

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
ET DE LA RECHERCHE SCIENTIFIQUE

AKLI MOHAND OULHADJ - BOUIRA UNIVERSITY

UNIVERSITE AKLI MOHAND OULHADJ-BOUIRA



وزارة التعليم العالي و البحث العلمي

جامعة أكلي محند أولحاج- البويرة

FACULTE DES SCIENCES ET DES SCIENCES APPLIQUEES

DEPARTEMENT DE GENIE ELECTRIQUE

كلية العلوم و العلوم التطبيقية

قسم الهندسة الكهربائية

Support de cours

Apprendre la Programmation en MATLAB

Programme de deuxième Année Licence en Génie Electrique (Informatique-3)

Polycopié à Caractère Pédagogique

Auteur :

D^r Amir Benzaoui

Maître de Conférences Classe - B

www.pimis.net/home/benzaoui.html

amirbenzaoui@gmail.com

Version 2016-2017

Avant propos

Ce document est un support de cours qui traite le programme détaillé du module « Informatique - 3 ». Il est destiné aux étudiants de deuxième année licence inscrits dans la spécialité « Génie Electrique »; mais il peut servir pour toutes les spécialités du domaine des sciences et techniques. Ce guide est inséré dans le génie électrique pour donner les connaissances nécessaires de base sur la programmation scientifique. Ces connaissances sont donc indispensables pour toute personne désirant une insertion dans une option du génie électrique dans le domaine professionnel ou recherche.

Etant donné que les langages informatiques sont généralement conçus pour résoudre un certain nombre de problèmes techniques, le choix d'un bon langage pour accomplir cette tâche est d'une importance capitale.

MATLAB est un logiciel interactif qui fournit à l'utilisateur un environnement lui permettant de réaliser un grand nombre de calculs scientifiques, en particulier ceux où les matrices interviennent. L'élément de base est un tableau qui ne demande pas de dimensionnement préalable. Cela nous permet de résoudre plus rapidement des problèmes techniques de calcul numérique par rapport aux autres langages de programmation tels que Java, C ou Fortran. MATLAB s'opère depuis une session de commandes en ligne (Mode Calculatrice). Celles-ci peuvent être exécutées une à une ou bien être enregistrées dans un fichier Script afin de l'exécuter comme un programme. Il existe un grand nombre de fonctions et commandes prédéfinies qui nous permet de réaliser:

- Des opérations vectorielles ou matricielles;
- Des calculs statistiques et des méthodes numériques;
- Des représentations de graphiques en deux ou trois dimensions;
- De la visualisation de données, signaux, et images;
- D'utiliser des techniques de l'intelligence artificielle comme: les réseaux de neurones, la logique floue... pour résoudre des problèmes mal-posés;
- ...

Semestre 3**UEM 2.1****Matière 2 : Informatique – 3** (VHS : 22h30, TP : 1h30)**Objectifs de la matière**

Apprendre à l'étudiant la programmation en utilisant des logiciels d'accès faciles (essentiellement: MATLAB). Cette matière sera un outil pour la réalisation des TP du module « Méthodes Numériques » en S4.

Connaissances préalables recommandées

Les bases de la programmation acquises en informatique 1 et 2.

Contenu de la matière

- Présentation d'un environnement de programmation scientifique (Matlab)
- Toolboxes
- Fichiers script et types de données et de variables
- Lecture, affichage et sauvegarde des données
- Vecteurs et matrices
- Polynômes
- Fichiers de fonction
- Instructions de contrôle (Boucles for et While, Instructions if et switch)
- Graphisme (Gestion des fenêtres graphiques, plot)

Mode d'évaluation

Contrôle continu: 100 %.

Références Bibliographiques

- N. Martaj and M. Mokhtari: « MATLAB R2009, SIMULINK et STATEFLOW pour Ingénieurs, Chercheurs et Etudiants ». Springer Science & Business Media, 2010.
- K. Ahlersten: « An Introduction to Matlab ». BookBoon Publisher, 2012.

Chapitre 01: Présentation de l'Environnement de Programmation Scientifique MATLAB

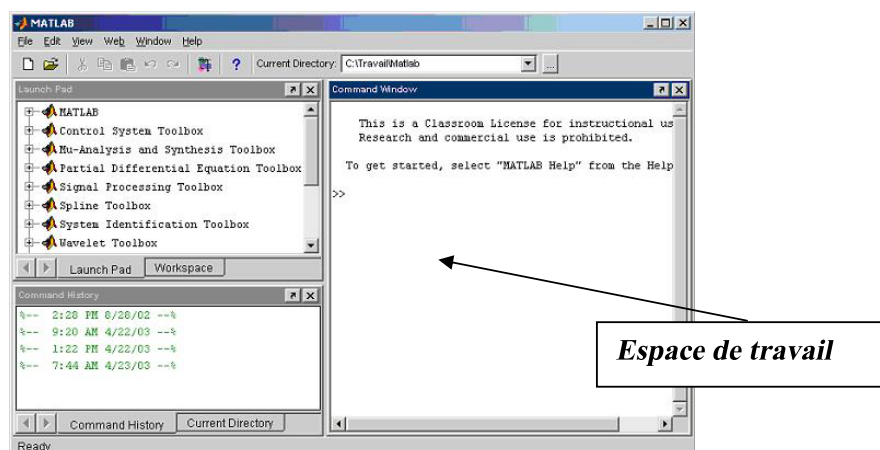
1. Introduction à MATLAB

1.1. Introduction

MATLAB est un système interactif et convivial de calcul numérique et de visualisation graphique destiné aux ingénieurs et aux scientifiques. Il possède un interpréteur de programmation à la fois puissant et simple à l'utilisation. Il permet d'exprimer les problèmes et les solutions d'une façon aisée, contrairement aux autres langages de programmation. MATLAB s'impose dans les mondes universitaire et industriel comme un outil puissant de simulation et de visualisation de problèmes numériques. Dans le monde universitaire, MATLAB est utilisé pour l'enseignement de l'algèbre linéaire, le traitement du signal, le traitement d'images, les visualisations graphiques, l'automatique, ainsi que dans la recherche scientifique. MATLAB est conforté par une multitude de boîtes à outils (Toolbox) spécifiques à des domaines variés.

1.2. Démarrage & Environnement

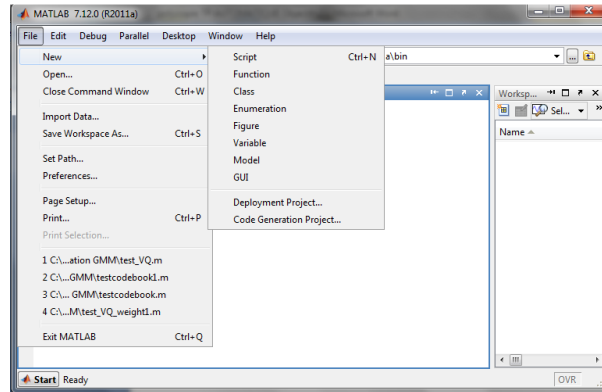
Pour démarrer *Matlab*, il suffit d'aller dans le menu *Démarrer, Programmes, Matlab 6.5* et de choisir *Matlab 6.5*. Ensuite, il est préférable de choisir le dossier dans lequel nous voulons travailler en entrant le chemin dans le menu *Current Directory*. L'environnement *Matlab* ressemble à celui-ci:



Au démarrage, *Matlab* affiche plusieurs fenêtres sur l'écran, les fenêtres les plus importantes sont:

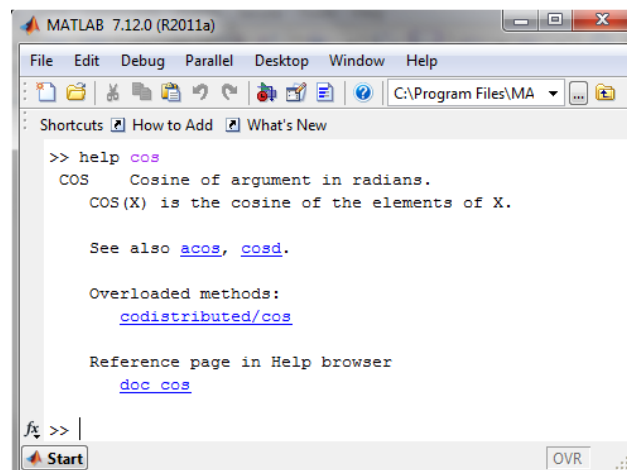
- **Command Window** Cette fenêtre permet d'entrer des commandes *Matlab*, de démarrer des fonctions, et d'utiliser des fichiers .M (l'extension d'un fichier *Matlab*). L'utilisateur peut entrer multiples commandes ou équations mathématiques après les chevrons ">>" qui apparaissent au côté gauche de la fenêtre. Pour exécuter une opération, il faut toujours appuyer sur la touche "enter" du clavier. De plus, il faut terminer l'opération par un point-virgule ";", sinon toutes les étapes du calcul seront affichées sur l'écran.
- **Launch Pad** Elle permet de lancer des applications comme *GUIDE*, *Simulink*, ou autres outils de *Matlab*. Pour y accéder, il suffit d'ouvrir l'arborescence en appuyant sur le signe "+".
- **Command History** Elle contient toutes les commandes qui ont été exécutées (historique de commandes). Il devient donc facile par la suite de trouver des erreurs dans l'exécution des commandes.
- **Workspace** Elle permet de visualiser toutes les variables qui ont été initialisées jusqu'à présent. Pour y accéder, il suffit de cliquer sur l'onglet *Workspace*. Ce menu permet aussi de modifier les valeurs des variables en cliquant deux fois sur celles-ci.
- **Current Directory** Fenêtre permettant de changer le dossier actuel et de démarrer un fichier quelconque avec un outil approprié.

