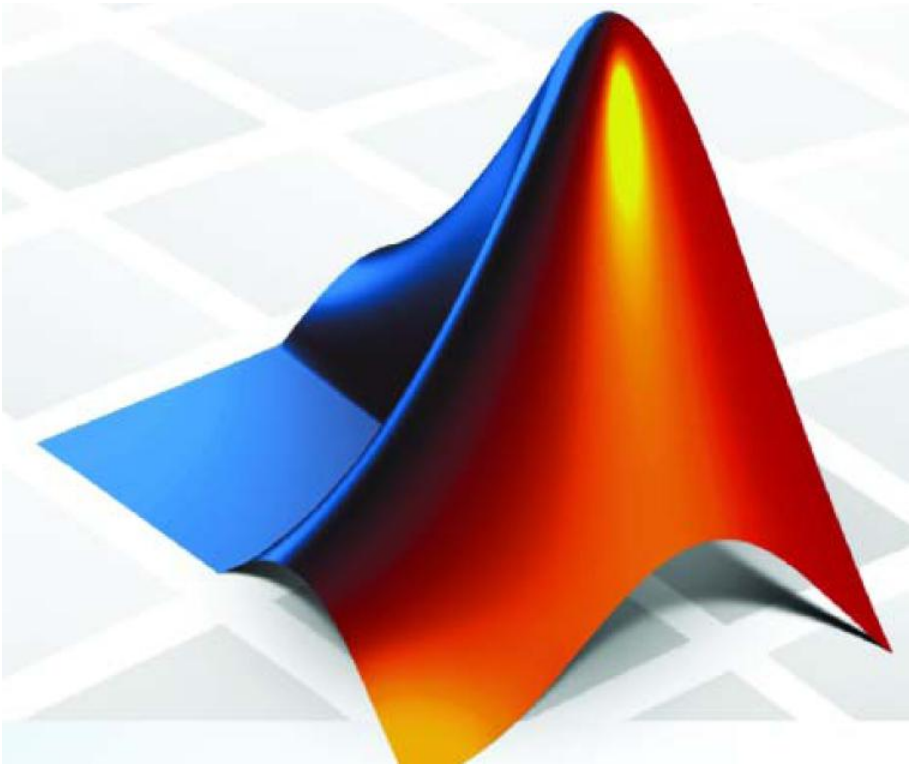
Université Sidi Mohammed Ben Abdallah
Faculté Des Sciences et Techniques Fès
Département Génie Electrique

LICENCE ETI

Electronique, Télécommunications et Informatique

Introduction à MATLAB et Simulink



Pr. Abdellah MECHAQRANE

Année Universitaire 2011-2012

Partie I

Les bases de programmation avec MATLAB

1. But :

Apprendre les bases de programmation à l'aide du logiciel MATLAB.

Exploitation pour ETI :

- Ondes &Transferts Thermiques (S4)
- Traitement du signal& de l'information (S5)
- Automatique (S6)
- …

2. Introduction à Matlab

MATLAB (abréviation de**MATrixLABoratory**) est un environnement puissant, complet et facile à utiliser destiné au calcul scientifique. Il intègre calcul numérique, visualisation (graphiques) et programmation dans un environnement facile à utiliser et où les problèmes et les solutions sont exprimés en notation mathématique familière.

3. Toolboxes :

MATLAB dispose de nombreuses "Toolboxes" (Boîtes à outils) qui sont des collections complètes des fonctions MATLAB (M-files) spécifiques à un domaine d'applications donné :

- Aerospace Toolbox,
- Communication Toolbox,
- Control system Toolbox,
- Image ProcessingToolbox,
- Signal ProcessingToolbox,
- …

4. Simulink

Matlab contient aussi l'environnement "Simulink" qui est un environnement puissant de modélisation basée sur les schémas-blocs et de simulation de systèmes dynamiques linéaires et non linéaires.

5. Quelques comparaisons avec d'autres langages de programmation :

- ✓ Langage de haut niveau,
- ✓ Environnement interactif,
- ✓ Exécution beaucoup plus rapide que C, C++ ou Fortran,
- ✓ Pas besoin d'intégration des bibliothèques comme en C :

```
# include <stdio.h>

# include <stdlib.h>
```

- ✓ Pas de dimensionnement des variables :

```
int,  float,  char,  …
```

6. Modes de fonctionnement:

Il existe deux modes de fonctionnement:

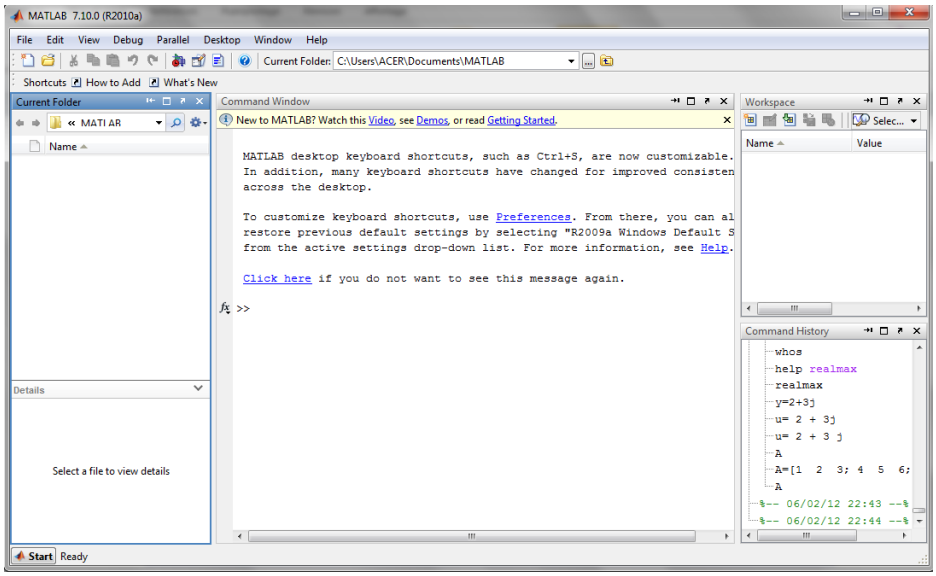
1. **mode interactif**: MATLAB exécute les instructions au fur et à mesure qu'elles sont données par l'usager.
2. **mode exécutif**: MATLAB exécute ligne par ligne un M-file (programme MATLAB dont l'extension est :"**.m**").

7. Environnement Matlab :

1. Présentation de l'environnement MATLAB en mode interactif :

1.1. Présentation de la fenêtre MATALB :

Sous Windows, on peut démarrer MATLAB en réalisant un double clic sur l'icône Matlab dans le bureau.On verra donc s'afficher l'interface suivante :



Par défaut, cette interface contient les fenêtres suivantes :

– Fenêtre des commandes (**Command Window**) : c'est la fenêtre du milieu dans laquelle on travaillera en entrant et exécutant les commandes.

– Historique des commandes (**Command History**) : fenêtre en bas à droite. Elle contient toutes les commandes déjà exécutées. Pour réexécuter une commande, il suffit de la glisser de la fenêtre "Command History" vers la fenêtre de commande "Command Window".

– Répertoire courant (**Current Directory**) : Contient les fichiers enregistrés dans le répertoire de travail actuel.

– Espace de travail (**Workspace**) : Contient l'ensemble des variables qui peuvent être accessibles à partir de la ligne de commande.

– Détails (**Details**) : donne quelques détails sur le fichier sélectionné dans la fenêtre "Current Directory ".

