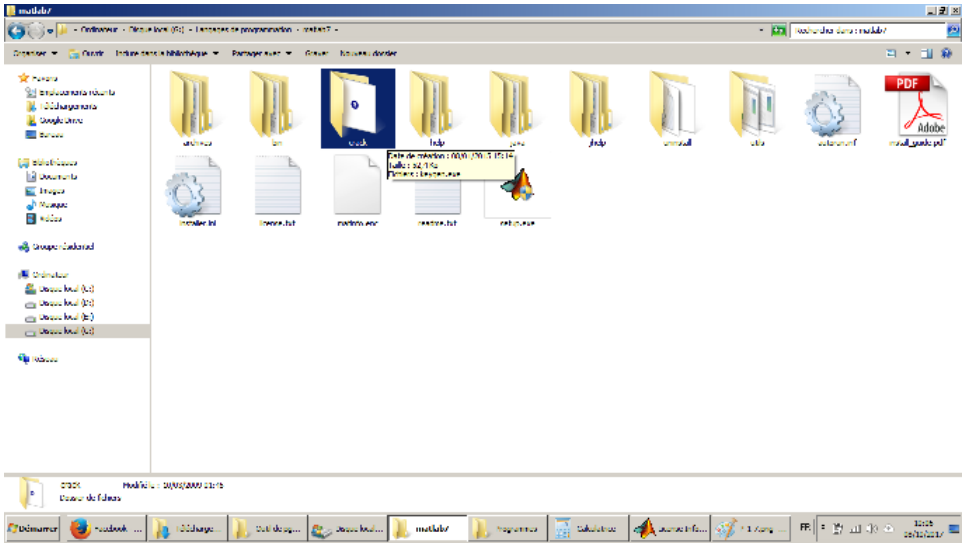


FIGURE 1.6 – Accéder au fichier Crack

FIGURE 1.7 – Exécuter le keygen

FIGURE 1.8 – Copier le numéro de série

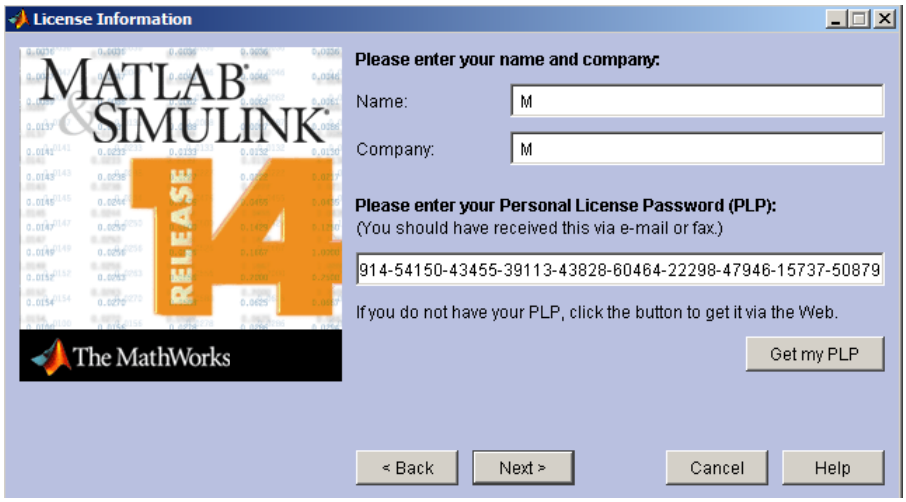


FIGURE 1.9 – Coller le numéro de série

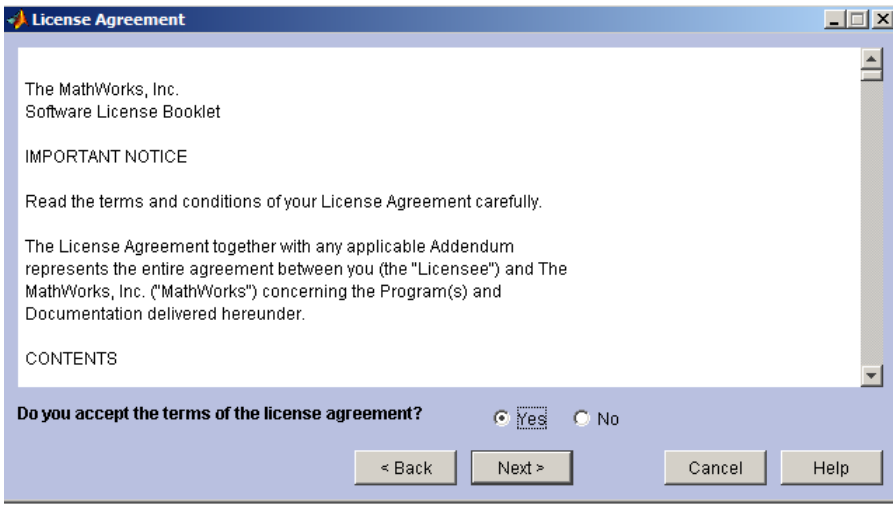


FIGURE 1.10 – Accepter le termes de la licence

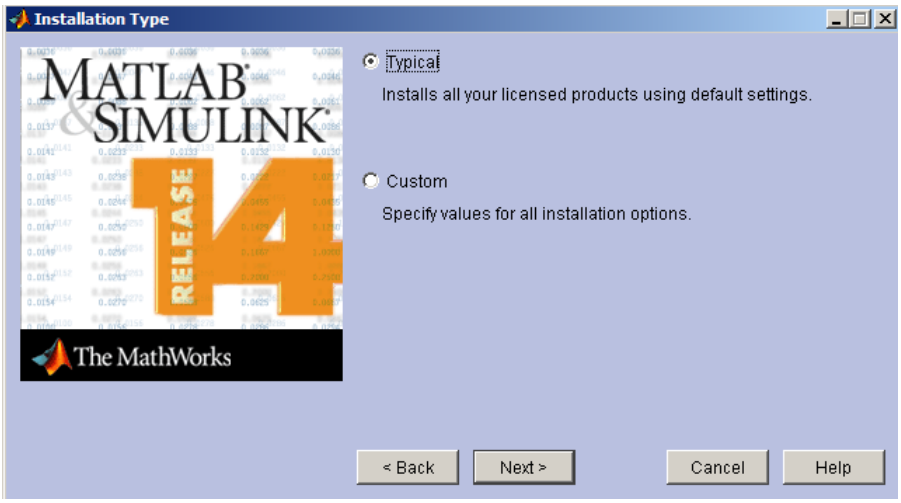


FIGURE 1.11 – Choisir installation typique

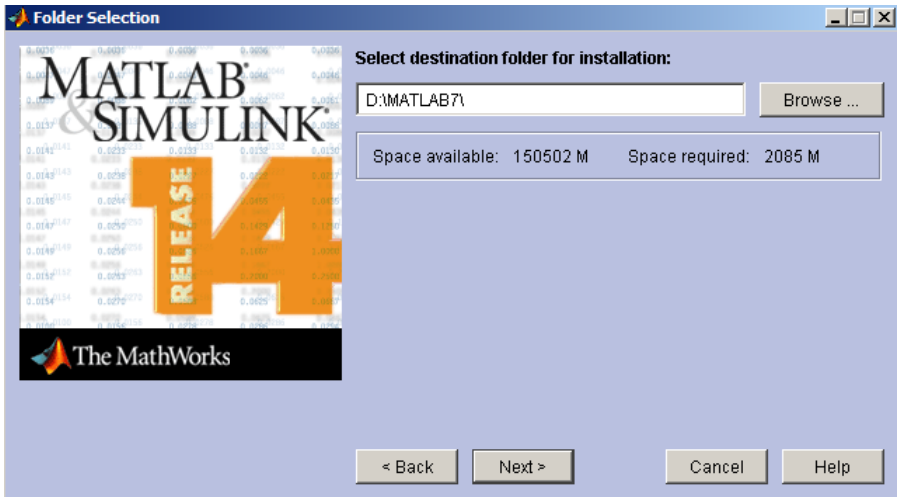


FIGURE 1.12 – Choisir l'emplacement de l'installation

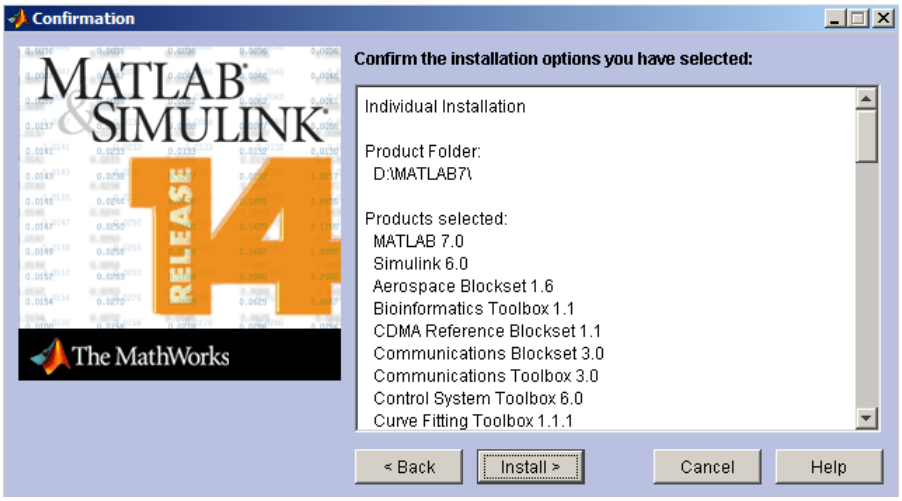


FIGURE 1.13 – Confirmer les options d'installation choisies

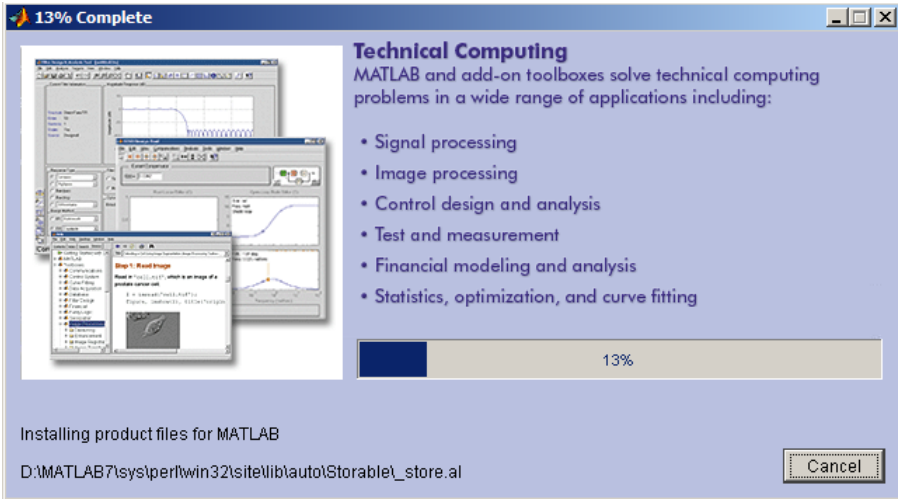


FIGURE 1.14 – Progression de l'installation

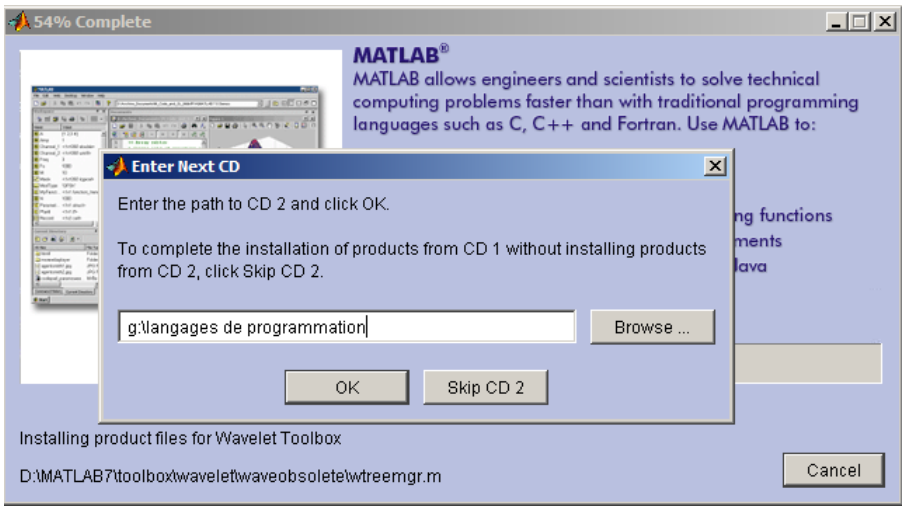


FIGURE 1.15 – Le CD-2 est demandé

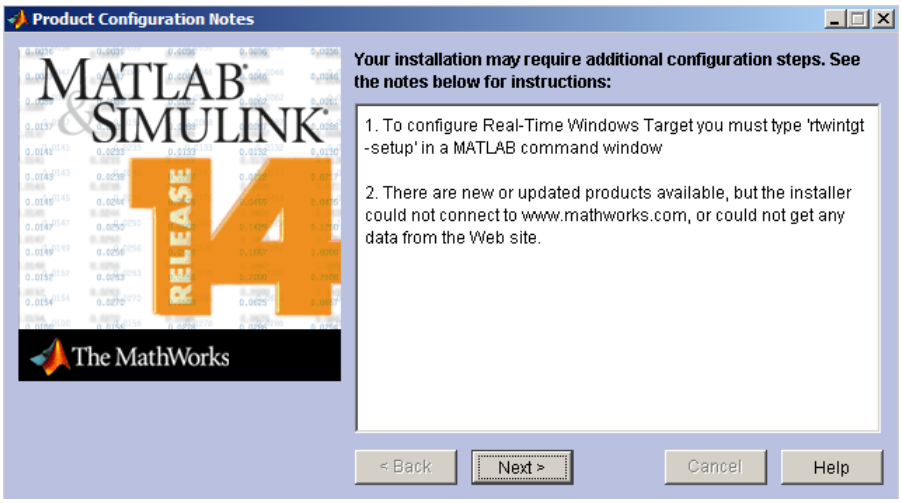
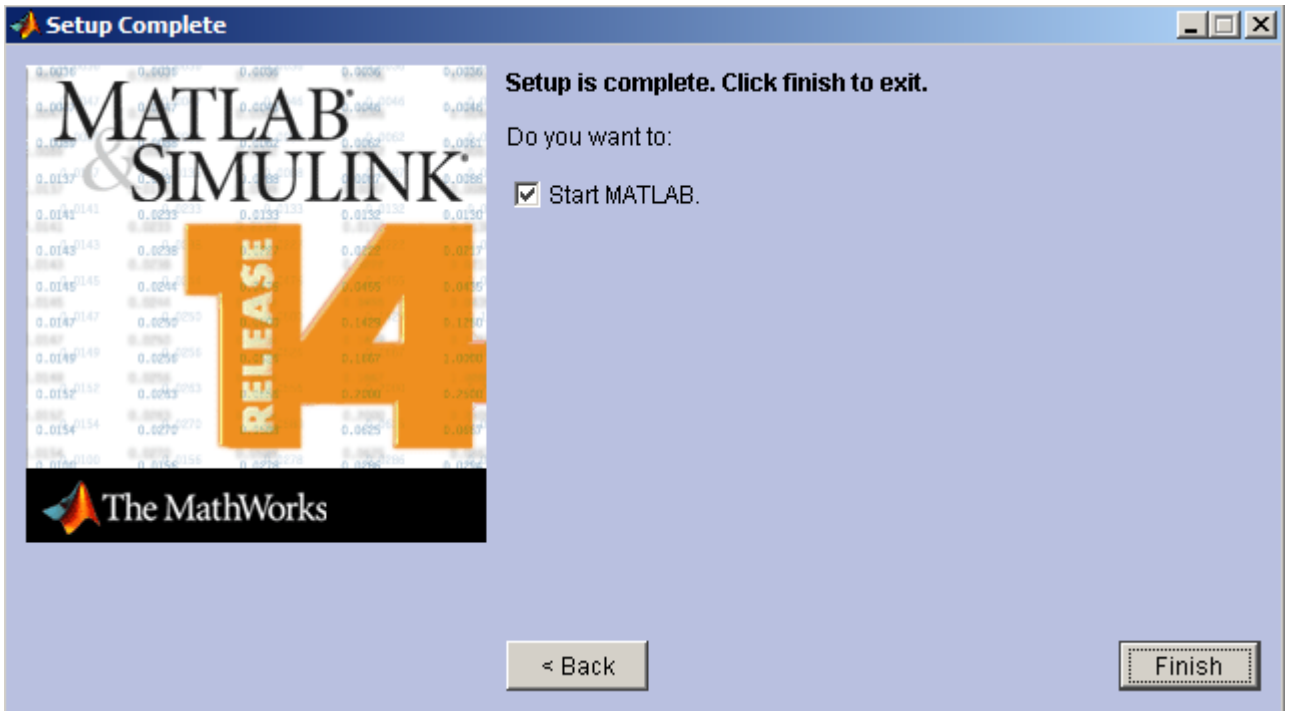


FIGURE 1.16 – Poursuivre l'installation



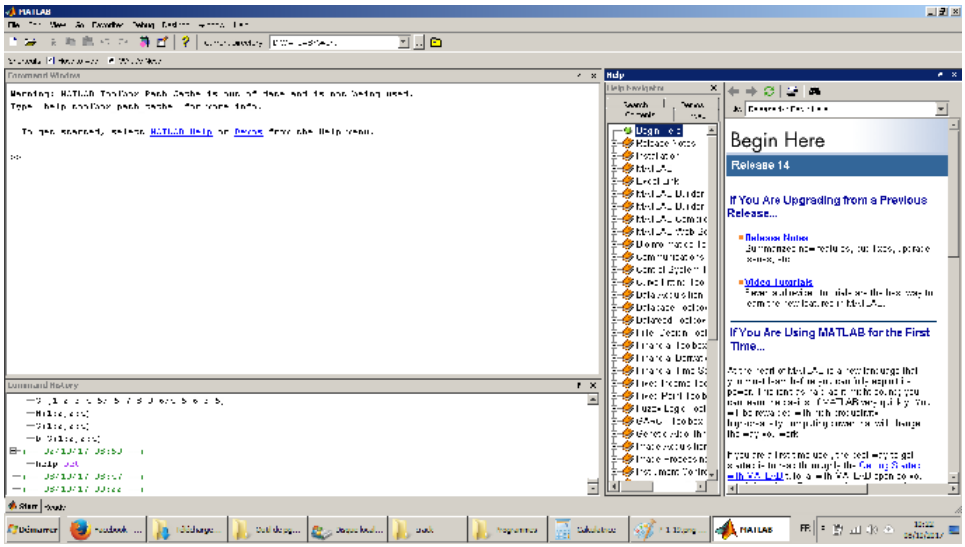


FIGURE 1.17 – Lancement du Matlab 7

1.4 Les commandes Matlab

Matlab est très riche en nombre et qualité des commandes qu'il offre, on peut donner à titre d'illustration les commandes suivantes :

- `help` : on utilise cet commande pour obtenir l'aide sur une méthode donnée.