L'environnement *Matlab* se présente sous forme d'un espace de travail (*Workspace*), où un interpréteur de commandes exécute des opérations et des fonctions *Matlab*. Dans un premier temps, nous pouvons contenter à introduire des commandes l'une après l'autre au niveau de l'espace de travail où elles sont interprétées directement. Par la suite, il est beaucoup plus pratique de décrire cette séquence de commandes complètement avec un éditeur de texte et de les sauvegarder dans un fichier avec l'extension .M. Par la suite, cette séquence peut être exécutée dans *Matlab* par une simple introduction du nom du fichier. *Matlab* possède un éditeur de texte intégré que nous l'appelons par: File-New-Script:

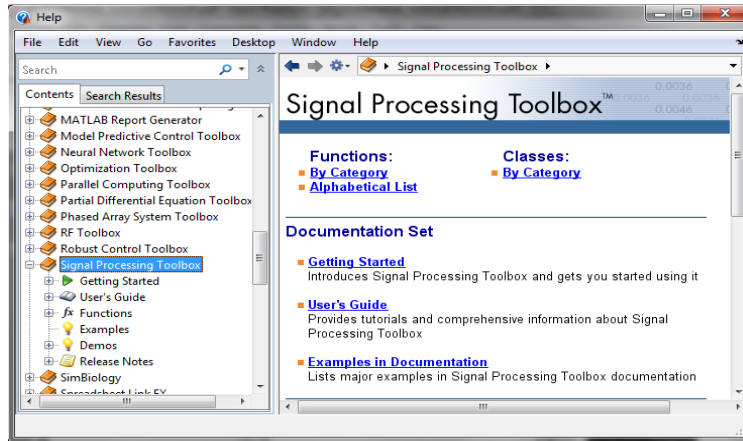


1.3. Obtenir de l'aide avec *Matlab*

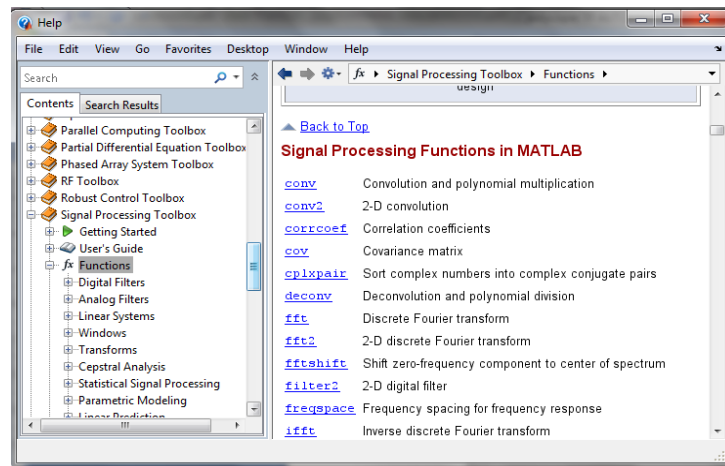
Il y a plusieurs manières d'obtenir de l'aide avec *Matlab*. La première option consiste à taper dans la fenêtre de commandes le mot « help » suivi du nom de la fonction ou la commande recherchée. Exemple: `help cos`; cela va fournir de l'aide sur l'utilisation de la fonction mathématique « cosinus »:



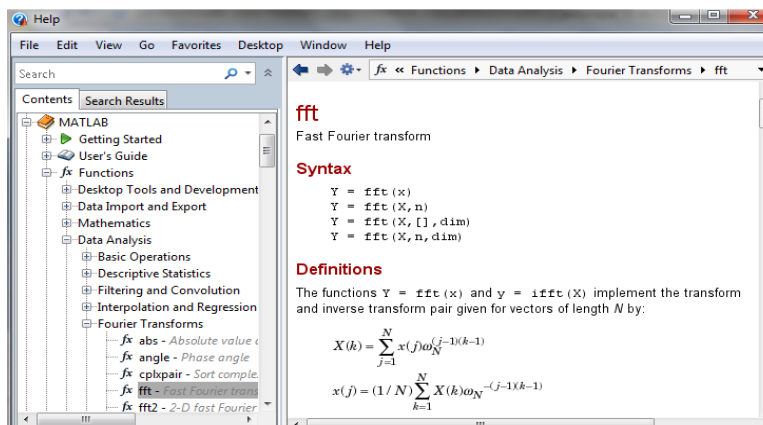
Si le nom exact de la fonction demandée n'est pas connu, il faut aller à la deuxième option qui consiste à taper « helpwin » dans la fenêtre de commandes. Cela, va afficher toutes les bibliothèques (Toolbox) de *Matlab* incluant les fonctions de chacune. Ensuite, nous pouvons choisir, par exemple, la section "Signal Processing Toolbox":



Ceci, va afficher toutes les fonctions disponibles dans cette catégorie:



Une fois le nom de la fonction qui nous intéresse est trouvé, nous pouvons cliquer au-dessus pour obtenir plus d'informations sur cette fonction (l'aide). Par exemple, si nous cliquons sur « `fft` ». Ceci, va expliquer comment utiliser la fonction « `fft` » (la transformée de Fourier discrète) et donne aussi l'ensemble de fonctions qui sont similaires ou de la même catégorie:



2. Les premiers calculs avec *Matlab* Variables, Vecteurs, et Matrices

Avec *Matlab* et contrairement aux autres langages de programmation, les variables sont représentées sous forme de matrices de « n » lignes par « m » colonnes. Cependant, il y a trois types particuliers de matrices: les matrices 1×1 (Scalaire ou Variable), les matrices 1×M ou N×1 (Vecteur), et les matrices M×N (Tableau ou Matrice). Ces trois types sont traités différemment par *Matlab*.

2.1. Variables

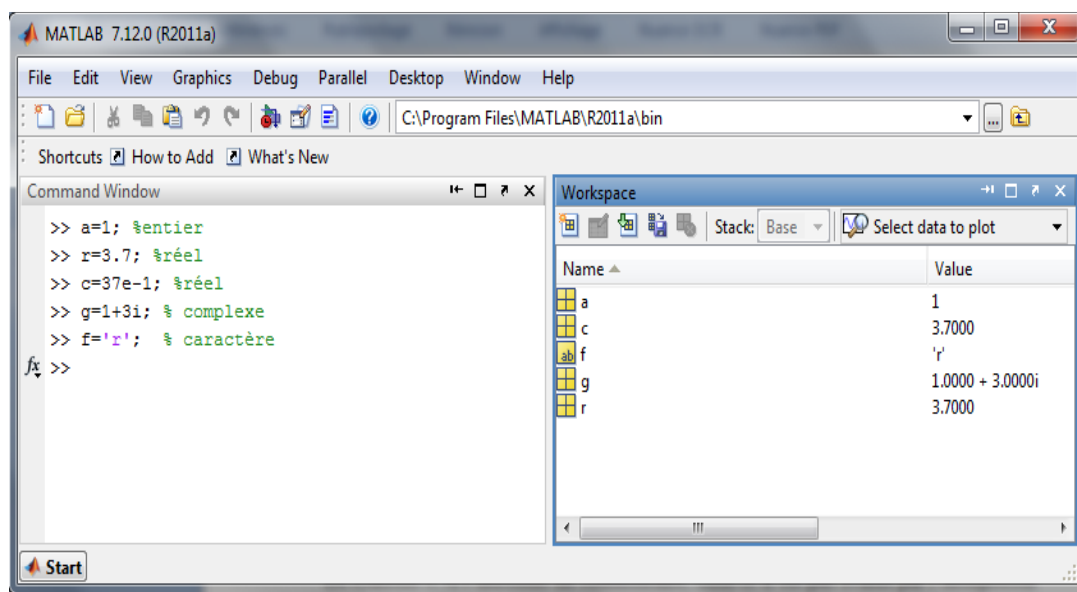
Les quatre principaux types de variables utilisés par *Matlab* sont les types: **réel**, **complexe**, **chaîne de caractères**, et **logique**. Signalons qu'il est inutile (impossible) de déclarer le type d'une variable. Ce type est établi automatiquement à partir de la valeur affectée à la variable. Pour assigner une valeur ou une expression à une variable sur *Matlab*, il faut utiliser l'opérateur « = ». Avec *Matlab*, la syntaxe générale pour créer une variable est de la forme:

$$\text{Nom_de_la_variable} = \text{expression} ;$$

Ou simplement:

$$\text{expression} ;$$

Ici « expression » peut être une constante, une autre variable, une matrice, un vecteur,...etc. Par exemple, les instructions `a=1; r= 3.7; c=5.2; g=1+3i; f=' r' ;` définissent trois variables *a*, *r* et *c* de type réel, une variable *g* de type complexe, et une variable *f* de type chaîne de caractères:



- Le symbole « % » introduit un commentaire, ceci sera ignoré par l'interpréteur.
- Le type logique (logical) possède 2 formes: 0 pour *faux* et 1 pour *vrai*. Un résultat de type logique est retourné par certaines fonctions ou dans le cas de certains tests.
- Les variables définies par l'utilisateur sont rangées dans l'espace mémoire de *Matlab*, ces variables sont dites *globales*.
- Comme nous ne définissons pas de manière explicite le type d'une variable, il est parfois utile de pouvoir le déterminer. Cela est possible grâce aux commandes: `ischar`, `islogical`, et `isreal` :
 - `ischar(x)` : retourne 1 si *x* est de type chaîne de caractères et 0 sinon.
 - `islogical(x)` : retourne 1 si *x* est de type logique et 0 sinon.
 - `isreal(x)` : retourne 1 si *x* est de type réel et 0 sinon.
- **Exemples d'utilisation des variables avec *Matlab***

Dans un premiers temps, *Matlab* peut être utilisé comme une calculatrice:

```
>> 1+2
```

Affiche comme résultat: `ans = 3`, `ans` est une variable temporaire qui contient le résultat de l'opération.

```
>> 4^2
```

Affiche comme résultat: `ans = 16`, le symbole « ^ » signifie la puissance.

```
>> ((1+2)^3)*4)/2
```

Affiche comme résultat: `ans = 54`, *Matlab* respecte les règles usuelles utilisées en mathématique.

Il est aussi possible, comme dans tout langage de programmation, de définir des variables pour stocker un résultat, par exemple:

```
>> a=1+2
```

```
>> b=2+3
```

```
>> a+b
```

Retourne comme résultat: `a = 3`, `b = 5`, et `ans = 8`

Si nous voulons éviter que *Matlab* affiche les résultats intermédiaires, nous ajoutons simplement un « ; » à la fin de chaque commande.

Enfin, les calculs avec les variables complexes sont parfaitement possibles, en utilisant *i* ou *j* (avec bien évidemment $i^2 = j^2 = -1$):

```
>> a=1+2i ;
```

```
>> b=2+3i ;
```

```
>> a*b
```

Donne la réponse attendue, à savoir: $ans = -4 + 7i$.

Nous pouvons ainsi vérifier que:

```
>> exp(i * pi)+1
```

Est *null*, et que:

```
>> sqrt(-1)
```

Retourne $-i$;

Matlab étant très tolérant, il est possible d'appeler une variable i (ou bien j), mais dans ce cas, nous ne pourrons plus utiliser par la suite i comme étant le générateur de nombres complexes. Il est donc utile de savoir quelles sont les variables que nous avons utilisées depuis le début de la session, et pour cela il suffit d'écrire:

```
>> who
```

Et pour en savoir plus sur elles:

```
>> whos
```

Enfin, pour les détruire, nous utilisons:

```
>> clear a
```

Ou encore pour toutes les variables:

```
>> clear all
```

Certaines variables très utiles sont déjà prédéfinies sur *Matlab*, comme:

```
>> pi
```

Ou :

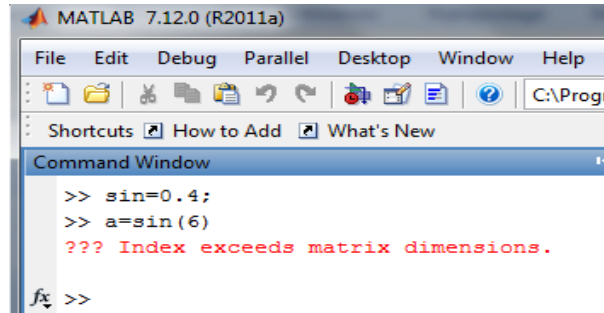
```
>> ans
```

Qui retourne la dernière réponse donnée par *Matlab*. Il est aussi utile de savoir qu'en tapant sur la touche \uparrow nous pouvons rappeler les commandes précédentes (la flèche \downarrow permettant de revenir) cela dispense de taper plusieurs fois des commandes identiques.

- **En comparaison avec d'autres langages de programmation:**

- En *Matlab*, il ne peut jamais déclarer les variables au préalable.
- *Matlab* fait la distinction entre minuscules et majuscules.