La fenêtre de commande est la fenêtre principale de MATLAB. Pour que cette fenêtre soit la seule visible, il suffit de sélectionner dans la barre de menu : **Desktop ⟶ Desktop Layout ⟶ Command Window Only**.

Pour revenir à l'interface par défaut, sélectionner dans la barre de menu :

Desktop ⟶ Desktop Layout ⟶ Default.

Les commandes Matlab sont tapées dans la "Fenêtre de commande" devant le prompt : **>>**.

1.2. Calculs élémentaires

a) Opérateurs de calcul

Fonctions	Définition
+	Addition
-	Soustraction
*	Multiplication
^(touches : Alt Gr + 9)	Puissance
/	Division
\(touches : Alt Gr + 8)	Division à gauche

Si nécessaire, on utilise des parenthèses.

b) Expressions mathématiques

Pour effectuer un calcul élémentaire, il suffit de taper une expression mathématique quelconque et valider en appuyant sur "Entrée" :

```
>> 3+2
```

Le résultat est mis automatiquement dans une variable appelée "ans"(answer). Celle-ci peut être utilisée pour le calcul suivant :

```
>> ans*2
>> (5-2)^3
>> (ans+2)/(4+2)
```

Remarque : "ans" prend la valeur du résultat de la dernière commande exécutée.

c) Format d'affichage

Par défaut, Matlab affiche les résultats numériques sous le format "short" (4 chiffres après la virgule). En fait, la précision de calcul est de 15 chiffres après la virgule (format long). Matlab peut aussi afficher les résultats sous le format hexadécimal et sous d'autres formats. Pour utiliser un format, il suffit de taper son nom et de valider.

```
>>format short
>> 20/3000
```

Contempler l'effet des formats suivants :

```
format short e
format short g
format long
format long e
format long g
format bank
format rat
formathex
```

d) Historique des commandes

C'est une fonctionnalité très utile, inspirée des shells UNIX modernes : toutes les commandesque vous aurez tapé sous MATLAB peuvent être retrouvées et éditées grâce aux touches de direction. Appuyez sur la touche "up"  pour remonter dans les commandes précédentes, sur la touche "down" pour redescendre.
Pour exécuter une commande, inutile de remettre le curseur à la fin, vous appuyez directementsur la touche «Entrée».
Vous pouvez retrouver toutes les commandes commençant par une lettre ou un groupede lettres. Par exemple pour retrouver toutes les commandes commençant par "f",tapezf, puis appuyez sur la touche "up" autant de fois que nécessaire.

1.3. Variables

Dans Matlab, on peut affecter une valeur ou un résultat de calcul à une variable. Un gros avantage sur les langagesclassiques : on ne déclare pas les variables : Leur type (entier, réel, complexe) et leur dimension s'affecterontautomatiquement.
Remarque : Matlab travaille avec un seul type de variable : **matrice**.

- **Scalaire :**Un scalaire est une matrice 1×1

```
>> a=1.5
>> b=3.2
>> c=-2
```

On peut inclure les variables déjà définies dans de nouvelles expressions mathématiques, pour en définir de nouvelles variables :

```
>> var1 = a+b
>> var_2 = var1^c
>>v_mult = e*var_2
>>v_div = v_mult/(b+a)
```

On peut aussi définir des nombres complexes :

```
>>v_complexe = 2+3i
```

Remarques :

→ Le nom d'une variable doit obligatoirement commencer par une lettre. Il peut contenir des chiffres et dessous-tirets. Il ne doit pas contenir d'opérateurs spéciaux (+, -, /, &, ., *, ^, …)

→ MATLAB utilise seulement les 31 premiers caractères d'un nom de variable.

→ MATLABfait la distinction entre majuscules et minuscules : A et a ne sont pas la même variable.

- **Vecteurligne :**Un vecteur ligne de N éléments ≡ matrice 1xN

```
>> L1 = [1 2 3 4 5 6 7]
>> L1 = [1, 2, 3, 4, 5, 6, 7]
>> L2 = [-2, 1+i, 2-3i, 3]
```

Remarques :

- Dans un vecteur ligne, les éléments sont séparés par une virgule ou un espace.
- Les vecteurs lignes dont les éléments sont régulièrement espacés peuvent être générés par :

a) L'opérateur ":"

Cet opérateur est très important dans la programmation Matlabsurtout lorsqu'il s'agit de générer les valeurs d'une fonction pour la tracer sur un intervalle donné. Ilpermet de générer un vecteurligne formé de valeurs équidistantes entre deux valeurs extrêmes. La syntaxe générale est :

variable = valeur initiale :lepas: valeur finale

```
>> L13 = 1:7
>> L14 = 0:0.1:1
>> L3 = -10:2:0
>> L4 = 10:-2:0
```

Pour créer 101 valeurs équidistantes sur l'intervalle [0, 2π] :

```
>> x = 0: 2*pi/100 : 2*pi;
```

b) La fonction MATLAB "linspace" :

```
>> L1 = linspace(1,7,7)
```

Remarques :

- Pour voir la syntaxe de la fonction "**linspace**" tapez :

```
>>help  linspace
```

- Le point-virgule à la fin d'une commande élimine l'affichage du résultat de la commande, mais la commande s'exécute :

```
>> k = 0:10;
```

On peut afficher le vecteur k en exécutant la commande:

```
>> k
```

Ceci est très utile surtout lorsqu'on exécutera des programmes importants : l'affichage des résultats des instructions dans la fenêtre de commande retarde l'exécution des programmes.

- On peut aussi créer des valeurs réparties de manière logarithmique avec la fonction logspace :

```
>> help  logspace
```

- **vecteur colonne :** Un vecteur colonne de N éléments ≡ matrice Nx1

```
>>C1=[-2; 1; 3i; 2]
```

Le vecteur C1 peut être aussi créé de la façon suivante :

```
>> C1=[-2
        1
        3i
         2]
```

Remarques :

- ☞ Dans un vecteur colonne, les éléments sont séparés par un point-virgule.
- ☞ Un vecteur colonne peut être obtenu en appliquant la fonction transposé "**transp**" ou l'opérateur "**.'** "à un vecteur ligne:

```
>> C2 = transp(L2)
>> C2 = L2.'
```

- ☞ Extractions des éléments d'un vecteur :

```
>> a2 = L1(2)
>> a24 = L1(2:4)
>> c3 = C1(3)
>> c35 = C2(3:5)
```

Remarque :pour voir le contenu d'une variable, il suffit de taper son nom et de valider :

```
>> a2
```

Si vous tapez :

```
>> a3
```

vous recevrez le message d'erreur suivant :

```
??? Undefined function or variable 'a3'
```

Ce message vous indique que la fonction ou variable 'a3' n'a pas été définie auparavant.