- `clock` : affiche l'année, le mois, le jour, l'heure, les minutes et les secondes.
- `Date` : Affiche la date.
- `input` : permet de lire une valeur à partir du clavier (l'instruction habituelle lire). Exemple : (X = input ('taper un nombre : ')).
- `disp` : permet d'afficher un tableau de valeurs numériques ou de caractères. L'autre façon d'afficher un tableau est de taper son nom. La commande disp se contente d'afficher le tableau sans écrire le nom de la variable, ce qui peut améliorer certaines présentations. On utilise fréquemment la commande disp avec un tableau qui est une chaîne de caractères pour afficher un message. Exemple : `» disp('la valeurs saisie est erronée')`.
- clear : permet de détruire une variable de l'espace de travail (si aucune n'est spécifiée, toutes les variables seront effacées).

- who : donne la liste des variables définies dans l'espace de travail actuel (essayer whos).
- `clc` : effacer le contenu de la fenêtre des commandes et affiche uniquement l'invite « » »

1.5 Quelques fonctions

On plus des commandes, Matlab comme tout langage de calcul scientifique et riche avec son bibliothèques de fonction mathématiques, ces fonctions sont offertes pour implémenté des calculs puissant avec un minimun de code. alors il n'est pass demandé à un tout utilisateur de Matlab de maitraiser l'implémentaion algorithméque de toutes les fonction mathématique (à savoir, sinus, cosinus, expenentielle, ...).