- Il faut faire attention au choix des noms de variables car nous pouvons surcharger le nom d'une fonction, que nous ne pourrions plus utiliser par la suite. Exemple:



```

MATLAB 7.12.0 (R2011a)
File Edit Debug Parallel Desktop Window Help
C:\Progr
Shortcuts How to Add What's New
Command Window
>> sin=0.4;
>> a=sin(6)
??? Index exceeds matrix dimensions.
fx >>

```

2.2. Lecture et affichage de données

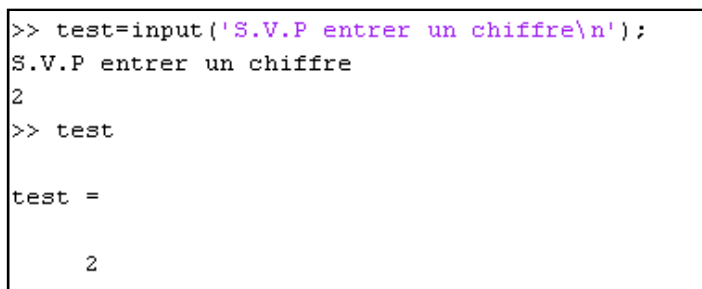
Il est également possible d'assigner une valeur provenant du clavier à une variable en utilisant la commande « `input` ». Ceci veut dire que l'utilisateur doit manuellement initialiser les variables une fois que le programme est exécuté. Par exemple :

```

>> test=input('S.V.P entrer un chiffre\n');
>> 2
>> test

```

Résultat:



```

>> test=input('S.V.P entrer un chiffre\n');
S.V.P entrer un chiffre
2
>> test

test =

    2

```

Une fois le code est exécuté, nous voyons que la variable « `test` » est maintenant égale à « 2 »; le contenu de cette variable a été saisi par l'utilisateur. Notons aussi que le symbole « `\n` » veut dire un « saut de ligne ». De plus, il est possible de créer des tableaux de caractères en utilisant le symbole « `s` ».

```

>> test_string= input ('Entrer un mot:\n', 's');
>> signaux
>> test_string

```

Résultat:

```

>> test_string= input ('Entrer un mot:\n', 's');
Entrer un mot:
signaux
>> test_string

test_string =

signaux

```

2.3. Vecteurs

a- Création et manipulation d'un vecteur

Le moyen le plus simple pour créer un vecteur de type « ligne » est de mettre ses éléments entre deux crochets [] et en les séparant par des espaces ou des virgules:

```

>> x = [6    4    7]
      x =
          6     4     7
>>

```

Nous définissons un vecteur de type « colonne » en donnant la liste de ses éléments séparés par des points virgules « ; » ou par des retours chariots:

```

>> x = [6;  4;  7]
      x =
          6
          4
          7
>>

```

Afin d'éviter l'affichage du résultat d'une expression quelconque, nous terminerons celle-ci par un point-virgule:

```

>> x = [6 4 7];
>>

```

Autre façon de saisir un vecteur de type « ligne » est:

```

>> x = [6, 4, 7];
>>

```

Ce vecteur est considéré comme une matrice à une ligne et trois colonnes:

```

>> size (x)

ans =

     1     3

```


Les dimensions d'un tableau quelconque peuvent être récupérées par la commande « `size` » sous forme d'un vecteur $[m \ n]$, où m et n étant respectivement le nombre de lignes et de colonnes.

```
>> [m n] = size (x)
      m =
          1
      n =
          3
>>
```

La longueur d'un tableau quelconque est, par définition, sa plus grande dimension:

```
>> l = length (x)
      l =
          3
>>
```

Comme mentionné au-dessus, un vecteur est facilement construit avec *Matlab* en mettant tous ses éléments l'un après l'autre. Si le vecteur est très grand, cette méthode devient inefficace parce qu'elle prend beaucoup de temps pour inscrire tous les éléments un par un. En d'autres termes, nous pouvons utiliser un raccourci pour créer des vecteurs; si les composantes d'un vecteur sont espacées d'un pas constant et si la première et la dernière valeur sont connues, alors ce vecteur peut être décrit de la manière suivante :

```
Vecteur = valeur_initiale : incrément : valeur_finale;
```

Exemple 01: Si nous voulons créer le vecteur suivant:

```
t = (0 0.2500 0.5000 0.7500 1.0000)
```

Nous utilisons les instructions suivantes:

```
>> % exemple 01
>> ind = 0.25; % création d'une variable ind (pas de transition)
>> debut = 0; fin = 1; % création de deux variables: début et
fin du vecteur
>> t = debut : ind : fin
      t =
          0    0.2500    0.5000    0.7500    1.0000
```

Si nous omettons de ne pas spécifier le pas de transition, celui-ci est pris par défaut à 1.

Exemple 02: Si nous voulons créer le vecteur x défini par: $x = (3 \ 4 \ 5 \ 6)$, nous utilisons la commande:

```
>> % exemple 02
>> x = 3 : 6
      x =
           3         4         5         6
```

b- Opérations sur les vecteurs

Une particularité sur *Matlab* est d'effectuer des opérations de manière globale sur les éléments d'un vecteur de type réel ou complexe sans avoir besoin de manipuler directement ses éléments:

- Si k est une variable scalaire et x un vecteur, l'instruction $k*x$ multiplie tous les éléments de x par k .
- Si x et y sont deux vecteurs de longueur identique, l'instruction $z = x+y$ (respectivement $x-y$) définit le vecteur z dont ses éléments sont $z(i) = x(i) + y(i)$ (respectivement $z(i) = x(i) - y(i)$).
- Nous obtenons un vecteur z dont la j° composante est le produit (respectivement le quotient) de la i° composante du vecteur x par la i° composante du vecteur y en effectuant l'instruction $z = x.*y$ (respectivement $z = x./y$).
- En procédant d'un point aux opérateurs $*$, $/$, \backslash et \wedge , nous réalisons des opérations élément par élément:

$v.*b$	Multiplier chaque élément du vecteur v par son correspondant du vecteur b
$x.^2$	Enlever en puissance de 2 chaque élément du vecteur x
$v./b$	Diviser chaque élément du vecteur v par l'élément correspondant du vecteur b
$v.\backslash b$	Diviser chaque élément du vecteur b par l'élément correspondant du vecteur v

c- Vecteurs particuliers

Les commandes **ones**, **zeros**, et **rand** permettent de définir des vecteurs dont ses éléments sont respectivement des valeurs 1, 0, et des nombres générés de manière aléatoire:

- `ones (1, n)` : vecteur ligne de longueur n dont tous les éléments valent 1.
- `ones (m, 1)` : vecteur colonne de longueur m dont tous les éléments valent 1.
- `zeros (1, n)` : vecteur ligne de longueur n dont tous les éléments valent 0.

- `zeros (m, 1)` : vecteur colonne de longueur m dont tous les éléments valent 0.
- `rand(1, n)` : vecteur ligne de longueur n dont les éléments sont générés de manière aléatoire entre 0 et 1.
- `rand(m, 1)` : vecteur colonne de longueur m dont les éléments sont générés de manière aléatoire entre 0 et 1.

2.4. Matrices

a- Création et manipulation d'une matrice

La matrice ou tableau à 2 dimensions est un outil de base en *Matlab*. La saisie d'une matrice peut être faite de différentes manières:

- Vecteurs lignes séparés par un saut de ligne (;)

```
>> m1 = [5 7 9 ; 1 4 2]
m1 =
      5      9      7
      1      4      2
```

- Vecteurs séparés par un retour à la ligne

```
>> m2 = [3 5 7
8 6 10]
m2 =
      3      5      7
      8      6     10
```

b- Matrices particulières

Certaines matrices se construisent très simplement grâce à des commandes dédiées.

Citons les plus utilisées:

- `ones (m, n)` : la matrice à m lignes et n colonnes dont tous les éléments valent 1.
- `zeros (m, n)` : la matrice à m lignes et n colonnes dont tous les éléments valent 0.
- `rand(m, n)` : une matrice à m lignes et n colonnes dont les éléments sont générés de manière aléatoire entre 0 et 1.

2.5. Manipulations des vecteurs et matrices

Pour les vecteurs, les indices commencent à 1. Pour récupérer ou modifier un élément d'un vecteur, il faut spécifier son indice à l'intérieur de deux parenthèses.

Exemple: Soit le vecteur $x = (1.0, 1.2, 1.4, 1.6, 1.8, 2.0, 2.2, 2.4, 2.6, 2.8, 3.0)$

✓ **Pour créer le vecteur $x \rightarrow x = 1 : 0.2 : 3$**

```
>> x = 1 : 0.2 : 3
x =
    Columns 1 through 7
    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000    2.2000
    Columns 8 through 11
    2.4000    2.6000    2.8000    3.0000
```

✓ **Pour récupérer le 2^{ème} élément de $x \rightarrow x(2)$**

```
>> x (2)
ans =
    1.2000
```

✓ **Pour modifier la valeur du 3^{ème} élément de $x \rightarrow x(3) = 0$**

```
>> x (3) = 0
x =
    Columns 1 through 8
    1.0000    1.2000    0    1.6000    1.8000    2.0000    2.2000    2.4000
    Columns 9 through 10
    2.6000    2.8000    3.0000
```

✓ **Pour récupérer le 5^{ème} élément jusqu'au 9^{ème} élément de x dans $y \rightarrow y = x(5:9)$**

```
>> y = x (5 : 9)
y =
    1.8000    2.0000    2.2000    2.4000    2.6000
```

✓ **Pour récupérer le 5^{ème} élément jusqu'au dernier élément de x dans $z \rightarrow z = x(5:end)$ ou bien $z = x(5:length(x))$**

```
>> z = x ( 5 : end )
z =
    2.0000    2.2000    2.4000    2.6000    2.8000    3.0000
```

ou

```
>> z = x ( 5 : length(x) )
```

```

z =
    2.0000  2.2000  2.4000  2.6000  2.8000  .0000
✓ Pour récupérer le 5ème élément, le 8ème et le dernier élément de x → x ([ 5 8 end ])
>> x ( [ 5 8 end ] )
z =
    1.8000  2.4000  3.0000

```

✓ **Pour remplacer le 5^{ème} élément, le 8^{ème} et le dernier élément de x par 1 → x ([5 8 end]) = 1**

```

>> x ( [ 5 8 end ] ) = 1
x =
    Columns 1 through 8
    1.0000  1.2000  0  1.6000  1.0000  2.0000  2.2000  1.0000
    Columns 9 through 10
    2.6000  2.8000  1.0000

```

✓ **Pour remplacer le 5^{ème} élément, le 8^{ème} et le dernier élément de x par 0, 2, 4 respectivement → x ([5 8 end]) = [0 2 4]**

```

>> x ( [ 5 8 end ] ) = [ 0 2 4 ]
x =
    Columns 1 through 7
    1.0000  1.2000  0  .6000  0  2.0000  2.2000
    Columns 8 through 11
    2.0000  2.6000  2.8000  4.0000

```

A partir d'une matrice, nous pouvons extraire une autre matrice, un vecteur ou l'un de ses éléments. Pour extraire un élément d'une matrice, il suffit de spécifier l'indice de la ligne et celui de la colonne où se trouve cet élément $x(i,j)$ à l'intérieur de deux parenthèses séparées par une virgule.