- **Matrices :** Toute variable **NxM** éléments (N lignes et M colonnes)

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

La matrice A peut être créée de la manière suivante :

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

On peut aussi créer des matrice à l'aide de vecteurs lignes ou colonnes de la même longueur :

```
>> B = [L3;L4]
>> C = [L3' L4']
```

1.4. Accès aux éléments d'une variable

1.4.1. Si la variable est un scalaire ou un nombre complexe, on peut y accéder en tapant son nom et en validant :

```
>> a
>> v_complexe
```

1.4.2. Si la variable est un vecteur (ligne ou colonne), on peut accéder à la valeur de l'élément i, en tapant le nom du vecteur suivi du chiffre i entre parenthèses :

```
>> L1(5)
>> C1(3)
```

1.4.3. Si la variable est une matrice, on a deux manières d'accéder à ses éléments :

- L'élément dans la ligne i et la colonne j de A est noté A(i, j):

$$A=\begin{bmatrix} A(1,1) & A(1,2) & A(1,3) & A(1,4) \\ A(2,1) & A(2,2) & A(2,3) & A(2,4) \\ A(3,1) & A(3,2) & A(3,3) & A(3,4) \\ A(4,1) & A(4,2) & A(4,3) & A(4,4) \end{bmatrix}$$

- Les éléments de la matrice sont indexés par ordre suivant les colonnes :

$$A=\begin{bmatrix} A(1,) & A(5) & A(9) & A(13) \\ A(2) & A(6) & A(10) & A(14) \\ A(3) & A(7) & A(11) & A(15) \\ A(4) & A(8) & A(12) & A(16) \end{bmatrix}$$

Ainsi :

```
>> A(3,2)
>> A(7)
```

représentent le même élément de la matrice A.

L'instruction A(:) crée un vecteur colonne contenant les éléments de A énumérés par colonne.

1.5. Extractions des éléments d'une matrice :

Pour extraire un seul élément d'une matrice, on procède de la manière présentée dans le paragraphe si dessus. Mais pour extraire un vecteur ou une sous matrice, on utilise l'opérateur "**:**" :

- Extraction d'une ligne :

```
>> A_l2 = A(1,:)
```

- Extraction d'une colonne :

```
>> A_c2 = A(:,3)
```

- Extraction d'une sous-matrice :

```
>> A_sub = A(2:3,2:3)
```

1.6. Valeurs et matrices spéciales :

- **Constantes particulières :**

  - **pi**: valeurdela constante trigonométrique π.

```
>> pi
```

  - **eps** : le plus petit nombre pouvant séparer deux nombres consécutifs (2.2204e-016).

```
>> eps
```

  - **realmax:** la plus grande valeur pouvant être utilisée :

```
>> realmax
```

  - **realmin:** la plus petite valeur pouvant être utilisée :

```
>> realmin
```

  - **i, j :** représentent (indifféremment) le nombre complexe $\sqrt{-1}$

```
>> x = 3-2i
>> y = 3-2j
```

Remarque : Si le nombre i ou j se trouve devant une variable ou une fonction, on doit utiliser l'opérateur "*****" :

```
>> z1 = 2+i*c
>> z = 1+i*cos(pi/4)
```

- **Inf** : est la notation MATLAB pour l'infini (1/0)

```
>> 1/0
```

- **Nan** : (not-a-number), souvent le résultat d'une opération de 0/0.
- **clock** : affiche la date et l'heure courantes sous le format :

année mois jour heure mn s

- **date** : affiche la date courante.

- **Matrices particulières :**
  - **zeros** :

Permet de créer des vecteurs ou matrices composés uniquement par des zéros :

- **zeros(N)** : matrice NxN dont les éléments sont tous des zeros.

```
>> zeros(3)
```

- **zeros(N,M)** : matrice NxM dont les éléments sont tous des zeros.

```
>> zeros(1,3)
>> zeros(3,1)
>> zeros(3,3)
```

- **ones** :

Permet de créer des vecteurs ou matrices composés uniquement par des un :

- **ones(N)** : matrice NxN dont les éléments sont tous des 1.

```
>>ones(4)
```

- **ones(N,M)** : matrice NxM dont les éléments sont tous des 1.

```
>> ones(1,4)
>> ones(3,1)
>> ones(3,3)
```

➢ **eye**: produit des matrices identités :

- **eye(N)** : matrice identité de dimension NxN.

```
>> eye(4)
```

- **eye(N,M)** : matrice NxM avec des 1 sur la diagonale et des 0 ailleurs.

```
>> eye(1,4)
>> eye(3,1)
>> eye(3,4)
```

➢ **rand**: générer des matrices contenant des nombres aléatoires (entre 0 et 1) uniformément distribués.

- **rand(N)** : matrice de dimension NxN.

```
>> rand(4)
```

- **rand(N,M)** : matrice NxM avec des 1 sur la diagonale et des 0 ailleurs.

```
>> rand(1,7)
```

➢ **randn**: générer des matrices contenant des nombres aléatoires normalement distribués.

- **randn(N)** : matrice de dimension NxN.

```
>> randn(4)
```

- **randn(N,M)** : matrice NxM avec des 1 sur la diagonale et des 0 ailleurs.

```
>> randn(1,7)
```

➢ **magic**: générer des matrices magiques (somme des lignes, des colonnes, de la diagonale et de l'anti-diagonale sont les mêmes).

- **magic(N)** : matrice magique de dimension NxN (N>2).

```
>> M = magic(5)
>> sum(M)
>> sum(M')
>> sum(diag(M))
>> sum(diag(filplr(m)))
```

➢ **diag** : Appliquée à une matrice A, la fonction "diag" retourne un vecteur colonne formé par les éléments de la diagonale principale de A :

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

Appliqué à un vecteur, elle retourne la matrice diagonale (les éléments hors de la diagonale sont nuls) :

```
>> diag([1,2,3])
```

1.7. Matrices creuses

MATLAB est particulièrement gourmand en place mémoire, aussi a-t-il été prévu de ne stocker que les éléments non nuls des matrices. La manipulation s'effectue avec les commandes **sparse** et **full**. Ainsi, sA=**sparse**(A) génère une matrice sA qui ne comporte (en mémoire) que les éléments non nuls de A. On peut revenir à A par A = **full**(sA) :

```
>> A = eye(10)
>> sA = sparse(eye(10))
>> whos
```

Comparez les places mémoires occupées par A et As

- **spones** : remplace les éléments non-nuls d'une matrice creuse par des 1 :

```
>> S=[2 0 3;-1 3 0; 0 -5 4], spones(S)
```

- **speye** : crée une matrice identité creuse.

```
>> As = speye(10)
```

Crée une matrice identité carrée de 10 lignes et 10 colonnes. Cette commande est équivalente à **speye(10,10).**

1.8. Opérations sur les matrices

Dans Matlab, les calculs se font au sens matriciel.

Soient les éléments suivants :

```
>> v1 = [1  2  3]
>> v2 = [4  5  6]
>> v3 = [1; 2]
>> A = [1   2; 3   4]
>> B = [5   6; 7   8]
>> C = ones(2)
```

1.8.1. Addition, soustraction

- L'addition ou la soustraction d'un scalaire d'un vecteur ou d'une matrice revient à sostraire le sclaire de chaque élément du vecteur ou de la matrice :

```
>> v1-2
>> A-1
```

- L'addition ou la soustraction doivent se faire entre vecteurs ou entre matrices de mêmes dimensions :

```
>> v1 + v2
>> A - C
```

- Si les dimensions ne concordent pas :

```
>> C + v1
```

on obtient le message d'erreur suivant :

```
??? Error using ==> plus
Matrix dimensions must agree.
```

c'est-à-dire que les dimensions des deux entités utilisées dans l'addition (ou la soustraction) doivent concorder.