1. `exp(x)` : exponentielle de x.
   Exemple :

```
>> x=5
>>exp(x)
ans =
  148.4132
```

2. `log(x)` : logarithme néperien de x.
   Exemple :

```
>> log(1)
ans =
     0
```

3. `log10(x)` : logarithme en base 10 de x. **Exemple :**

```
>> log10(2)
ans =
    0.3010
```

4. `x^n` : x à la puissance n.
5. `sqrt(x)` : racine carrée de x.
6. `abs(x)` : valeur absolue de x.
7. `sign(x)` : 1 si x > 0 et -1 si x < 0, et 0 si x=0.
8. `sin(x)` :sinus de x
9. `cos(x)` : cosinus de x

10. `tan(x)` : tangente de x
11. `round(x)`: Affiche l'arrondi d'un nombre x.
12. `floor(x)`: Arrondissement vers $-\infty$.
    Exemple :

```
>> floor(5.2)
ans =
     5
>> floor(5.7)
ans =
     5
>> floor(-5.7)
ans =
    -6
```

13. `ceil(x)` : fait exaactement l'inverse de `floor(x)`:
    Exemple :

```
>> ceil(-5.7)
ans =
    -5
>> ceil(-5.2)
ans =
    -5
>> ceil(5.2)
ans =
     6
```

14. `rem(m,n)`: reste de la division entière de m par n.
15. `lcm(m,n)`: plus petit commun multiple de m et n.
16. `gcd(m,n)`: plus grand commun diviseur de m et n.
17. `factor(n)`: d'ecomposition en facteurs premiers de n.