Exemple: Soit la matrice A définie par :

$$A = \begin{bmatrix} 0 & 6 & 7 \\ 5 & 8 & 9 \\ 3.6 & 5.7 & 7 \\ 0 & 0 & 1 \end{bmatrix}$$

✓ **Pour créer la matrice A** → $A = [0 \ 6 \ 7 ; 5 \ 8 \ 9 ; 3.6 \ 5.7 \ 7 ; 0 \ 0 \ 1]$

```
>> A = [ 0 6 7 ; 5 8 9 ; 3.6 5.7 7 ; 0 0 1 ]
A =
    0         6.0000    7.0000
    5.0000    8.0000    9.0000
    3.6000    5.7000    7.0000
    0         0         1.0000
```

✓ **Pour récupérer l'élément de la deuxième ligne et la première colonne dans R1**

→ $R1 = A(2, 1)$

```
>> R1 = A( 2 , 1 )
R1 =
    5.0000
```

✓ **Pour récupérer la deuxième colonne dans R2** → $R2 = A(:, 2)$

```
>> R2 = A( : , 2 )
R2 =
    6.0000
    8.0000
    5.7000
    0
```

✓ **Pour récupérer la troisième ligne dans R3** → $R3 = A(3, :)$

```
>> R3 = A( 3 , : )
R3 =
    3.6000    5.7000    7.0000
```

✓ **Pour récupérer une partie de la matrice composée de lignes 1 à 2 de A et des colonnes 2 à 3 de A dans R4** → $R4 = A(1:2, 2:3)$

```
>> R4 = A(1:2, 2:3)
```

```
R4 =
      6.0000      7.0000
      8.0000      9.0000
```

- ✓ **Pour récupérer une partie de la matrice A dans R5 où cette partie est composée de toutes les colonnes de A et des lignes 3 à 4 → $R5 = A(3 : 4, :)$**

```
>> R5 = A( 3 : 4 , : )
R5 =
      3.6000      5.7000      7.0000
      0          0          1.0000
```

- ✓ **Pour remplacer la 3^{ème} colonne de A par les valeurs 1 5 9 10 respectivement → $A(:, 3) = [1; 5; 9; 10]$**

```
>> A ( : , 3 ) = [ 1 ; 5 ; 9 ; 10 ]
A =
      0          6.0000      1.0000
      5.0000      8.0000      5.0000
      3.6000      5.7000      9.0000
      0          0          10.0000
```

- ✓ **Pour récupérer le bloc formé par les lignes 1 et 3 et les colonnes 2 à 3 de la matrice A dans R6 → $R6 = A([1 3], 2 : 3)$**

```
>> R6 = A( [ 1 3 ] , 2 : 3 )
R6 =
      6.0000      1.0000
      5.7000      9.0000
```

Nous pouvons aussi supprimer une ou plusieurs lignes (respectivement colonnes) d'une matrice en les remplaçant par un ensemble vide symbolisé par deux crochets [] :

- ✓ **Si nous voulons supprimer la 2^{ème} colonne de A → $A(:, 2) []$**

```
>> A ( : , 2 ) = [ ]
A =
      0          1.0000
      5.0000      5.0000
      3.6000      9.0000
      0          10.0000
```

2.6. Concaténation et fonctions spécifiques aux vecteurs et matrices

La concaténation de 2 ou plusieurs vecteurs / matrices de dimensions adéquates peut être faite horizontalement ou verticalement :

Exemple: Soit les matrices A et B suivantes: $A = [5\ 7\ 3; 2\ 9\ 4]$ et $B = [5\ 3\ 0; 7\ 1\ 6]$

La concaténation **horizontale** se faite de la manière suivante:

```
>> A = [ 5 7 3 ; 2 9 4 ]; % Création de la matrice A
>> B = [ 5 3 0 ; 7 1 6 ]; % Création de la matrice B
>> ConcatHoriz = [ A B ] % Concaténation des deux matrices A
et B dans ConcatHoriz
ConcatHoriz =
      5      7      3      5      3      0
      2      9      4      7      1      6
```

La concaténation **verticale** se faite de la manière suivante (Ajout du point-virgule):

```
>> ConcatVert = [ A ; B ] % Concaténation des deux matrices A
et B dans ConcatVert
ConcatVert =
      5      7      3
      2      9      4
      5      3      0
      7      1      6
```

Il existe également quelques fonctions spécifiques aux vecteurs, telles que:

- `sum(x)` : calcule la somme des éléments du vecteur x ,
- `prod(x)` : calcule le produit des éléments du vecteur x ,
- `max(x)` : retourne plus grand élément du vecteur x ,
- `min(x)` : retourne le plus petit élément du vecteur x ,
- `mean(x)` : calcule la moyenne des éléments du vecteur x ,
- `sort(x)` : ordonne les éléments du vecteur x par ordre croissant,
- `fliplr(x)` : renverse l'ordre des éléments du vecteur x .

Exemple:

```
>> x = [10 5 8 0 4]; % création du vecteur x
>> sum ( x )
ans =
      27
>> prod ( x )
ans =
      0
>> max ( x )
ans =
      10
>> min ( x )
```



```

ans =
      0
>> sort ( x )
ans =
      0      4      5      8     10
>> fliplr ( x )
ans =
      4      0      8      5     10

```

Pour les matrices, les opérations propres aux vecteurs, telles que **sum**, **mean**, **max**,... se font selon les colonnes puis le vecteur ligne obtenu, par exemple :

```

>> A = [ 0 6 7 ; 5 8 9 ; 3.6 5.7 7 ; 0 0 1 ] ;
>> sum ( A )
ans = 13      22      16.3      1

```

Par contre:

```

>> sum ( sum ( A ) )
ans =
      52.3000

```

Le symbole « : » transforme une matrice en un vecteur de type colonne. Avec cette transformation, une seule commande « **sum** » suffit pour faire la somme de tous les éléments d'une matrice multidimensionnelle:

```

>> sum ( A ( : ) )
ans =
      52.3000

```

2.7. Opérations matricielles sur les vecteurs et les matrices

Matlab est vraiment performant pour la manipulation des vecteurs, des matrices, ou des tableaux en général. Pour la facilité, nous noterons par la suite les vecteurs par des minuscules et les matrices par des majuscules. Par exemple, la multiplication des vecteurs et des matrices se fait par l'utilisation de la notation « * » comme suit:

```

>> A = [ 2 5 0; 9 7 3 ]; % Création d'une matrice A
A =
      2      5      0
      9      7      3
>> C = [ 1 3 4; 5 2 1; 3 1 2 ]; % Création d'une
matrice C
C =
      1      3      4
      5      2      1
      3      1      2
>> b = [ 3 1 2 ] % Création d'un vecteur b

```

```

b =
      3      1      2
>> v = [ 1 3 4 ] % Création d'un vecteur v
v =
      1      3      4
>> v * b'
ans =
      14

```

Explication du calcul:

$$\begin{matrix} & & & & 3 \\ & & & & 1 \\ & & & & 2 \\ 1 & 3 & 4 & * & \begin{matrix} 1 \\ 3 \\ 2 \end{matrix} = 1 \times 3 + 3 \times 1 + 4 \times 2 = 14 \end{matrix}$$

```

>> v' * b
ans =
      3      1      2
      9      3      6
     12      4      8

```

Explication du calcul:

$$\begin{matrix} 1 & & & 1 \times 3 & 1 \times 1 & 1 \times 2 \\ 3 & * & 3 & 1 & 2 = & 3 \times 3 & 3 \times 1 & 3 \times 2 \\ 4 & & & 4 \times 3 & 4 \times 1 & 4 \times 2 \end{matrix}$$

Notez bien que cette multiplication n'est pas commutative.

```

>> A*C
ans =
     27     16     13
     53     44     49

```

Explication du calcul matriciel :

$$\begin{matrix} 2 & 5 & 0 & * & \begin{matrix} 1 & 3 & 4 \\ 5 & 2 & 1 \\ 3 & 1 & 2 \end{matrix} = & \begin{matrix} (2 \times 1 + 5 \times 5 + 0 \times 3) & (2 \times 3 + 5 \times 2 + 0 \times 1) & (2 \times 4 + 5 \times 1 + 0 \times 2) \\ (9 \times 1 + 7 \times 5 + 3 \times 3) & (9 \times 3 + 7 \times 2 + 3 \times 1) & (9 \times 4 + 7 \times 1 + 3 \times 2) \end{matrix} \end{matrix}$$

```
>> C*A
```

??? Error using ==> mtimes

Inner matrix dimensions must agree.

Explication : Pour faire la multiplication entre deux matrices, le nombre de lignes de la matrice A doit être égal au nombre de colonnes de la matrice C.

Pour les matrices carrées, il existe aussi des fonctions spécifiques extrêmement utiles:

- `inv(A)` : calcule l'inverse de A.
- `det(A)` : calcule le déterminant de A.
- `diag(A)` : retourne la diagonale de A.

Si nous voulons travailler directement avec les éléments des vecteurs/matrices. Par exemple: pour multiplier tous les éléments de A par ceux de B, il ne s'agit plus alors

d'une opération matricielle, mais bien d'une multiplication des éléments de matrices. Pour spécifier que nous utilisons les éléments et non pas les matrices, nous écrivons - par exemple pour la multiplication - « .* » au lieu de « * » :

```
>> A.*B ;    % A et B doivent avoir les mêmes dimensions  
>> A./B ;  
>> A.^2 ;
```

Chapitre 02: Polynômes et Résolution de Systèmes Linéaires

1. Polynômes

1.1. Construction d'un polynôme

Avec *Matlab*, le polynôme de degré n , $p(s) = a_n s^n + a_{n-1} s^{n-1} + \dots + a_1 s^1 + a_0$ est défini par un vecteur A de dimension $n+1$ contenant les coefficients $\{a_i\} \{i = 0, \dots, n\}$ rangés dans l'ordre décroissant des indices; c'est-à-dire: $A(1) = a_n$, $A(2) = a_{n-1}$, ..., et $A(n+1) = a_0$. Nous pouvons construire un polynôme avec *Matlab* de deux manières différentes :

- a) En mettant, dans un vecteur de type ligne, les coefficients classés dans l'ordre de puissances décroissantes:

Exemple: pour créer le polynôme $p(s) = s^3 + 5s^2 + 4s$, nous écrivons:

```
>> A = [ 1 5 4 0 ];
```

- b) En spécifiant les racines du polynôme et en utilisant la fonction «`poly`»:

Exemple: Spécification du même polynôme à: $p(s) = s(s+1)(s+4)$, ensuite:

```
>> A = poly ( [ 0 -1 -4 ] );
```

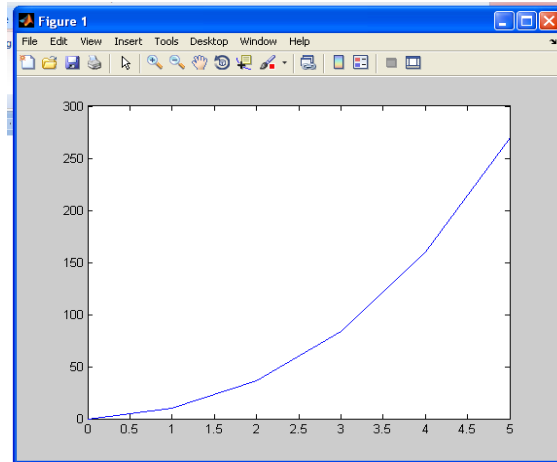
La commande «`polyval`» permet d'évaluer le polynôme $p(s)$ (la fonction polynomiale) en un ensemble de points donnés. Sa syntaxe est `polyval(A, x)` ou x est une valeur numérique ou un vecteur. Dans le second cas, nous obtenons un vecteur contenant des valeurs de la fonction polynomiale aux différents points spécifiés dans le vecteur x . Avec la fonction «`plot`», la commande «`polyval`» permet de tracer le graphe de la fonction polynomiale sur l'intervalle du vecteur x . La syntaxe de l'instruction est:

```
plot ( x, polyval ( A , x ) )
```

Exemple:

```
>> A = [ 1 5 4 0 ]; % Création du polynôme p
>> x = 0 : 5 ; % Création du vecteur x = (0 1 2 3 4 5)
>> polyval ( A , x)
ans =
      0      10      36      84     160     270
```

```
>> plot ( x , polyval ( A , x ) )
>>
```



1.2. Multiplication et division de polynômes

La multiplication et la division de polynômes peuvent être réalisées facilement avec *Matlab*. Pour multiplier deux polynômes $p_1(s)$ et $p_2(s)$, nous utilisons la commande « `conv` », et pour la division nous utilisons la commande « `deconv` ».