1.8.2. Multiplication et puissance

- La multiplication d'un vecteur ou d'une matrice par un scalaire se fait en multipliant chaque terme par le scalaire :

```
>> 2*v1
```

```
>> 3*A
```

- Le résultat de la multiplication d'un vecteur colonne de N éléments par un vecteur ligne de M éléments est une matrice NxM où chaque ligne est obtenue par la multiplication du vecteur ligne par l'élément de la ligne correspondante du vecteur colonne :

```
>> v3*v1
>> v1'*v2
```

- La multiplication entre vecteurs et/ou matrices se fait au sens matriciel (ligne par colonne) :

```
>> A*v3
>> v3'*A
>> A*C
```

- La puissance n^{ème} d'une matrice représente la multiplication n fois, au sens matriciel, de cette matrice par elle même :

```
>> A^2
>> A^3
```

- On peut effectuer la multiplication ou la puissance élément par élément (terme à terme) des vecteurs ou des matrices en utilisant les opérateurs "**.***" pour la multiplication et "**.^**" pour la puissance:

```
>> v1.*v2
>> A.*B
>> A.*A
>> A.^2
>> v2.^v1
>> B.^A
```

1.8.3. Division

- Dans MATLAB, on distingue deux opérateurs pour effectuer la division. L'opérateur " **/** " représente la division à droite alors que l'opérateur " **** " représente la divisionà gauche:

```
>> 3/4
>> 3\4
```

Ainsi, la division représente le produit par l'inverse de la variable située du côté vers lequel penche la barre :

$$3/4 \text{ représente } 3\times\frac{1}{4} \text{ alors que} 3\backslash 4 \text{ représente} \frac{1}{3}\times 4 .$$

- Pour les matrices, la division fait appelle à l'inverse de matrices. Ainsi, A/B représente A multiplié (au sens des matrices) à la matrice inverse de B :

```
>> A/B
>> A*inv(B)
```

- B\A représente le produit inv(B)*A :

```
>> B\A
```

NB : A/B qui est égale à A*inv(B) est différent de B\A qui est égale à inv(B)*A.

- Les divisions (à gauche ou à droite) élément par élément s'obtiennent en utilisant les opérateurs "**.**" ou "**./**" :

```
>> B./A
>> A.\B
```

1.8.4. Opérateur transposé et transposé conjugué :

L'opérateur transposé "**.'**" est utilisé pour transformer les vecteurs lignes en vecteurs colonnes ou les lignes d'une matrice en colonnes. L'opérateur transposé conjugué "**'**" a le même effet que l'opérateur transposé mais qui remplace en plus les termes complexes du vecteur ou matrice considéré par leur conjugué :

```
>> D = [1+i, 2, 3-i; 4, 5+2i, 6];
>> D_tc = D'
>> D_t = A.'
```

2. Espace de travail

Dans Matlab, l'epace de travail représente l'ensemble de variables qui ont été définies ou calculées dans une session. Cet ensemble peut être affiché à l'aide de commande "**who**". Pour obtenir plus d'informations sur chaque variable, on utilise la commande "**whos**".

On peut aussi obtenir les dimensions d'une matrice en utilisant la commande :

```
>> size(A)
```

La commande "**size(A)**" retorne un vecteur ligne de deux termes. Le prenier terme représente le nombre de lignes de A et le deuxième terme le nombre de colonnes.

La commande "**size(A,1)**" retourne le nombre de lignes de A et la commande "**size(A,2)**" le nombre de colonnes.

La commande "**length(V)**" est souvent utilisée pour déterminer la longueur d'un vecteur ligne ou colonne. Utilisée avec une matrice, la La commande "**length**" retourne la plus grande dimension de la matrice.

3. Suppression de lignes ou de colonnes d'une matrice

Soit la matrice :

```
>> X = [1  2  3  4; 5  6  7  8; 9  10  11  12];
```

Pour supprimer la 2ème colonne de la matrice X, il suffit d'utiliser un double crochet. La commande à exécuter est la suivante :

```
>> X( : , 2) = [ ]
```

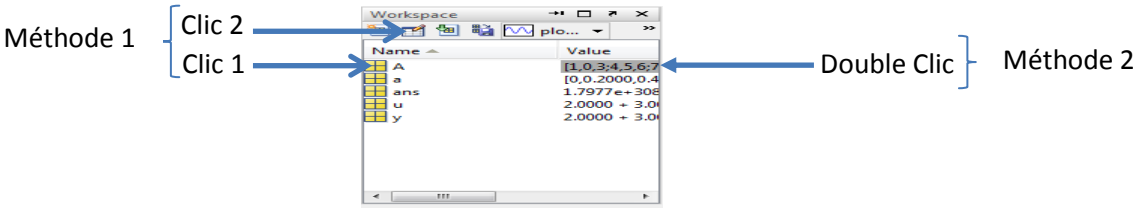
4. Remplacement d'un terme dans une matrice

Pour remplacer un terme dans une matrice, on peut procéder de deux manières :

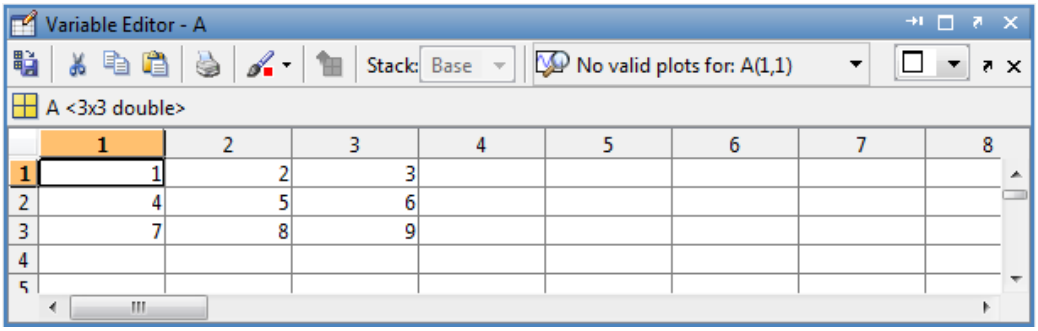
- A partir de la fenêtre de commande "Command Window", en exécutant une commande comme la suivante :

```
>> X(1,2) = -1
```

- A partir de la fenêtre "Workspace"



une fenêtre contenant les valeurs de la matrice (analogue à une fenêtre Excel) apparait



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

et on peut alors cliquer sur une case pour modifier la valeur correspondante.

Cette dernière fenêtre peut aussi être ouverte par la fonction "**open**". Par exemple, pour éditer la matrice A, on exécute la commande :

```
>> open A
```

Remarque : les même démarches peuvent être utilisées pour modifier unecomposante d'un vecteur.

5. Opérateurs relationnels

Les opérateurs relationnels sont :

==	égal
~=	différent
>	strictement supérieur
>=	supérieur ou égal
<	strictement inférieur

<= inférieur ou égal.

Ces opérateurs permettent d'effectuer des tests relationnels sur les variables. Le résultat est 1 si le test est vrai et 0 s'il est faux. Lorsqu'on applique un opérateur relationnel à une matrice, le test est effectué pour chaque élément. Le résultat est une matrice de même dimension formée par des 1 (test vrai) et des 0 (test faux). Par exemple :

```
>> A = [2 7 6;9 0 5;3 0.5 6];
>> B = [8 7 0;3 2 5;4 1 7];
>> A == B
```

Opérateurs logiques

Les opérateurs logiques sont :

& et

| ou

~ non.