1.6 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

Chapitre 2

Vecteurs et matrices

Matlab est un langage matriciel, de ce fait comprendre le principe des matrices et son manipulation est primordiale pour travailler efficacement sur Matlab.

2.1 Création de matrices

Pour créer les matrices, on a plusieurs façon.

Matrice 1×1 (scalaire)

```
>> x=4
x =
     4
```

Matrice 1×4 (vecteur ligne)

On peut séparer les élément avec un espace ou une vergule :

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

Ou bien :

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

     x =

     1     2     3     4
```

Matrice 4×1(vecteur colonne)

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

Vecteur ligne des éléments de 1 à 10

```
>> x=[1:10]
x =
     1     2     3     4     5     6     7     8     9    10
```

Vecteur ligne des éléments de 0 à 10 avec un pas de 2

```
>> x=[0:2:10]
x =
     0     2     4     6     8    10
```

2.1.1 Vecteur à éléments linéairement espacés

Dans cet exemple la commande `linspace` permet de créer un vecteur de 5 éléments entre 2 et 3.

```
>> y=linspace(2,3,5)
y =
    2.0000    2.2500    2.5000    2.7500    3.0000
```

Matrice de 3×3

```
>> 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.

2.2 Transposé 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 :

$$y = \begin{matrix} 0 & 4 \\ 1 & 5 \\ 2 & 6 \end{matrix}$$

2.3 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.
S'il s'agit d'un vecteur on peut utiliser `size` comme on peut utiliser tout simplement `length`.

```
>> y=linspace(2,3,5)
y =
    2.0000    2.2500    2.5000    2.7500    3.0000
>> length(y)
ans =
     5
```

2.4 Sélection des élémnents d'une matrice

2.4.1 Sélection de lignes et/ou de colonnes

La colonne j de la matrice x est donnée par : `y(:,j)`. La ligne i de la matrice x est donnée par : `y(i,:)`

Exemple : Soit la matrice `y`.

```
>> y=[0 1 2;4 5 6]'
y =
     0     4
     1     5
     2     6
```

Pour la colonne j=2, on a :

```
 y(:,2)=
  4
  5
  6
```

Pour la ligne i=2, on a :

```
y(2,:)=
   1 5
```

2.4.2 Selection des éléments du diagonale

```
>> 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
```

2.5 Concaténation de matrices

La concaténation consiste à joindre 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, soit par des points-virgules pour une concaténation verticale, ou bien par des virgules ou des espaces pour une concaténation horizontale.

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
```

2.6 La matrice Identité

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

Exemple : pour n=3, on a :

```
>> eye(3)
ans =
     1     0     0
     0     1     0
     0     0     1
```

2.7 Opérations sur les matrices

2.7.1 La somme

La somme de deux matrices A et B de même dimension est une matrice S de même dimension que A et B, et dont les élément sont la somme terme à terme des deux matrices.

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

>> S=A+B
S =
     7     7     7
     7     7     7
```

2.7.2 La soustraction

La différence de deux matrices A et B de même dimension est une matrice D de même dimension que A et B dont les élément sont la différence terme à terme des deux matrices.

Exemple : Soit les deux matrices A et B précédentes :

```
>> D=A-B
D =
    -5    -3    -1
     1     3     5
```

2.7.3 Le produit

Produit matricielle

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

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

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 =
    20    14
    56    41
```

Produit terme à terme

Dans ce cas le produit $A. * B$ n'est possible que si les deux matrices sont de mêmes dimensions, :

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
```

2.7.4 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 :

```
>> 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
```

2.7.5 La division

Division à droite

La division de A sur B est le produit de A par l'inverse de B :

$$A/B = A \times B^{-1}$$

Exemple

```
>> A=[1 2 3; 4 5 6; 7 8 1 ];
>> b=[ 7 5 6 ; 2  0  8; 5 7 1];
>> A/b
ans =
   -0.5254    0.6864    0.6610
   -0.4237    0.9407    1.0169
    0.6271   -0.4322    0.6949

>> A*inv(b)
ans =

   -0.5254    0.6864    0.6610
   -0.4237    0.9407    1.0169
    0.6271   -0.4322    0.6949
```

Division à gauche

la division à gauche que l'on note sous Matlab A b cela signifie `inv(A)*b`.

Exemple

```
>> A\b
ans =
  -11.3333   -9.8333   -3.5417
   10.6667    9.6667    3.0833
   -1.0000   -1.5000    1.1250

>> inv(A)*b
ans =

  -11.3333   -9.8333   -3.5417
   10.6667    9.6667    3.0833
   -1.0000   -1.5000    1.1250
```

2.7.6 Autres opérations

Soit `x=[2 15 0]` un vecteur ligne (matrice $1 \times n$).

sort(x) Mettre dans l'ordre croissant les éléments d'un vecteur ligne :

```
>>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
```

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

```
>>max(x)
ans =
15
```

fliplr(x) Inverse l'ordre des éléments d'une matrice.

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

>> fliplr(x)
ans =
     3     2     1
     6     5     4
```

rand(m,n) : donne une matrice de dimensions $m \times n$ de valeurs aléatoires.

Exemple :

```
>> rand(3,3)
ans =

    0.9501    0.4860    0.4565
    0.2311    0.8913    0.0185
    0.6068    0.7621    0.8214

>> rand(3,3)
ans =

    0.4447    0.9218    0.4057
    0.6154    0.7382    0.9355
    0.7919    0.1763    0.9169
```

diag(x) permet de créer une matrice carrée dont le diagonal est le vecteur x.

Exemple :

```
>>D=diag(1:4)
D =
     1     0     0     0
     0     2     0     0
     0     0     3     0
     0     0     0     4
```

Chapitre 3

Eléments de programmation

3.1 Les structures conditionnelles

3.1.1 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 satisfaite.

Le test `if` a la forme générale suivante : `if-elseif-else-end`.

```
if <condition 1>
 <instruction 1.1>
 <instruction 1.2>
 ...
elseif <condition 2>
 <instruction 2.1>
 <instruction 2.2>
 ...
...
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 entraîne l'exécution de l'instruction correspondante.

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

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

Exemple 1

```
>> V=268.0826;
>> R=4;
>> if V>150, surface=pi*R^2, end
```

On a la valeur de V >150 (268.0826>150) Alors l'exécution on a :

```
surface =
         50.2655
```

Exemple 2 Pour calculer les racines d'un trinôme de second degré, $ax^2 + bx + c$, on met 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 commande disp(' ') affiche simplement ce qui est écrit entre crochets.
- La double apostrophe sert à représenter une apostrophe dans une chaîne de caractères. Ceci est nécessaire car une simple apostrophe est une commande Matlab.