Exemple: Soit deux polynômes $p_1(s) = s + 3$ et $p_2(s) = s^2 + 5s + 2$ définis par:

```
>> p1 = [ 1 3 ] ;
>> p2 = [ 1 5 2 ] ;
>> p = conv ( p1 , p2 )
p =
      1      8     17      6
```

Ce qui signifie le polynôme $p(s) = s^3 + 8s^2 + 17s + 6$.

Pour la division, si nous voulons, par exemple, retrouver le polynôme $p_{11}(s)$ à partir de la division de $p(s)$ sur $p_2(s)$, nous utilisons la fonction « `deconv` » comme suit :

```
>> p11 = doconv ( p , p2 )
p11 =
      1      3
```

Dans le cas général, la division ne peut pas toujours donner un résultat exact; il existe alors un quotient Q et un reste de la division R . De préférence, d'utiliser:

```
>> [ Q , R ] = doconv ( p , p2 )
Q =
      1      3
```

R =
0 0 0 0

1.3. Expansion en fractions partielles

Une autre opération très utile est la décomposition en éléments simples d'une fraction rationnelle. Soit une fraction rationnelle $G(s)$ strictement propre qui possède n pôles $\{\alpha_i; i = 1, \dots, n\}$ réels ou complexes:

$$G(s) = \frac{B(s)}{A(s)} = \frac{b_m s^m + b_{m-1} s^{m-1} + \dots + b_1 s^1 + b_0}{a_n s^n + a_{n-1} s^{n-1} + \dots + a_1 s^1 + a_0}$$

Lorsque tous les pôles sont distincts, nous pouvons représenter $\frac{B(s)}{A(s)}$ de la façon suivante dans laquelle les $\frac{r_i}{s-p_i}$ sont les « *éléments simples* » et les r_i sont les « *résidus* »:

$$\frac{B(s)}{A(s)} = \frac{r_1}{s-p_1} + \frac{r_2}{s-p_2} + \dots + \frac{r_i}{s-p_i} + \dots + \frac{r_n}{s-p_n} + K(s)$$

Voici un exemple d'utilisation de la fraction rationnelle suivante:

$$\frac{s^2 + 2s + 3}{s^4 - 8s^3 + 23s^2 - 28s + 12}$$

Nous pouvons faire la décomposition en éléments simples à l'aide de la commande « `residue` », comme suit:

```
>> B= [ 1 2 3 ] % Création du polynôme B
>> A= [ 1 -8 23 -28 12 ] % Création du polynôme A
>> [ R , P, K ] = residue ( B , A)
R =
    9.0000
   -6.0000
  -11.0000
   -3.0000
P =
    3.0000
    2.0000
    2.0000
    1.0000
K =
    [ ]
```

Ce qui signifie que la décomposition en éléments simples s'écrit (voir l'aide de *Matlab* pour cette fonction) comme suit:

$$\frac{9}{s-3} - \frac{6}{s-2} - \frac{11}{s-2} - \frac{3}{s-1} + 0$$

K est le reste, qui est nul pour ce cas.

1.4. Racines d'un polynôme

Nous obtenons les racines du polynôme p grâce à l'instruction « `roots(A)` ».

Exemple: Soit le polynôme $p(s) = s^2 + 2s + 5$, pour déterminer les racines du polynôme p , nous écrivons:

```
>> A = [ 1 2 5 ] ;
>> roots (A)
ans =
    -1.0000 + 2.0000i
    -1.0000 - 2.0000i
>>
```

Ce polynôme possède deux racines complexes, puisque $\Delta = b^2 - 4ac$ est inférieur à 0. Dans ce cas, *Matlab* retourne un résultat exprimé en nombres complexes.

2. Résolution de systèmes linéaires avec *Matlab*

La commande *Matlab* « \ » (back slash) est la commande la plus utilisée pour résoudre les systèmes linéaires. Cet algorithme est lié à la structure de la matrice A du système linéaire. *Matlab* utilise dans l'ordre les méthodes suivantes:

- Si A est une matrice triangulaire, le système est résolu par simple substitution.
- Si A est une matrice symétrique ou hermitienne, définie positive, la résolution est effectuée par la méthode de *Choleski*.
- Si A est une matrice carrée mais n'entrant pas dans les deux cas précédents, une factorisation $L U$ est réalisée en utilisant la méthode d'élimination de *Gauss* avec stratégie de pivot-partiel.
- Si A n'est pas une matrice carrée, la méthode QR est donc utilisée.

Exemple:

```
>> A = [ 1 2 ; 3 4 ]; b = [ 1 1 ]';
>> x = A \ b
x=
```

```
          -1
          1
>> A * x
ans =
          1
          1
>>
```


Chapitre 03: Les Fichiers Scripts et les Fonctions

1. Introduction

Au lieu de saisir les commandes individuellement et directement dans la fenêtre de commandes, il est possible de créer un fichier appelé "M-file" qui contient toutes les fonctions et commandes nécessaires et qui peut être rapidement exécuté en tapant le nom du fichier dans la fenêtre de commandes. Ce nom provient du fait que l'extension est ".M". La fenêtre "Edit Window" est utilisée pour créer ou modifier les "M-files". La majorité des travaux avec *Matlab* seront liés à la manipulation de ces fichiers. Il y a deux types de fichiers .M: les fichiers de commandes (fichiers scripts) et les fichiers de fonctions.

2. Les fichiers de commandes (Scripts)

Un script est un ensemble d'instructions *Matlab* qui joue le rôle d'un programme principal. Si le script est écrit dans un fichier dont son nom est « *prog1.m* », par exemple, nous l'exécutons dans la fenêtre *Matlab* en tapant « prog1 ». Il est, en effet, beaucoup plus simple de modifier des instructions dans un fichier à l'aide d'un éditeur de texte que de retaper un ensemble d'instructions *Matlab* dans la fenêtre de commandes.

- Pour créer un nouveau fichier, allez dans le menu de sélection à:

"File/New/M-file"

- Pour ouvrir un fichier déjà créé, allez à:

"File/Open" et choisissez le nom du fichier en question.

La fenêtre "Edit Window" peut être vue comme un éditeur de texte où:

- Les commentaires sont écrits en vert et débutent par "%".
- Les variables et les équations apparaissent en noir.
- Les caractères apparaissent en violet.
- Les mots-clés dans *Matlab* comme les boucles apparaissent en bleu.

Exemple:

```
% Ceci est un commentaire
x = 1 : 5 ;
y = 3*x + 2 ;

dips ('Bonjour tout le monde')

for

end
```

À noter que les "M-files" sont exécutés en tapant le nom du fichier dans la fenêtre de commandes.

ATTENTION:

Il faut toujours s'assurer que nous travaillons dans le bon répertoire, là où notre fichier est sauvegardé, sinon nous obtiendrons des erreurs:

**3. Les fichiers de fonctions**

Les fichiers de fonctions ont deux rôles. (i) Ils permettent à l'utilisateur de définir des fonctions qui ne figurent pas parmi les fonctions incorporées à *Matlab* (built-in functions) et (ii) de les utiliser de la même manière que ces dernières (ces fonctions sont nommées: fonctions de l'utilisateur). Ils sont également un élément important dans la programmation d'applications puisqu'elles jouent le rôle des fonctions et procédures des langages de programmation usuels.

Nous définissons une fonction **func** de la manière suivante:

```
function [vars_1,...,vars_m] = func( vare_1,..., vare_n )
Séquence d'instructions ;
```

Où: *vars_1,..., vars_m* sont les variables de sortie de la fonction;

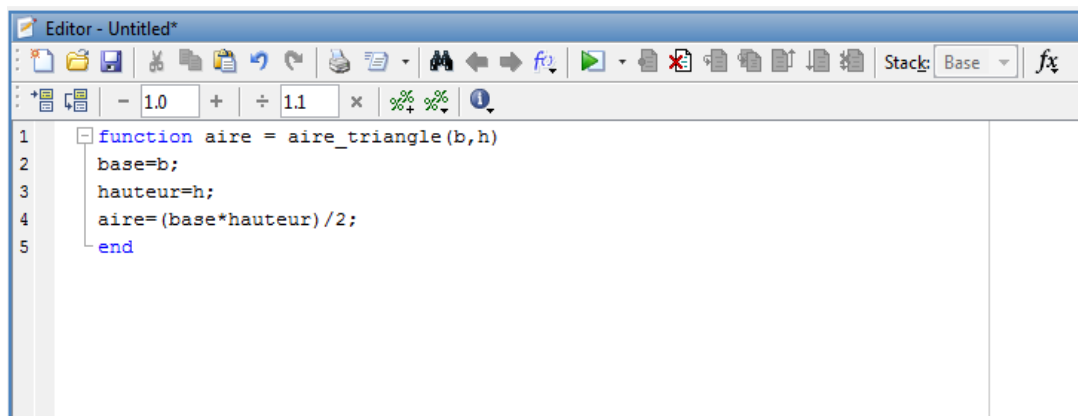
Et: *vare_1,..., vare_n* sont les variables d'entrée de la fonction;

Séquence d'instructions est le corps de la fonction.

Le fichier de la fonction doit impérativement commencer par le mot-clé « *function* », suivi entre crochets par les « variables de sortie » de la fonction, le symbole « = », le « nom de la fonction » et enfin les « variables d'entrée » entre parenthèses.

Si la fonction ne possède qu'une seule variable de sortie, dans ce cas les crochets sont inutiles. La fonction ayant le nom « *fonc* » doit être enregistrée dans un fichier nommé « *fonc.m* » (ayant le même nom de la fonction) pour qu'elle devienne visible par *Matlab* dans des traitements ultérieurs.