Exécutons les commandes suivantes :

```
>> a = [0; 0; 1; 1], b = [0; 1; 0; 1]
>> c = a&b
>> d = a|b
>> e = (~a)|b
>>d = [a, b, a&b, a|b, (~a)|b]
```

Les variables c, d et e résultants de tests logiques sont de "type logical" (consulter la fenêtre Workspace ou exécuter la commande "`whos`").

```
>> x = 1; y = 3; w = -1; z = 10;
>> (x>y)&(w>z)
>> ~(x>y)&(w>z)
>> (x>y)&(w<z)
>> (x<y)&(w<z)
```

Pour voir les différents opérateurs utilisés par MATLAB, il suffit de taper "help" suivi d'un opérateur (quelqu'il soit) :

```
>> help /
```

6. Eléments répondant à une condition donnée

Pour trouver les éléments d'un vecteur ou d'une matrice répondant à un critère (ou condition) donné, on peut procéder de la manière suivante :

```
>> R=[1 -1 0;-3 0 -2;0 1 -3]
>> Rpos = R > 0
>> Rpositif = R(Rpos)
```

On obtient un vecteur colonne contenant les éléments positifs (R>0) de la matrice R.

Le même résultat peut être obtenu en utilisant la fonction "**find**". Cette fonction retourneun vecteur colonne formé parlesindices des éléments répondant à la condition. Par exemple :

```
>> k = find(R<0)
```

Pour trouver les valeurs des éléments associés à ces indices, il suffit de taper :

```
>> R(k)
```

Remarque : appliquée à une matrice A, `find(A)` retourne les indices des éléments non nuls de la matrice A. Lacommande :

```
>> [i j v] = find(A)
```

retourne les indices des lignes et colonnes (i,j) des éléments non nuls de A et retourne ces éléments dans le vecteur v.

Il y a aussi des fonctions de tests dont la réponse est de type logical :

- Fonction "**isreal**"

retourne 1 si *x* est un nombre réel et 0 si x ne l'est pas. Si *x* est une matrice, la fonction isreal retourne 1 si tous les éléments de la matrice sont réels, 0 sinon.

```
>>x = [2   2+i   -3.2; 1-i   3-2i   4],isreal(x)
```

retourne 1 si *x* n'est formé que de nombres réels et 0 si *x* ne l'est pas.

• Fonction "**isequal**"

Teste l'égalité entre entités.

```
>> A=[1 0; 0 1], B=[1 0; 0 1], C=[1 0; 1 0]

>>isequal(A,B)
```

```
>> isequal(A,B,C)
```

• Fonction "**isprime**"

Appliquée à une matrice A, la fonction "isprime" retourneune matrice de même dimension formée par des 1 à la place des éléments qui sont des nombres premiers et des 0 à la place des éléments qui ne le sont pas :

```
>> x = [2    5   6; 3   11   8]
>> k = isprime(x)
>> x(k)
```

La dernière commande retourne un vecteur colonne formé par les éléments de la matrice répondant à la condition.

D'autres fonctions de ce genre existent dans Matlab (isscalar, isreal, isvector,isempty, …).

7. Fonctions usuelles

Dans Matlab, on appelle fonction toute instruction acceptant une ou plusieurs entrées et retournant une ou plusieurs sorties :

```
>> k = isprime(x)
```

"isprime" est une fonction acceptant une entrée (qui peut être une matrice) et retournant une sortie (qui peut aussi être une matrice).

7.1.1. Fonctions spécifiques aux vecteurs

- Somme des éléments de x : **sum(x)**
- Produit des éléments de x : **prod(x)**
- Le plus grand élément de x : **max(x)**
- Le plus petit élément de x : **min(x)**

- Moyenne des éléments de x : **`mean(x)`**
- Ordonner les éléments de x par ordre croissant : **`sort(x)`**
- Produit scalaire de deux vecteurs v1 et v2 : **`dot(v1,v2)`**
- Produit vectoriel de deux vecteurs v1 et v2 : **`cross(v1,v2)`**
- …

7.2. Fonctions mathématiques usuelles

7.2.1. Fonctions trigonométriques usuelles

Fonction	Description
`cos (x)`	Cosinus de l'angle x (x en radians)
`cosd (x)`	Cosinus de l'angle x (x en degrés)
`sin (x)`	Sinus de l'angle x (x en radians)
`sind (x)`	Sinus de l'angle x (x en degrés)
`tan (x)`	Tangente de l'angle x (x en radians)
`tand (x)`	Tangente de l'angle x (x en degrés)
`cot (x)`	Cotangente de x (x en radians): $\cot(x) = \frac{1}{\tan(x)}$
`cotd (x)`	Cotangente de x (x en degrés)
`cosh(x)`	Cosinus hyperbolique de x : $\cosh(x) = \frac{e^{x} + e^{-x}}{2}$
`sinh(x)`	Sinus hyperbolique de x : $\sinh(x) = \frac{e^{x} - e^{-x}}{2}$
`tanh (x)`	Tangente hyperbolique de x : $\tanh(x) = \frac{\sinh(x)}{\cosh(x)}$
`coth (x)`	Cotangente hyperbolique de x : $\coth(x) = \frac{1}{\tanh(x)}$

L'inverse de chacune de ces fonctions peut être obtenu en metant un "a" devant la fonction (asin : arcsinus, acos : arcosinus, atan : arctangent, …).

Remarques :

- Dans Matlab, lorsqu'une fonction est appliquée à un vecteur (ou une matrice), elle s'applique élément par élément (le résultat est donc un vecteur (matrice) de même dimension) :

```
>> x = [0, pi/6, pi/3, pi/2, 2*pi/3, pi]
>> y = cos(x)
```

- Pour convertir les angles en radians en degrés ou inversement, on utilise les fonctions "**rad2deg**" ou "**deg2rad**". Dans les versions récentes de Matlab, ces deux fonctions fonctions ont été remplacées par "**radtodeg**" et "**degtorad**".

7.2.2. Fonctions exponentielles usuelles

Fonction	Description
exp(X)	Exponentielle de X. Si X est une matrice, la fonction s'applique à chaque élément. Pour un nombre complexe $z = x + i*y$, la fonction "exp" retourne : $\exp(z) = e^{x}(\cos(y) + i * \sin(y))$. Exemple : trouver l'exponentielle de $z=i\pi$.
log(X)	Logarithme naturel ou logarithme népérien ou logarithme de base e. Si X est une matrice, la fonction s'applique à chaque élément. Pour un nombre negatif ou complexe $z = x + i*y$, la fonction "log" retourne : log(z)=log(abs(z))+i*atan2(y,x) Exemple : trouver le log de z=-1.
log10(X)	Logarithme de base 10 de X: $\log 10(X) = \frac{\log(X)}{\log(10)}$
log2(X)	Logarithme de base 2 de X.
pow2(X)	Retourne une matrice de même dimension que X et dont chaque élément est donné par 2^{x} où x est l'élément correspondant de la matrice X. Cette fonction est équivalente à l'opération : 2.^X. Exemple : X=[1,2,3;4,5,6;7,8,9], pow2(X), 2.^X

sqrt(X)	Retoune une matrice de même dimension que X et dont les éléments sont formés par les racines carrées des éléments de la matrice X. Exemple :`X=[1,4,9;16,25,36;49,64,-1],Y=sqrt(X)`

7.2.3. Fonctions complexes usuelles

Fonction	Description
abs(X)	Retourne la valeur absolue des élément de la matrice X si tous les éléments sont réels. Appliquée à un nombre complexe, la fonction `abs` retournele module de ce nombre : **(sqrt(real(X).^2+imag(X).^2)** Exemple : **X=[1+i,-2,-1-i;2-i,3,2+i],abs(X)**
angle(X)	Retourne les arguments (phases) en radians des éléments de X.
complex(a,b)	Crée des nombres complexes à partir des vecteurs a (partie réelle) et b (partie imaginaire). Exemple : **a=[1,2;3,4], b=[5 6;7 8],complex(a,b)**
conj(X)	Retourne le complexe conjugué des éléments de la matrice X. Exemple : **a=[1,2;3,4], b=[5 6;7 8],X=complex(a,b), conj(X)**
real(X)	Retourne les parties réelles des éléments de X.
imag(X)	Retourne les parties imaginaires des éléments de X.