3.1.2 La clause 'switch'

Une clause `switch` exécute un block d'instructions parmi plusieurs choix si sa condition est satisfaite. A chaque choix est attribué un `case`.

```
switch < switch expression>
  case <case expression>
       <statements>
  case <case expression>
       <statements>
          ...
          ...
  otherwise
     <statements>
 end
```

Exemple :

```
Mention = input('Donner la Mention:');
switch(grade)
    case 'A'
        disp('Excellent' );
    case 'B'
        disp('Trés bien\n' );
    case 'C'
        disp('Bien\n' );
    case 'D'
        disp('Passable\n' );
    case 'E'
        disp('Essaye encore\n' );
    otherwise
        disp('Invalid grade\n' );
end
```

Imbrication de switch

Une switch peut faire partie d'autres switch :

```
switch(ch1)
    case 'A' fprintf('Cet A appartient à switch extérieur');
        switch(ch2)
            case 'A' fprintf('Cet A appartient à switch intérieur' );
            case 'B' fprintf('Cet B appartient à switch intérieur' );
        end
    case 'B' fprintf('Cet B appartient à switch extérieur');
end
```

Exemple : Créer un script avec le code suivant :

```
a = 100;
b = 200;
switch(a)
    case 10
        fprintf('This is part of outer switch %d\n', a );
    switch(b)
        case 200
            fprintf('This is part of inner switch %d\n', a );
    end
end
fprintf('Exact value of a is : %d\n', a );
fprintf('Exact value of b is : %d\n', b );
```

L'execution de ce programme donne :

```
This is part of outer switch 100
This is part of inner switch 100
Exact value of a is : 100
Exact value of b is : 200
```

3.2 Les boucles

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

Remarque
- 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 tapé `end`.

3.2.2 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
```

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

3.3 Les scripts

Un nouveau programme Matlab doit être placé dans un fichier, appelé m-fichier, dont le nom comporte l'extension .m . Les programmes Matlab peuvent êtres des scripts ou des fonctions. Un script est simplement une collection de commandes Matlab, placée dans un m-fichier et pouvant êtres utilisée interactivement. Un script n'accepte pas des entrée, et ne retourn aucune sortie. Les fonctions sont aussi stockées dans des m-fichiers, les fonctions peuvent recevoir des entrées et retourner des sorties. Ils peuvent aussi avoir des variables locales.

Création et exécution de scripts

Pour créer un un fichier script, on utilise l'editeur de texte de Matlab, comme on peut utiliser n'importe quel editeur de texte :
- Utilisant la ligne de commande prompt >>.
- Utilisant l'interface graphique IDE(Integrated Development Environment).

Si vous utilisez la ligne de commande, taper `edit` dans ligne de commande, l'editeur de commande de MATLAB s'ouvre. Comme on peut taper le nom de fichier directement apres `edit`.

```
>> edit
ou
>> edit <file_name>
```

La commande précédente permet de créer le script dans le répertoire MATLAB par défaut. Or, dans le cas où on veut stocker tous les programmes dans un dossier spécifier on doit donner le chemin.

Exemple : Pour créer un dossier "progs".

```
>> mkdir progs % creer un dossier progs
>> chdir progs % changer le dossier courant vers progs
>> edit prog1.m % creer un m file nommee prog1.m
```

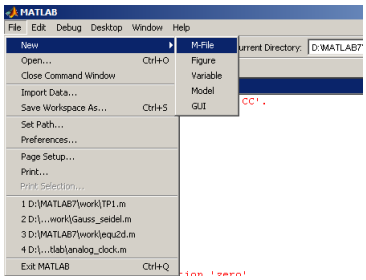


FIGURE 3.1 – Création de script avec l'interface graphique IDE

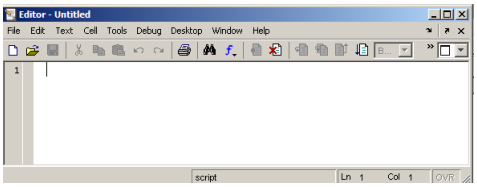


FIGURE 3.2 – Editeur de texte Matlab

Après la création la saisie des instructions et la sauvegarde du fichier, on peut l'éxécuter avec :

1. Avec l'IDE en cliquant sur le boutton `Run` dans la fenêtre de l'editeur.
2. Avec la ligne de commande prompt, en tapant le nom de fichier (sans l'extension), exemple :

```
>> tp1
```

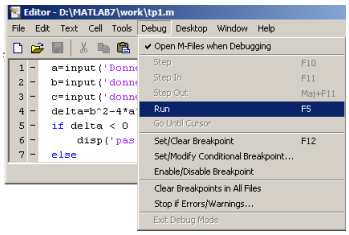


FIGURE 3.3 – Execution de script avec l'IDE

Exemple :
Pour le polynôme : $g(x) = \frac{2x^3+7x^2+3x+1}{x^2-3x+5.e^{-x}}$
On peut écrire dans un script, qu'on choisit de nommer TP, comme suit :

```
>>edit TP
```

ensuite on tape sur le script :

```
 x=3 ;
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 écrit simplement l'instruction TP après le prompt >> Matlab.

```
>> TP
g=
   502.1384
```

3.4 Les fonctions

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

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

Le nom du fichier script doit être le même nom de la fonction.

Exemple 1 On définit une fonction `determ` , qui calcule le déterminant d'une matrice carrée d'ordre 2, on va créer un fichier `determ.m`, avec le code suivant :

```
function det=determ(A)
%Cette fonction calcule le determinant
%seulement d'une matrice caree 2*2
[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 carr\'ees ');
end
return
```

La première ligne de la fonction commence avec le mot clé `function`. Ensuite on a une variable de sortie `det` et une variable d'entrée A . La ligne suivante est un commentaire, qui permet d'afficher un aide sur cette fonction, comme suit :

```
help determ
```

Et MATLAB retourne :

```
cette fonction calcule le determinant seulement d'une matrice caree 2*2
```

Pour appeler la fonction, on doit entrer en premier une matrice A avec la ligne de commande :

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

A =

     1     2
     3     4
```

Ensuite La fonction `determ` est appelée depuis la ligne de commande ainsi :

```
>> det=determ(A)

det =

    -2
```

Exemple 2 La fonction suivante permet de calculer les solutions d'une équation de 2^{eme} degrée :

```
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
end
```

Cette fonction doit etre enregistrer sur le fichier : degre2.m .
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 ...
```

3.4.1 Fonction principale et sous fonctions

Chaque fichier de fontion doit contenir une fonction principale, et n'importe quel nombre de sous fonctions, qui figurent après la fonction principale, quelle les utilise. Les fonctions principales peuvent êtres appelés de l'extèrieur du fichier dans lequel ils sont définies, soit depuis la ligne de commande ou depuis d'autres fonctions, bien que les sous fonctions ne peuvent êtres appelés qu'à partir du fichier de la fonction. Les sous fonctions sont visibles seulement pour la fonction principale, et les autres sous fonctions dans le même fichier où il sont définies.