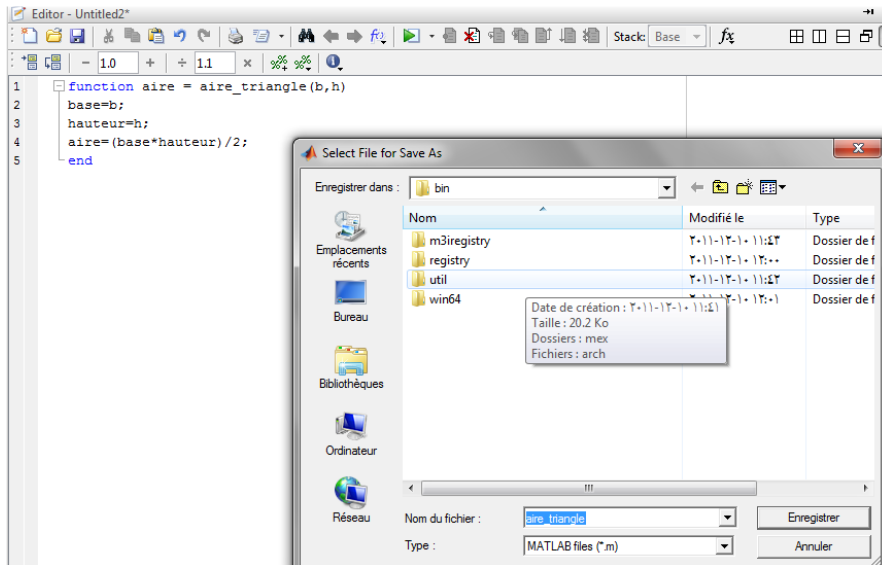
Dans l'exemple qui suit, nous définissons une fonction « *aire_triangle* » qui calcule l'aire d'un triangle:



```
1 function aire = aire_triangle(b,h)
2     base=b;
3     hauteur=h;
4     aire=(base*hauteur)/2;
5 end
```

Une fois le fichier est complété, nous devons sauvegarder ce fichier en sélectionnant dans le menu: "**File/save**"

Nous verrons automatiquement le nom de la fonction apparaissant au bas de notre écran: "**aire_triangle.m**", ensuite nous cliquons sur "**save**" :



Maintenant, nous pouvons convoquer cette fonction à partir de la fenêtre de commandes quand nous voulons calculer l'aire d'un triangle. Notons qu'en convoquant cette fonction, il faudrait y passer deux arguments comme paramètres: la longueur de la base du triangle et la hauteur, comme présenté dans cet exemple :

```
>> test = aire_triangle (10,5)

test =

    25
```

Exemple 02 : Ecrire une fonction permettant de calculer: $(T_1, T_2) = f(x, y, z)$, avec:

$$T_1 = x^2 + 3y - 9z \quad \text{et} \quad T_2 = y^6 + 2z$$

Solution :

```
function [T1 ,T2 ] = ma_fonction ( x ,y, z )
    T1 = x^2 + 3*y - 9*z ;
    T2 = y^6 + 2*z ;
end
```

```
>> [ A , B ] = ma_fonction ( 3 , 1 , 2)

A =

    - 6

B =

    5
```

4. Les principales fonctions prédéfinis sur *Matlab*

Parmi les fonctions fréquemment utilisées et prédéfinis sur *Matlab*, nous pouvons citer:

- $\sin(x)$: le sinus de x (en radian).
- $\cos(x)$: le cosinus de x (en radian).
- $\tan(x)$: le tangent de x (en radian).
- $\text{asin}(x)$: l'arc sinus de x (en radian).
- $\text{acos}(x)$: l'arc cosinus de x (en radian).
- $\text{atan}(x)$: l'arc tangent de x (en radian).
- $\text{sqrt}(x)$: la racine carrée de x (\sqrt{x}).
- $\text{abs}(x)$: la valeur absolue de x ($|x|$).
- $\text{exp}(x)$: e^x .
- $\log(x)$: logarithme naturel de x ($\ln(x)$).
- $\log_{10}(x)$: logarithme à base 10 de x ($\log_{10}(x)$).
- $\text{imag}(x)$: la partie imaginaire du nombre complexe x .
- $\text{real}(x)$: la partie réelle du nombre complexe x .
- $\text{round}(x)$: arrondi un nombre vers l'entier le plus proche.
- $\text{floor}(x)$: arrondi un nombre vers l'entier le plus petit.
- $\text{ceil}(x)$: arrondi un nombre vers l'entier le plus grand.

Chapitre 04: Instructions de contrôle (Boucles « *for* » et « *While* », Instructions « *if* » et « *else* »)

Similaire aux autres langages de programmation, *Matlab* peut contrôler la séquence des instructions d'un programme. Il y a trois façons à faire :

1. La boucle « *for* »

Matlab peut répéter un ensemble d'instructions plusieurs fois, en utilisant l'instruction « *for* ». La forme générale de la boucle est alors:

```
for variable = expression;
    instructions;
end;
```

En effet, « *expression* » est une matrice contenant des nombres.

Exemple 01: Générer la matrice *A* telle que:

$$A_{ij} = i^2 - j^3 + 1, \text{ pour } i = 1, \dots, 20 \text{ et } j = 1, \dots, 10$$

Le code *Matlab* correspondant est:

```
for i = 1 : 1 : 20;
    for j = 1 : 1 : 10;
        A ( i , j ) = i^2 - j^3 + 1;
    end;
end;

% i va de 1 à 20 avec un pas de 1: for i = init: step: final
% j va de 1 à 10 avec un pas de 1: for j = init: step: final
```

Exemple 02: Calculer A_{ij}^b , pour *b* allant de 0 à 1 avec un pas de 0.1, et afficher le résultat à chaque itération. Le code est:

```
A = [1 2 ; 2 1];
for b = 0 : 0.1 : 1; % i va de 0 à 1 avec un pas de 0.1
    C = A.^b ;
    disp (C) ;      % Afficher C
end;
```

2. La boucle « while »

Matlab peut aussi utiliser des boucles pour répéter des instructions jusqu'à ce qu'une condition terminale soit satisfaite. La syntaxe générale des boucles avec « *while* » est la suivante:

```
while condition;
    instructions;
end;
```

Tant que « *condition* » est satisfaite, les « *instructions* » seront exécutées.

Exemple 01: Trouver un point fixe de la relation dynamique suivante (pas de solution analytique):

$$x_{t+1} = 1 + x_t^{0.2}$$

Tout ce que nous pouvons à faire, c'est d'itérer sur cette relation et d'arrêter lorsque la différence en valeur absolue entre deux itérations est inférieure à un seuil de tolérance. Les codes sont:

```
tol = 1e - 6; % critère de tolérance
crit = 1; % initialisation du critère
x0 = 1; % valeur initiale de x
while crit > tol;
    x = 1 + x0 ^ 0.2; % relation dynamique
    crit = abs (x - x0); % valeur absolue entre deux itérations
    % consécutives
    x0 = x; % la nouvelle valeur de x(t)
end;
```

3. La boucle avec l'alternative « if »

L'instruction « if » exécute un ensemble de commandes si une condition est satisfaite. La syntaxe générale est:

```
if condition;
    commande_1;
else
    commande_2;
end;
```

Exemple 01: Savoir si un entier est impair ou non. Les codes sont alors les suivants:

```
x = input ('entrer un entier:'); % demande à l'utilisateur
                                %d'entrer un numéro
                                % et le stock dans la variable x
if rem (x,2) == 0;              % si le reste de la division par
                                % 2 est nul alors
    disp ('x est pair'); % afficher le message: x est pair
else                             % sinon
    disp ('x est pair'); % afficher le message: x est impair
end;                               % fin du test
```

Ces commandes peuvent être enrichies de la façon suivante:

```
if condition;
    commande_1;
elseif condition;
    commande_2;
else
    commande_3;
end;
```

Exemple 02: Construire une fonction de la forme:

$$D(P) = \begin{cases} 0 & \text{si } p \leq 2 \\ 1 - 0.5 P & \text{si } 2 < p \leq 3 \\ 2P^{-2} & \text{sinon} \end{cases}$$

Les codes sont les suivants:


```

p = input ('entrer un chiffre: ');
if P <= 2;
    D = 0;
elseif (P > 2) & (P <= 3);
    D = 1 - (0.5 * P);
else
    D = 2 * P ^ (-2);
end;
disp (D);

```

Ces instructions peuvent évidemment être combinées les unes avec les autres. L'exemple 03 donne une illustration de la combinaison entre les instructions « `for` » et « `if` », en utilisant l'instruction « `break` » qui permet de sortir d'une boucle avant la fin de celle-ci.

Exemple 03: Construire la séquence suivante: $x_t = x_{t-1}^{0.4} - x_{t-1}^{0.2}$

Pour $t = 1, \dots, 100$. Nous devons nous assurer que x reste positif. *Matlab* trouvera une valeur complexe ! Les codes suivants permettent de générer cette séquence et nous font sortir de la boucle dès que le résultat d'une itération est négatif:

```

x(1) = 2 ;
for i = 2 : 1 : 100
    y = x(i-1) ^ 0.4 - x(i-1) ^ 0.2;
    if y < 0;
        disp (' le résultat ne peut pas être négatif');
        break;
    end;
    x(i) = y;
end

```

Avec cet ensemble de trois instructions, il est normalement possible de résoudre tous les problèmes numériques à l'aide de *Matlab*.

Chapitre 05: Graphisme (Gestion de Fenêtres Graphiques, Plot)

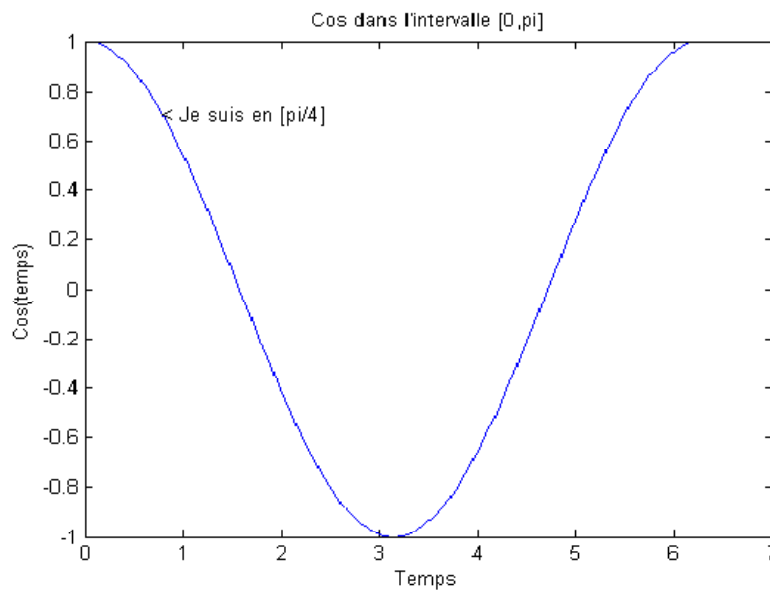
1. Tracer un graphique / courbe simple

Matlab peut produire des graphiques couleurs à deux dimensions, il offre aussi des outils et moyens pour personnaliser et modifier pratiquement leurs aspects, de manière facile et parfaitement contrôlée.