7.2.4. Arrondi et reste

Fonction	Description
ceil(A)	ceil (plafond) arrondiles éléments réels de A aux

	entiers supérieurs les plus proches. Si des éléments sont complexes, les parties réelles et imaginaires sont arrondies indépendamment. Exemple : **a = [-1.9, -0.2, 5.6, 1/3, 2.4+3.6i], ceil(a)**
fix(A)	Arrondi les éléments réels de A aux entiers les plus proches du côté de 0. Si des éléments sont complexes, les parties réelles et imaginaires sont arrondies indépendamment. Exemple : **a = [-1.9, -0.2, 5.6, 1/3, 2.4+3.6i], fix(a)**
floor(A)	Arrondi chaque élément réela de A à l'entier le plus procheinférieur ou égal à a. Si des éléments sont complexes, les parties réelles et imaginaires sont arrondies indépendamment. Exemple : **a = [-1.9, -0.2, 5.6, 1/3, 2.4+3.6i], floor(a)**
round(A)	Arrondi chaque élément réela de A à l'entier le plus proche. Si des éléments sont complexes, les parties réelles et imaginaires sont arrondies indépendemment. Exemple : **a = [-1.9, -0.2, 5.6, 1/3, 2.4+3.6i], round(a)**
rem(X,Y)	Retourne le(s) reste(s) après la division de X par Y. Exemple : **rem([0:5],3)** **rem(magic(3),3)**

Partie II

Scripts et fonctions

Les fichiers qui contiennent un programme dans le langage MATLAB ont l'extension **.m** (on les appelle les M-files). Ces fichiers doivent être enregistrés dans un répertoire dont le chemin est connu par MATLAB.

Avant de passer au mode programmation, il est donc vivementrecommandé(c'estmêmeindispensable)quechaqueutilisateurcréeunrépertoiredetravailMATLAB dans lequel seront enregistrés les programmes qu'il va créer.

Pour connaître le répertoire de travail actuel, tapez la commande **cd** (Current Directory):

```
>> cd
```

Le même résultat peut être obtenu en utilisant la commande **pwd** (Show (print) current working directory):

```
>> pwd
```

On peut créer un répertoire de travail de deux manières :

- classique : ouvrir le disque dur ou amovible dans lequel vous voulez créer votre répertoire, clic droit → Nouveau → Dossier, puis taper le nom choisi.
- Avec MATLAB : on tape la commande **mkdir** (make new directory) suivie du chemin du répertoire qu'on veut créer; exemple:

```
>> mkdir D:\ETI\S4\Matlab
```

Remarque : Dans le chemin du répertoire, il faut éviter d'utiliser des espaces ou des symboles.

Pour ensuite indiquer à MATLAB de travailler dans notre nouveau répertoire, il y a différentes manières :

- Utiliser la commande `cd` :

```
>> cd D:\ETI\S4\Matlab
```

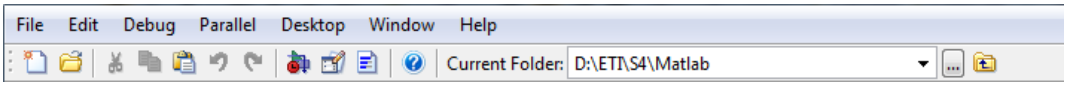
Vous pouvez vérifier que vous êtes bien dans le répertoire choisi en exécutant les commandes cd ou `pwd`.

- Utiliser la commande **`addpath`** (Add directory to search path) suivie du chemin du répertoire désiré. Ceci ajoutera le chemin de notre répertoire aux chemins reconnus par MATLAB :

```
>> addpath D:\ETI\S4\Matlab
```

Ceci peut aussi être réalisé en procédant de la manière suivante : aller dans le menu **`File`** → **`Set Path`**… puis dans la fenêtre qui s'affiche cliquer sur **`Add Folder`**… et choisissez le chemin de votre répertoire.

- En utilisant le Une manière rapide de changer le répertoire courant est d'utiliser le champ "Current folder" dans la barre d'outils du bureau comme ci-dessous :



Remarques : Ci-dessous quelques commandes MATLAB qui agissent sur les répertoires de travail :

– **`dir`**ou **`ls`** : affiche le contenu du répertoire de travail actuel ;

– **`delete`** : efface le fichier spécifié du répertoire de travail.

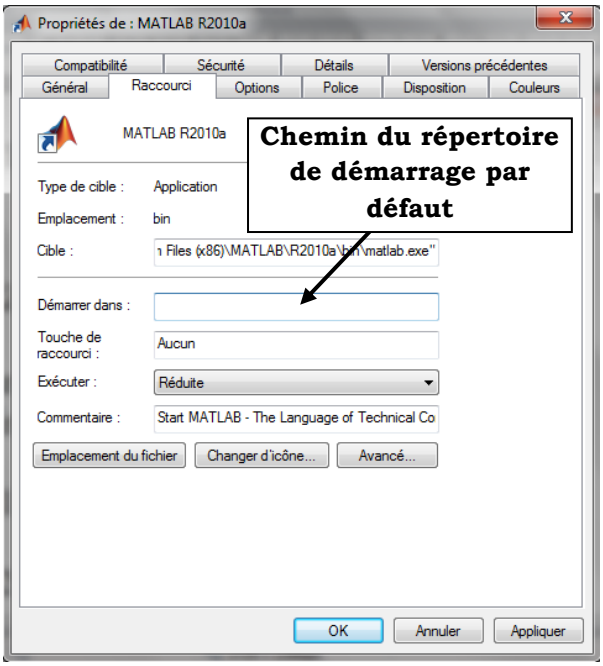
– **`rmpath`** : supprime un chemin d'accès du MATLAB PATH.

– **`what`** : retourne la liste des m-files et des mat-files du répertoire de travail.

Changer le répertoire de démarrage par défaut :

Pour changer le répertoire de démarrage par défaut :

1. Réaliser un clic droit sur l'icône Matlab dans le Bureau de votre PC, ou accéder au raccourci Matlab à partir du menu Démarrer et réaliser un double clic.
2. Choisir Propriétés.
3. Dans la fenêtre qui apparait, entrez le chemin de votre répertoire dans le champ devant "Démarrer dans" :



4. Clic sur OK.

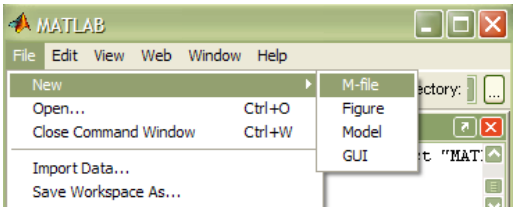
Démarrer ensuite Matlab et vérifier le répertoire de travail actuel à l'aide de la commande `cd` ou `pwd`.

Programmation avec Matlab : M-files

Les programmes Matlab sont des fichiers dont l'extension est "**.m**", c'est pourquoi on les appelle souvent : M-files. On peut créer des M-files en utilisant le propre éditeur de texte de Matlab (le M-files Editor) ou en utilisant un éditeur de texte comme le Bloc-notes.