Example : Soit la fonction `equa2` qui calcul les racines d'une équation d'ordre 2. Le fichier du fonction equa2.m, va contenir la fonction principale `quadratic` et une sous fonction `dics` qui calcule le discriminant, comme suit :

```
function [x1,x2]= equa2(a,b,c)
%cette fonction retourn les racines
% d'une equation de 2 eme degree.
d = disc(a,b,c);
 if d<0
  disp('Pas de solutions');
 else
  x1 =(-b + sqrt(d))/(2*a);
  x2 =(-b - sqrt(d))/(2*a);
end
```

```
end   %end de la fonction principale

function dis = disc(a,b,c)
%fonction qui calculate le discriminant
 dis = b^2-4*a*c;
end  %end de la sous fonction
```

L'appel de la fonction depuis la ligne de commande :

```
[x1,x2]= equa2(2,4,-4)
```

MATLAB exécute les intructions précédengtes et retourne le résultat :

```
x1 =
    0.7321
x2 =
   -2.7321
```

3.4.2 Fonctions imbriquées

Les fonctions imbriquées sont des fonctions définies dans le corps d'autres fonctions.

Exemple : reprennat la fonction `equa2` précédente :

```
function[x1,x2]= equa2(a,b,c)
 function disc % fonction imbriquée
  d = sqrt(b^2-4*a*c);
 end    %end de la fonction disc
disc;
x1 =(-b + d)/(2*a); x2 =(-b - d)/(2*a);
end     %end of fonction principale equa2
```

Cette fois-ci la fonction `disc` est à l'intérieur de `equa2`.

Chapitre 4

Le graphisme

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

- Définir le vecteur x, repréentant les valeurs pour lesquel 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, et des labels dans les axes x-axis et y-axis, des lignes de grille et aussi adjuster les axes pour convenir au graph.

- `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` : Lorsuqe il s'agit de tracer plusieurs fonctions dans un même graphe, on peut mettre une légend 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 valuer 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 de la Figure 4.1 est obtenu.

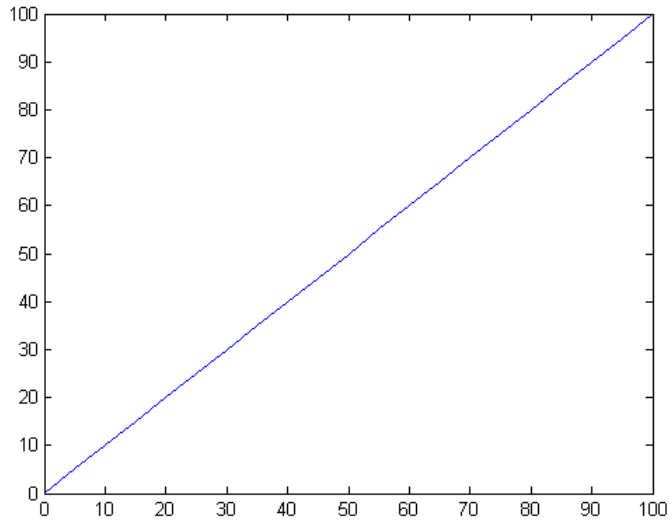


FIGURE 4.1 – Y=x

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)
```

Le graphe de la Figure 4.2 représente la courbe.

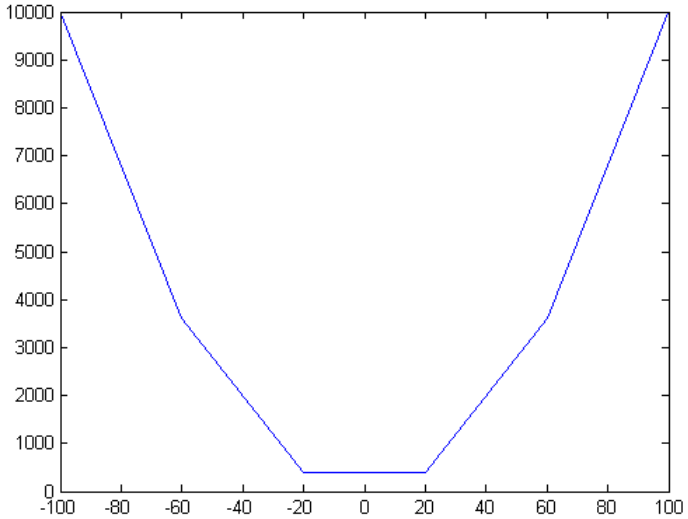


FIGURE 4.2 – $Y = x^2$

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. c.f dans la Figure 4.3.

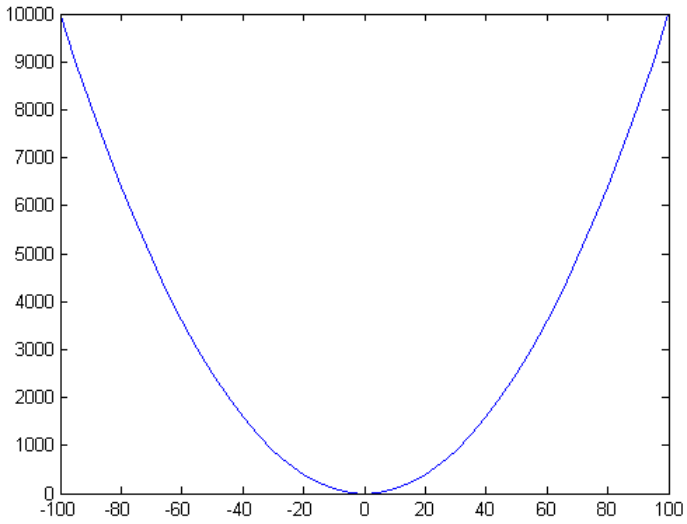


FIGURE 4.3 – $Y = x^2$

4.1 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 telque `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.

Exemple 1 : La fonction *sinus* :

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

La figure 4.4 montre le graphe correspondant, les valeur de y_{min}, y_{max} sont pris comme $[0, 1]$.

Exemple 2 : La fonction : $x.sin(x)$, voir Figure 4.5

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

4.2 Les titres sur les graphes

Soit la courbe de la fonction

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

Pour ajouter les étiquetes et le titre.

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

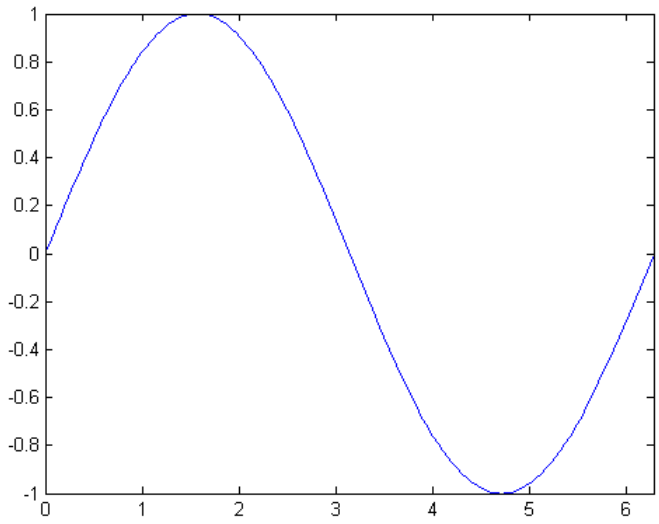


FIGURE 4.4 – Y=sin(x)

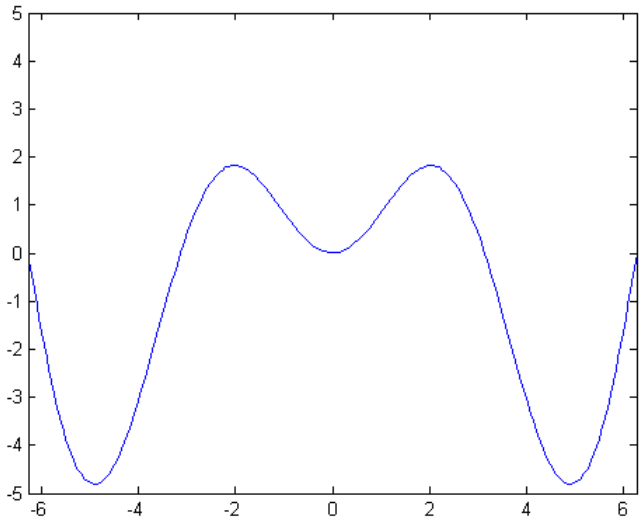


FIGURE 4.5 – Y=x.sin(x)

4.3 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)')
```