Pour tracer une courbe avec *Matlab*, nous devons utiliser la fonction « `plot` » qui peut avoir en paramètres (entre parenthèses) un ou plusieurs éléments. *Matlab* ne peut travailler qu'avec des valeurs discrètes, donc il faut d'abord commencer par définir l'intervalle de valeurs de l'abscisse (vecteur t). Une fois ceci effectué, nous calculons la valeur de la fonction (ex., un cosinus) pour chaque échantillon de temps afin de trouver le vecteur y .

```
% Graphique %
t = 0: pi/30: 2*pi      ; % Abscisse de 0 à 2pi par pas de pi/30
y = cos(t);           % Valeurs de la fonction dans le temps
figure;               % Nouvelle fenêtre
plot(t,y)             % Tracer
title ('Cos dans l'intervalle [0,pi]') ; % Titre
xlabel ('Temps');     % Légende en abscisse
ylabel ('Cos (temps)'); % Légende en ordonnée
text (pi/4, cos(pi/4), '< Je suis en [pi/4]') % Chaîne de caractères
à la position x,y
```

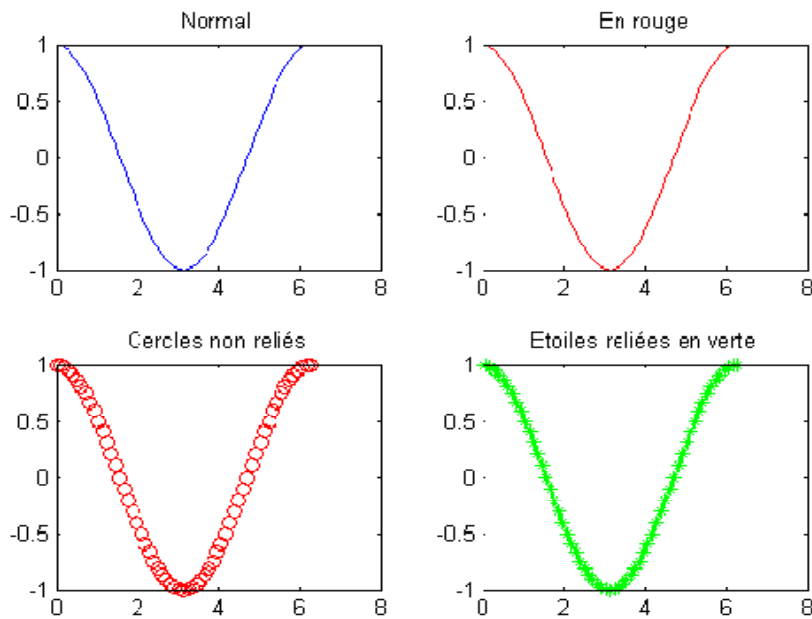
Résultat :



2. Graphiques avancées

Avec *Matlab*, nous pouvons personnaliser les graphiques d'une manière très simple. Par exemple, il est possible de changer la couleur du graphe, de représenter les points par différents symboles, tels que: x,o,*, ... etc. Le programme suivant illustre un aperçu simple de ce qui est possible à faire.

```
% Graphique avancé %
figure; % Nouvelle fenêtre
t = 0: pi/30: 2*pi; % Définition de temps
y = cos(t); % Fonction du temps
subplot (2,2,1) % Zone supérieure gauche
plot (t,y) ; % Tracer
title ('Normal'); % Titre
subplot (2,2,2) % Zone supérieure droit
plot (t,y,'r'); % Tracé en rouge
title ('En rouge'); % Titre
subplot (2,2,3) % Zone inférieur gauche
plot (t,y,'or'); % Tracé points ronds en rouge
title ('Cercles non reliés'); % Titre
subplot (2,2,4) % Zone inférieur droite
plot (t,y,'*-g'); % Tracé point étoiles relié en vert
title ('Cercles en vertes); % Titre
```



3. Titres et étiquettes

Les principales fonctions d'édition des graphiques se résument à:

<code>title('titre')</code>	Pour afficher le titre
<code>xlabel('étiquette')</code>	Pour assigner une étiquette à l'axe des x
<code>ylabel('étiquette')</code>	Pour assigner une étiquette à l'axe des y
<code>text(x,y,'texte à ajouter')</code>	Pour afficher le texte à la coordonné (x,y)

– Axes

Lors de la création d'un graphique, il est possible de spécifier les limites des axes x, y et même de z. La fonction « `axis` » permet de spécifier les limites des axes x,y (et z) en spécifiant un vecteur de paramètres.

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

De même, pour les graphiques à trois dimensions:

```
>> axis([xmin xmax ymin ymax zmin zmax])
```

Matlab contient aussi des fonctions spéciales permettant d'ajuster les axes à une figure carrée :

```
>> axis square
```

D'ajuster les axes à une même échelle :

```
>> axis equal
```

Et de remettre les axes à leurs valeurs calculées automatiquement :

```
>> axis auto normal
```

Les fonctions suivantes permettent d'activer la visibilité des axes et de la grille:

```
>> axis on, axis off, grid on, grid off
```

– Styles et couleurs

Pour différencier les courbes d'un graphique, il est possible de changer la couleur, le style des marqueurs et le style de courbe. Pour assigner les styles, nous utilisons la fonction «`plot`» en ajoutant en paramètre la couleur et le style du marqueur. La commande suivante permettant de créer une courbe $y = f(x)$ de couleur *cyan* et des marqueurs de type *+*;

```
>> plot(x, y, 'c', '+');
```

Les couleurs possibles sont :

'c'	Cyan
'm'	Magenta
'y'	Jaune
'r'	Rouge
'g'	Vert
'b'	Bleu
'w'	Blanc
'n'	Noir

Les styles de courbes:

'-'	Ligne pleine
'--'	Ligne discrète
'.'	Ligne pointillée
'none'	Aucune ligne

Et les types de marqueurs :

'+'	Marqueur de type croix
'o'	Marqueur rond
'*'	Marqueur en étoile
'x'	Marqueur de type X
's'	Marqueur carrés
'd'	Marqueur losange
'^'	Marqueur triangulaire vers le haut
'v'	Marqueur triangulaire vers le bas
'>'	Marqueur triangulaire vers la droite
'<'	Marqueur triangulaire vers la gauche
'p'	Marqueur pentagonal
'h'	Marqueur hexagonal
'none'	Aucun marqueur

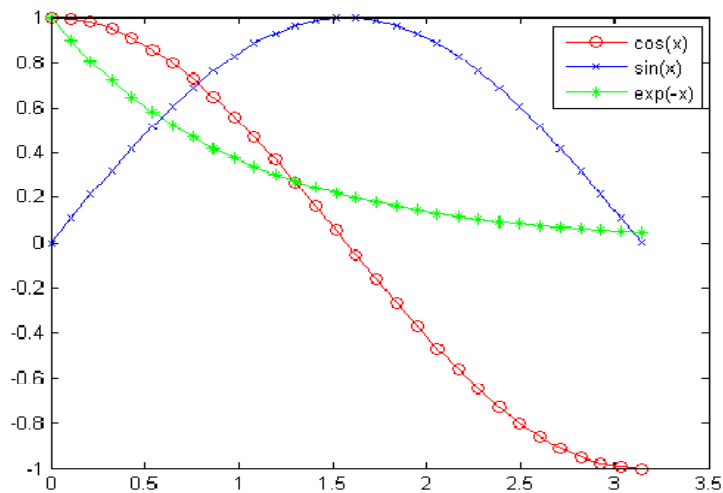
4. Graphiques multiples

Avec *Matlab*, nous pouvons tracer plusieurs courbes simultanément sur le même graphique. Cela peut être réalisé en mettant l'ensemble de fonctions à tracer dans les parenthèses (paramètres) qui suivent la commande « `plot` ».

Exemple:

```
% Graphique multiple 1 %
figure; % Nouvelle fenêtre
x = linspace (0, pi, 30); % Fonction du temps
plot (x, cos(x) , 'o-r', x, sin(x), 'x-b', x, exp(-x), '*-g') % Tracer
legend ('cos(x)', 'sin(x)', 'exp(-x) % Légende sur graphique
```

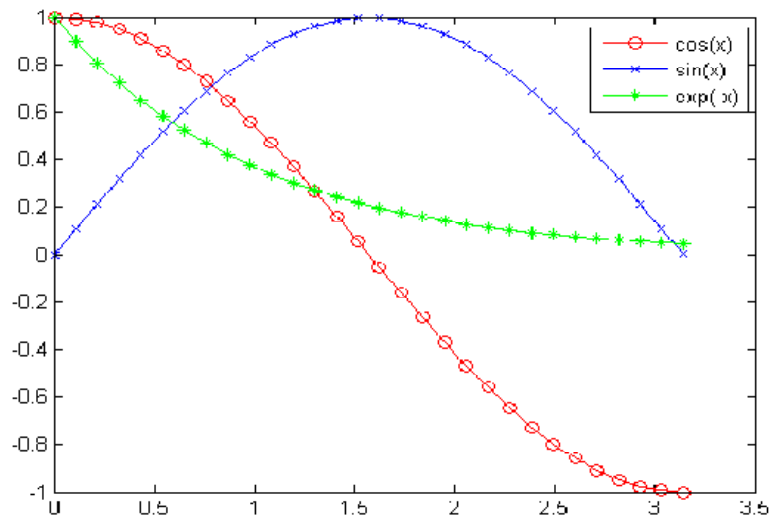
Résultat:



Une autre méthode permettant de tracer plusieurs courbes sur le même graphique consiste à activer la commande «[Hold](#)», ce qui stipule à *Matlab* de dessiner les graphiques les uns après les autres.

```
% Graphique multiple 2 %
figure; % Nouvelle fenêtre
x = linspace (0, pi, 30); % Fonction du temps
hold on % Maintient du graphique
plot (x, cos(x), 'o-r') % Tracer points ronds reliés en rouge
plot (x, sin(x), 'x-b') % Tracer points croix reliés en vert
plot (x, exp(-x), '*-g') % Tracer points étoiles reliés en bleu
legend ('cos(x)', 'sin(x)', 'exp(-x)') % Légende sur graphique
```

Résultat:



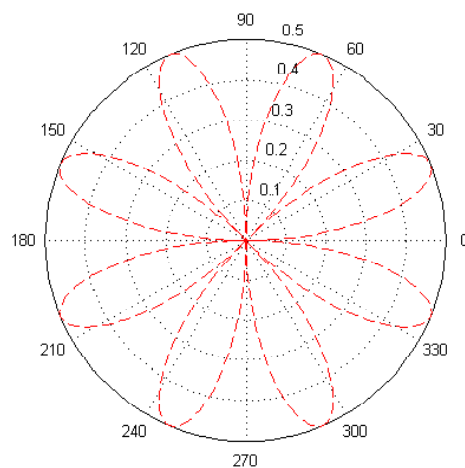
5. Graphiques polaires

Avec *Matlab*, il est aussi possible de dessiner et tracer des graphiques à coordonnées polaires. Cela peut être effectué en utilisant la fonction « `polar` » au lieu de « `plot` ».