Le M-files Editor peut être ouvert de trois manières :

- à partir de la barre d'outils en cliquant sur l'icône  , ou
- en tapant la commande **`>> edit`**, ou
- à partir du menu fichier en cliquant sur File→New→ M-file (figure ci-dessous).



Il y a deux types de programmes Matlab : les **scripts** et les **fonctions**.

1. Scripts

Un script n'est rien d'autre qu'un fichier contenant une suite de commandes. Lorsque l'on demande à Matlab d'exécuter un script, il se contente de lancer les différentes commandes du fichier.

Les scripts peuvent manipuler lesdonnées contenues dans l'espace de travail. Bien que les scripts n'aient pas d'arguments de retour, toutes les variables créées pendant leur exécution sont directement accessibles dans le work space, et peuvent donc être utilisées pour d'autres tâches.

Si un script s'appelle **`prog1.m`**, alors son exécution se fera simplement par la commande :

```
>> prog1
```

Bien sûr, le répertoire contenant le programme `prog1.m` doit être le répertoire de travail actuel ou devait avoir été ajouté auparavant au path de Matlab.

Exercice :

Ouvrir l'éditeur de programme (M-files editor) de Matlab. Ecrire, alors les commandes qui permettront de : (écrire une commande par ligne)

1. Définir le vecteur :

   x = (1,1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6, 6.5, 7, 7.5, 8, 8.5, 9, 9.5, 10)
2. Calculer la longueur N de ce vecteur.
3. Calculer la somme S des éléments de x.
4. En déduire la moyenne $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$.
5. Calculer l'écart-type $\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}\left(x_i - \bar{x}\right)^2}$

Enregistrer le dans le répertoire actuel de travail sous le nom "**`firstscript`**".

Pour exécuter ce script, il suffit d'exécuter la commande :

```
>> firstscript
```

Remarque :

- Il est important de commencer un programme par l'instruction **`clear`**. Cette instruction effacera toutes les variables se trouvant dans

l'espacede travail. A la fin de l'exécution, seules les variables créées par notre script seront présentes dans l'espace de travail.

- Il est important de commenter abondamment un programme. Ceci permet de comprendre le programme lorsqu'on a besoin de le réutiliser après une longue période. Dans Matlab, un commentaire commence par « **%** ».

La figure suivante donne une manière d'écrire le script ci-dessus :

```
clear % cette commande va effacer toutes les
      % variables existantes dans l'espace de travail
x=1:0.5:10; % création du vecteur x
N=length(x),% calcul de la longueur du vecteur x
S=sum(x), % calcul de la somme des éléments de x
xbar=S/N, % calcul de la moyenne
sigma=sqrt(sum((x-xbar).^2)/(N-1)), % calcul de l'écart-type
```

2. Fonctions

Une fonction $y = f(x)$ est une procédure mathématique qui reçoit en entrée la variable x et l'utilise pour calculer la sortie y suivant une (ou plusieurs) relation(s) bien définie(s). Une fonction Matlab est un M-file qui calcule des variables de sorties (arguments) en utilisant des arguments d'entrée.

La syntaxe générale est la suivante:

```
function[y1, ...,ym]=mafonction(x1, ...,xn)
% y1, ..., ym sont les variables de sortie de la fonction;
% x1, ..., xn sont les variables d'entrée de la fonction
% Commentaires permettant la description de la fonction. Et c'est
% ces commentaires, se trouvant juste après la déclaration
% function qui vont être affichés quand on tapera la commande
% >> help mafonction.
```

help

Corps de la fonction *contenant les instructions et les commandes*

Le fichier doit impérativement commencer par la déclaration **function**. Suit entre crochets 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, les crochets sont inutiles. Il est impératif qu'une fonction soit enregistrée dans un fichier du même nom.

Exemple :

La programmation de la fonction mathématique $f(x)=x^2$ peut se faire par le M-file suivant :

```
function y = carre(x)
% Création de la fonction f(x) = x au carré
%  x = scalaire ou vecteur
% y = scalaire ou vecteur

y = x.^2;
```

Notez aussi l'utilisation de l'opérateur : **.^** qui permet de calculer la puissance élément par élément.

Pour utiliser la fonction du fichier « carre.m », il suffit d'entrer au clavier:

```
>> carre(2)
```

On peut aussi placer le résultat dans une autre variable:

```
>> b = carre(5)
```

ou encore puisque l'on peut utiliser des vecteurs:

```
>> v = [1 2 3]; r = carre(v)
```

Pour utiliser cette fonction, il n'est pas nécessaire d'utiliser les mêmes noms de variables que ceux utilisés dans le fichier carre lui-même.

On peut aussi utiliser notre fonction dans un script.

```
clear
%  Partition de l'intervalle [0,5]
   x = 0 : 0.01 : 5.0 ;
%  Evaluation de la fonction sur tout le vecteur x
   y = carre(x)
```

Exécuter une fonction : multiples outputs

```
function[out1, …, outn] = nomfonc(in1, …, inm)
```

- Quand on exécute la commande :

```
>>[out1, …, outk] = nomfonc(inp1,…,inpn)
```

où k<n, la fonction retourne les k premiers outputs.

- Quand on exécute la commande :

```
>> nomfonc(inp1,…,inpn)
```

ou

```
>> y = nomfonc(inp1,…,inpn)
```

la fonction retourne uniquement lepremier output

NB :

- L'ordre d'appel des arguments input et output est important !
- Ranger les variables de sortie (outputs) par ordre d'importance.

Les sous fonctions

Un fichier M peut contenir le code de plusieurs fonctions : la première fonction du fichier sera la **fonction principale**i.e. la fonction appelée par le fichier M et les fonctions suivantes sont les **sous fonctions**appelées par la fonction principale.

L'ordre d'apparition des sous fonctions est indifférent du moment que la fonction principale apparaît en premier.

Exemple :

```
function [avg, med] = newstats(u) %fonction principale
%NEWSTATS calcule la moyenne et la médiane en utilisant des
%sous fonctions
n = length(u);
avg = moyenne(u,n); %appel de la sous fonction mean
med = mediane(u,n); %appel de la sous fonction median

function a = moyenne(v,n) %sous fonction
%calcule la moyenne du vecteur v de longueur n
a = sum(v)/n;

function m = mediane(v,n) %sous fonction
%calcule la médiane du vecteur v de longueur n
w = sort(v);
if rem(n,2) == 1
m = w((n+1)/2);
else
m = (w(n/2)+w(n/2+1))/2;
end
```

Variable locale et variable globale

- **Variables locales dans une fonction :**
  - Variables définies dans le corps de la fonction
  - Ne restent pas en mémoire après l'exécution de la fonction
  - propres à chaque fonction qui les utilise.

Exemple :

```
function [xbar S]=varlocale(x)
% Dans cette fonction, N est une variable locale qui
% n'apparaitrait pas dans l'espace de travail après
% l'exécution de la fonction
N=length(x);
S=sum(x);
xbar=S/N;
```

```
>> v=1:10; [xbar S]= varlocale(x); whos
```

La variable N n'apparait pas dans l'espace de travail.

- **Variable globale**

Une variable globale peut être partagée par plusieurs fonctions etéventuellement par l'espace de travail de base.

Toute affectation d'une valeur à une variable globale est accessible à toutesles fonctions et l'espace de travail de base qui l'auront déclarée commevariable globale

- **Déclaration de variables globales**

```
function[vars1, …, varsm] = nomfonc(vare1, …, varen)
global VAR1 VAR2 VAR3 … VARN
….
```

☞ VAR1, VAR2, …, VARN sont des variables globales par convention, déclarées en début du fichier après la ligne de définition de la fonction (elles doivent être séparées par des espaces et non par des virgules ou points-virgules).