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

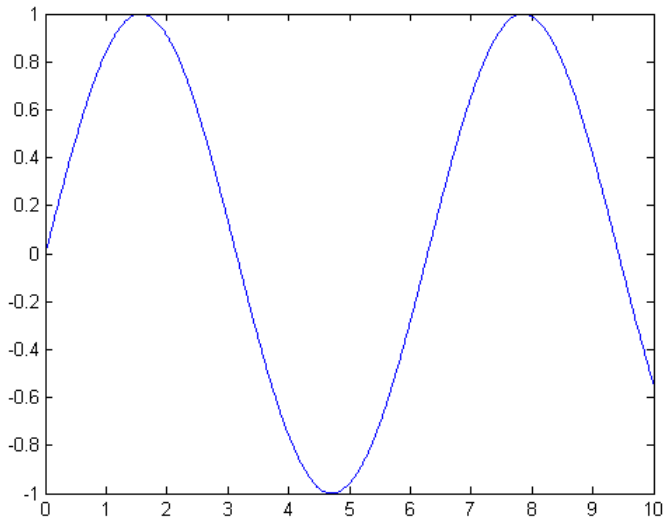


FIGURE 4.6 – Y=sin(x)

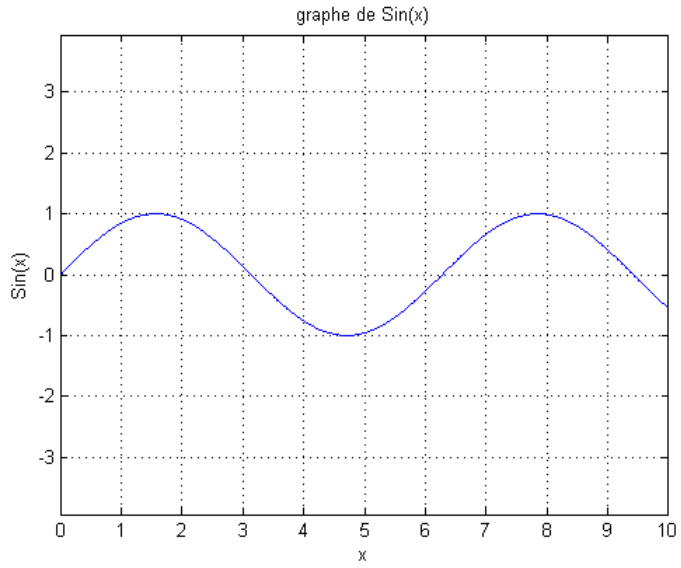


FIGURE 4.7 – Grapbe de : Y=sin(x)

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

4.4 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) :

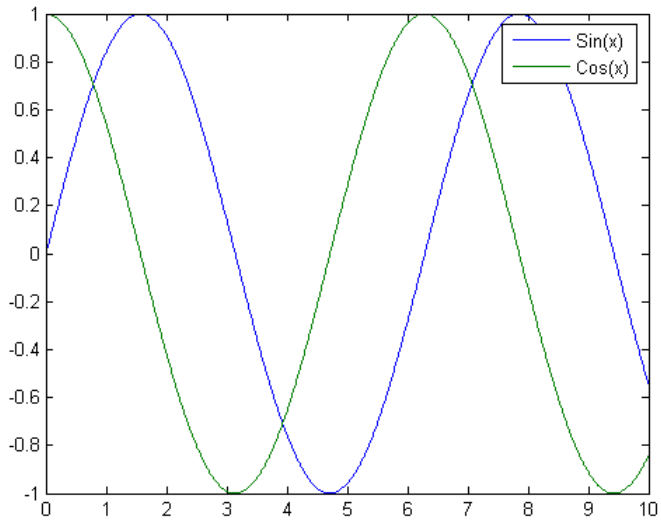


FIGURE 4.8 – Grapbe de : sinus et cosinus

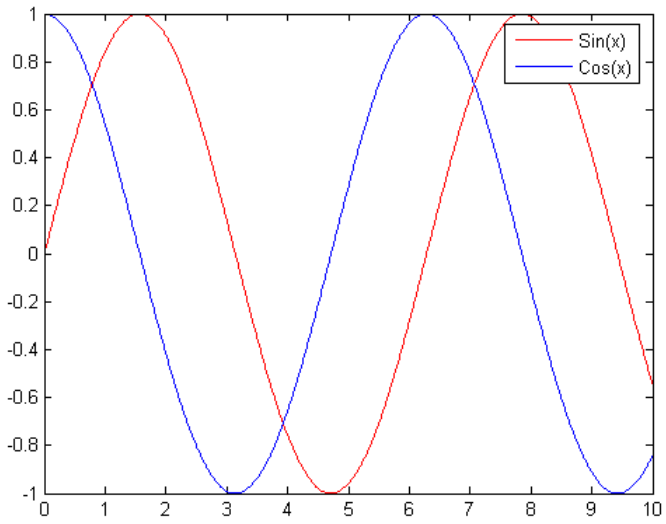


FIGURE 4.9 – Grapbe de : sinus et cosinus

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 court)
```

4.5 Plages de valeurs des axes

On peut paramétrer les valuers 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
```