Par exemple :

```
% Graphique polaire %
figure;                               % Nouvelle fenêtre
t = 0: 0.1: 2*pi;                      % Echelle du temps
polar (t, sin(2*t).*cos(2*t), 'r')     % Graphique polaire en
pointillé rouge
```

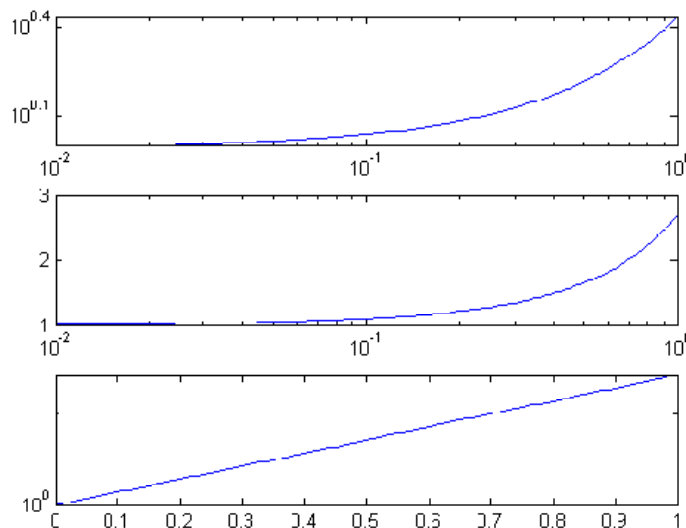
Résultat :



6. Graphiques à échelles logarithmiques

Avec *Matlab*, Il est aussi possible de dessiner et tracer des graphiques avec un ou plusieurs axes sous forme (à échelle) logarithmique. Par exemple :

```
% Graphique à échelles logarithmiques %
figure;                               % Nouvelle fenêtre
x = 0: 0.01: 1;
subplot (3,1,1);                       % Zone supérieure
loglog (x, exp(x));                    % Echelle logarithmique deux axes
subplot (3,1,2)                         % Zone centrale
semilogx (x, exp(x))                   % Echelle logarithmique sur l'axe Ox
subplot (3,1,3)                         % Zone inférieure
semilogy (x, exp(x))                  % Echelle logarithmique sur l'axe Oy
```



7. Graphiques à 3 dimensions

Matlab capable de faire des représentations pour graphique à trois dimensions. Pour ce faire, il faut passer en paramètre de la fonction « `meshc` » trois matrices représentant les coordonnées selon les trois axes des points de la fonction.

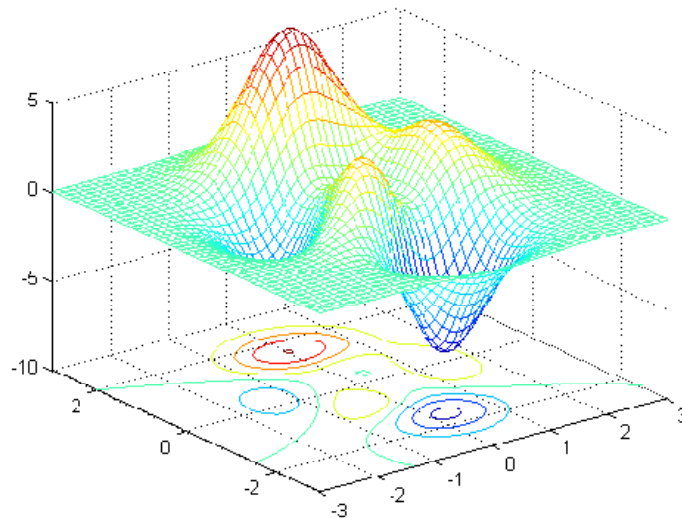
Dans l'exemple suivant, les matrices X et Y sont construites à l'aide de la fonction « `meshgrid` » de *Matlab* qui permet de définir une grille carrée. Pour la matrice Z , elle est définie à l'aide de la fonction « `peaks` » qui représente une *Gaussienne* en fonction des matrices X et Y .

```

% Graphique 3 dimensions %
figure; % Nouvelle fenêtre
[X,Y] = meshgrid (-3: 0.125: 3); % Génération d'une grille
de -3 à 3 pas de 0.125
Z = peaks (X,Y); % Distribution gaussienne en Z
meshc (X, Y, Z); % Affichage 3 dimensions
axis ([-3 3 -3 3 -10 5]) % Etalonnage des axes

```

Résultat :

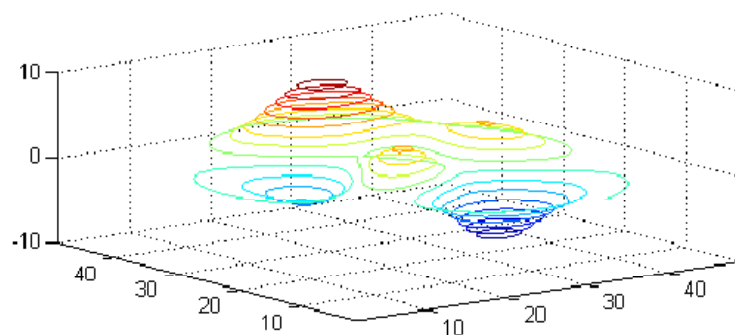
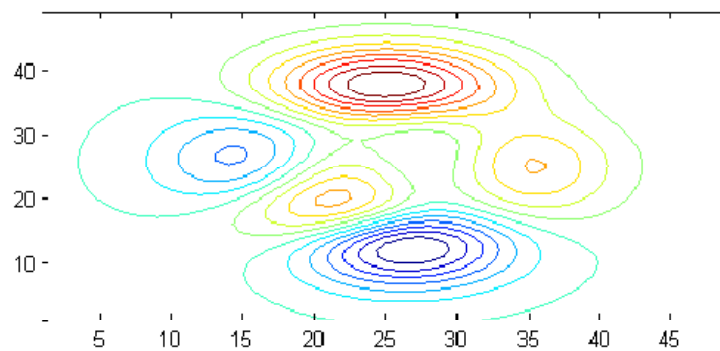
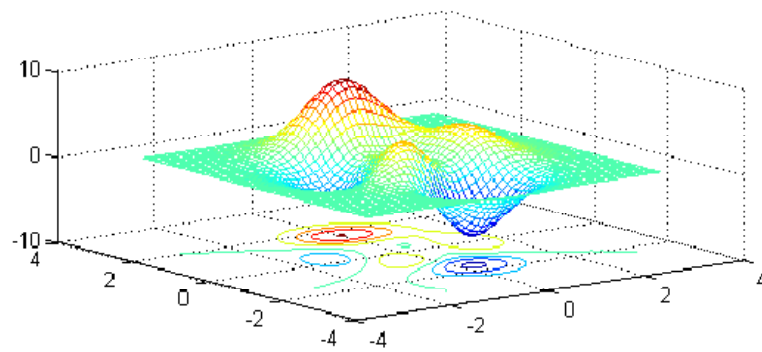


Il est aussi possible de représenter les contours des courbes à trois dimensions, en utilisant la fonction « `contour` » (Pour des contours à 3D, en utilisant « `contour3` »). Ces derniers relient les points de la même hauteur (valeur) par des courbes. De plus, la couleur de chaque courbe est dépendante à la valeur qu'elle représente. Par exemple :

```

% Graphique de Contours 3 dimensions %
Figure; % Nouvelle fenêtre
[X,Y] = meshgrid (-3: 0.125: 3) ; % Génération d'une grille
de -3 _a 3 pas de 0.125
Z = peaks (X,Y); % Distribution gaussienne en Z
subplot (3,1,1); % Zone supérieure
meshc (X,Y,Z); % Affichage 3 dimensions
subplot (3,1,2) % Zone centrale
contour (Z,15) % Contour 15 niveaux
subplot (3,1,3) % Zone inférieure
contour3(Z,15) % Contour 3 dimensions 15 niveaux

```



Enfin, il est possible, avec *Matlab*, de modéliser des surfaces à trois dimensions ainsi que leurs normales. Par exemple :

```
% Normales de surfaces %
figure; % Nouvelle Fenêtre
[x,y,z] = cylinder (1 :10); % Création d'un cylindre 3 dimensions
surfnorm (x,y,z) % Affichage des surfaces et normes
axis([-12 12 -12 12 -0.1 1]) % Etalonnage des axes
```

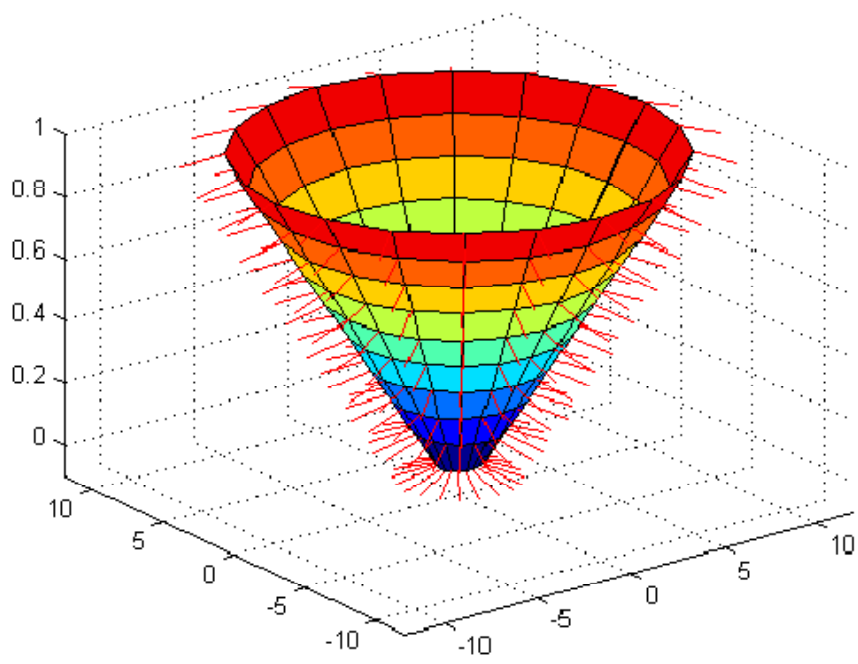


Table des Matières

Chapitre 01: Présentation de l'Environnement de Programmation

Scientifique MATLAB

1. Introduction à MATLAB	4
1.1. Introduction.....	4
1.2. Démarrage & Environnement.....	4
1.3. Obtenir de l'aide avec Matlab	6
2. Les premiers calculs en <i>Matlab</i> : Variables, Vecteurs, et Matrices.....	8
2.1. Variables	8
2.2. Lecture et affichage de données.....	11
2.3. Vecteurs	12
a. Création et manipulation d'un vecteur.....	12
b. Opérations sur les vecteurs.....	14
c. Vecteurs particuliers	14
2.4. Matrices.....	15
a. Création et manipulation d'une matrice.....	15
b. Matrices particulière.....	15
2.5. Manipulations des vecteurs et matrices.....	15
2.6. Concaténation et fonctions spécifiques aux vecteurs et matrices.....	19
2.7. Opérations matricielles sur les vecteurs et les matrices.....	21

Chapitre 02: Polynômes et Résolution de Systèmes Linéaires

1. Polynômes.....	24
1.1. Construction d'un polynôme.....	24
1.2. Multiplication et division de polynômes.....	25
1.3. Expansion en fractions partielles.....	26
1.4. Racines d'un polynôme.....	27
2. Résolution de systèmes linéaires avec Matlab.....	27

Chapitre 03: Les Fichiers Scripts et les Fonctions

1. Introduction.....	29
2. Les fichiers de commandes (Scripts).....	29
3. Les fichiers de fonctions.....	30

-
4. Les principales fonctions prédéfinis sur *Matlab*.....33

Chapitre 04: Instructions de contrôle (Boucles « *for* » et « *While* »,

Instructions « *if* » et « *else* »)

1. La boucle « *for* ».....34
2. La boucle « *while* ».....35
3. La boucle avec « *if* ».....36

Chapitre 05: Graphisme (Gestion de Fenêtres Graphiques, Plot)

1. Tracer un graphique / courbe simple.....38
2. Graphiques avancées.....39
3. Titres et étiquettes.....40
4. Graphiques multiples.....42
5. Graphiques polaires.....44
6. Graphiques à échelles logarithmiques.....45
7. Graphiques à 3 dimensions.....45
-