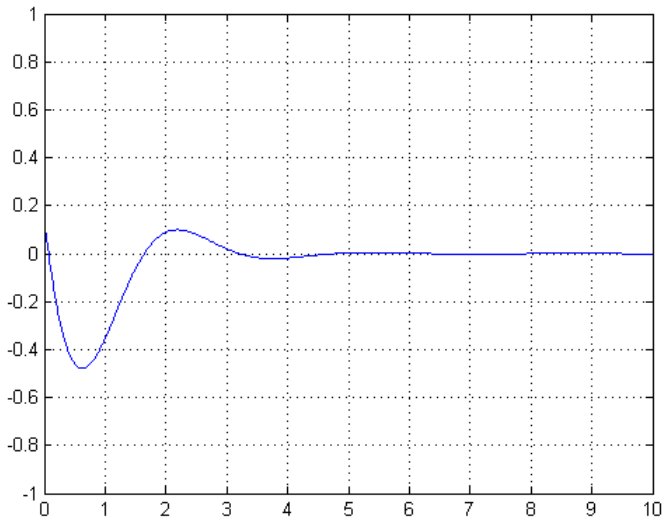


FIGURE 4.10 – Axes prédéfinies

4.6 Sous Graphes

On peut mêttre plusieurs sous graphes fonctions sur le même graphe, pour comparer plusieurs phénomèmes 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])
```

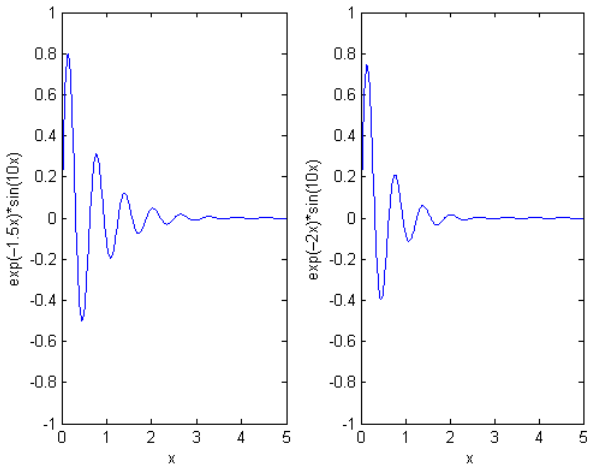


FIGURE 4.11 – Sous graphes

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')
```

4.7 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)`

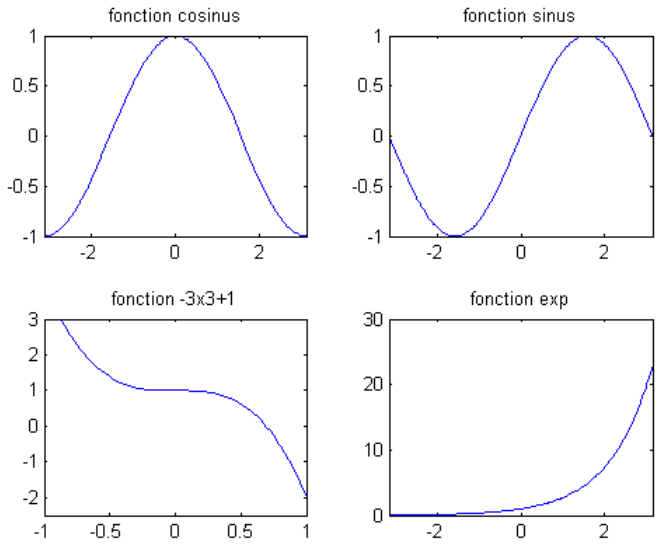


FIGURE 4.12 – Sous graphes(Exemple 2)

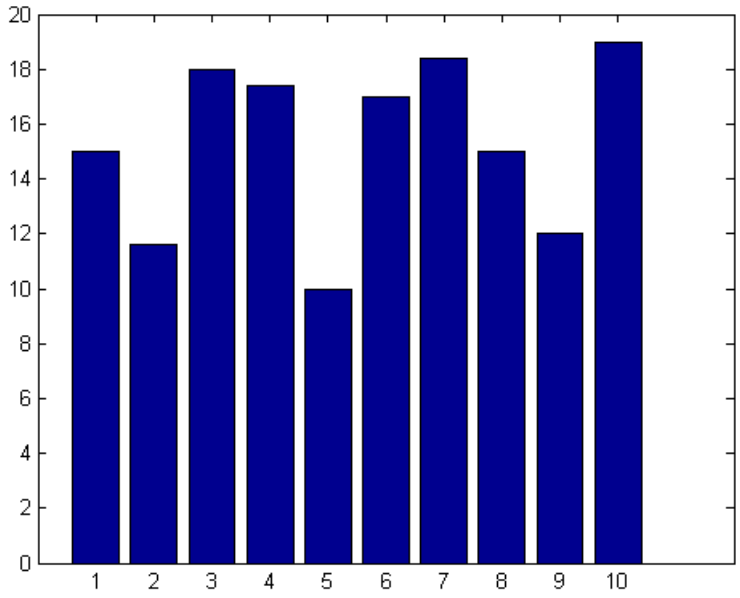


FIGURE 4.13 – Histogramme

Exemple 2 :

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

4.8 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);
```

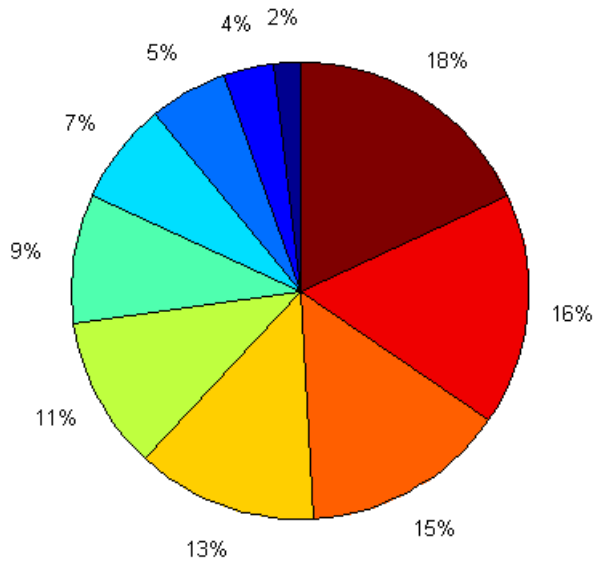


FIGURE 4.14 – Proportions

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

Voir Figure 4.15

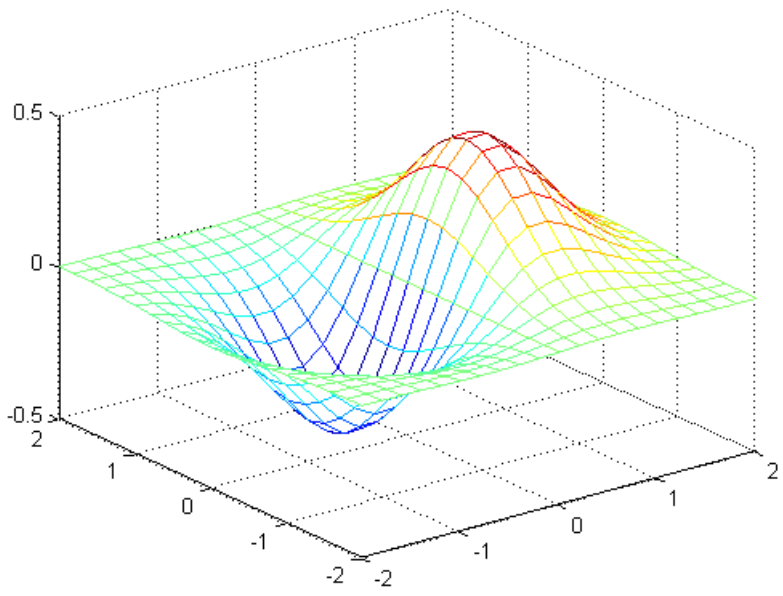


FIGURE 4.15 – Le fonction $f(x,y) = x.e^{-x^2-y^2}$ en 3D

Exemple2 :

```
x=0:1/1000:10;y=-sin(x);
```

```
z=cos(x).^2;
plot3(x,y,z)
xlabel('x');ylabel('y');zlabel('z');
```