☞ Les noms des variables globales doivent être en majuscules.

☞ Si des sous fonctions utilisent des variables globales, elles doivent être déclarées comme globales au début de chaque sous fonction.

Exemple 1 :

```
function [y x]=sinus1(a,b) %fichier sinus1.m
global H
H = 100;
x = a:(b-a)/H:b;
```

```
function [y,x]=cosinus1(a,b) %fichier cosinus1.m
global H %la valeur affectée à la variable globale H
dans la
x = a:(b-a)/H:b; %fonction sinus1 est accessible
y = cos(x);
```

Appel des fonctions :

```
>> y1=sinus1(0,2*pi);

>> y=cosinus1(0,2*pi);
```

Remarque :

– Après un premier appel de la fonction sinus1, la valeur 100 est affectée à la variable globale H : elle est alors accessible à la fonction cosinus1

– Si la variable H n'est pas déclarée comme globale dans la fenêtre de commande alors elle n'est pasaccessible dans l'espace de travail de base mais seulement dans les espaces de travail des deux fonctions. Ainsi, si on exécute la commande suivante :

```
>>global H;y1=sinus1(0,2*pi);y=cosinus1(0,2*pi);
```

la valeur de H sera maintenant enregistrée dans l'espace de travail.

Afficher les variables globales :

```
>> who global

>> whos global
```

Supprimer des variables globales :

```
>> clear global% supprime toutes les variables globales

>> clear global VAR1, VAR2, …, VARN% supprime les variables
                                    % globales VAR1, VAR2, …, VARN
```

Exercices :

Exercice 1 :

Ecrire un script Matlab qui nous permettra de calculer les caractéristiques suivantes des matrices carrées :

Fonction	Description
det(A)	Déterminent de la matrice carrée A
rank(A)	Le rang d'une matrice A est le nombre maximal de vecteurs lignes ou colonnes linéairement indépendants.
trace(A)	La trace d'une matrice A = la somme des éléménts de sa diagonale. Ce résultat peut aussi être obtenu de la façon suivante : **sum(diag(A))**
[V,D] = eig(A)	Retourne la matrice V dont les colonnes sont les vecteurs propres et la matrice diagonale D dont les éléments sont les valeurs propres de A. Les valeurs propres et vecteurs propres d'une matrice A sont des entités vérifiant la relation : $(A-\lambda_i I)V_i = 0$ I étant la matrice identité. On dit que V_i est un vecteur propre associé à la valeur propre λ_i.
tril(A)	Partie triangulaire inférieure de A
triu(A)	Partie triangulaire supérieure de A

Exercice 2 :

1) Ecrire une fonction nommée "stats" s'appliquant à un vecteur ou à une matrice est qui retournerait les statistiques suivantes :
   - Somme (fonction Matlab**"sum"**)
   - Moyenne (fonction Matlab**"mean"**)
   - Médiane (fonction Matlab**"median"**)
   - Variance (fonction Matlab**"var"**)
   - Ecart-type (fonction Matlab**"std"**)
2) Ecrire un script dans lequel il sera demandé d'entrer un vecteur ou une matrice (commande **"input"**) et qui utilisera la fonction `stats` pour calculer les statistiques correspondantes.

Exercice 3 :

L'écriture sous forme de produit matriciel d'un système d'équations linéaire respecte un certain ordre dans la multiplication. Ainsi le système :

$$\begin{cases} a_{11}x + a_{12}y + a_{13}z = b_1 \\ a_{21}x + a_{22}y + a_{23}z = b_2 \\ a_{31}x + a_{32}y + a_{33}z = b_3 \end{cases}$$

peut être écrit sous une forme matricielle: `AX=B`, avec:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \qquad X = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \qquad B = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Résoudre ce système d'équations, c'est trouver X tel que X=inv(A)*B. Cette solution existe si la matrice A est inversible.

1) Ecrire une fonction Matlab qui, connaissant A et B, elle nous retourne la solution de notre système.
2) Appliquer cette fonction pour résoudre les systèmes suivants :

a) $\begin{cases} 3x + 2y - z = 10 \\ -x + 3y + 2z = 5 \\ x - y - z = -1 \end{cases}$

b) $\begin{cases} 3x - 2y = 5 \\ 6x - 2y = 2 \end{cases}$

c) $\begin{cases} x + 4y - z + w = 2 \\ 2x + 7y + z - 2w = 16 \\ x + 4y - z + 2w = 1 \\ 3x - 10y - 2z + 5w = -15 \end{cases}$

Partie III

Graphiques dans Matlab

MATLAB fournit une grande variété de techniques pour l'affichage des données sous forme graphique.

Des outils interactifs vous permettent de manipuler les graphiques révéler la plupart des informations contenues dans vos données.

Le processus de visualisation des données implique une série d'opérations. Cette section fournit une vue générale sur les possibilités que fournit Matlab pour la représentation graphiques.

I. Graphiques 2D

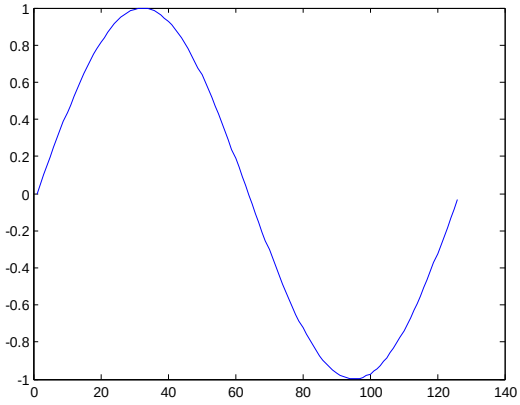
I.1. Fonction `plot`

C'est la fonction la plus utilisée dans la représentation graphique 2D.

Ecrire le script suivant :

```
x=0:0.05:(2*pi);
y= sin(x);
plot(y);
```

On obtient la figure suivante :

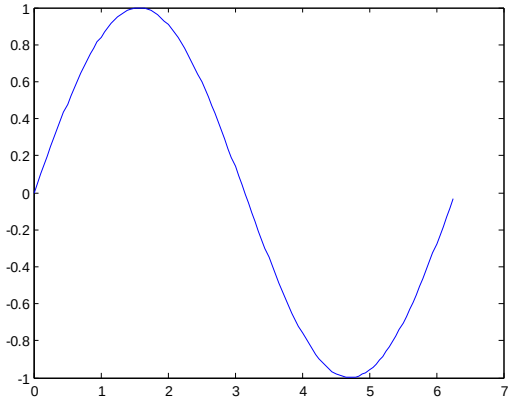


On peut remarquer que la commande plot(y) trace le vecteur y pour les abscisses 1 à 126 qui sont les indices du vecteur y.

En remplaçant `plot(y)`, dans le script précédent, par **`plot(x,y)`**:

```
x=0:0.05:(2*pi);
y= sin(x);
plot(x,y);
```

on obtient la courbe suivante :



On peut remarquer que la commande `plot(x,y)` permet de tracer le vecteur `y` en fonction `x`.

Remarques :

- ☞ Lorsqu'on utilise la commande `plot` pour la première fois, cette commande crée automatiquement une fenêtre graphique dans laquelle la courbe sera tracée. L'utilisation de la commande `plot` une 2ème fois utilisera cette même fenêtre et va effacer l'ancienne courbe pour tracer la récente.

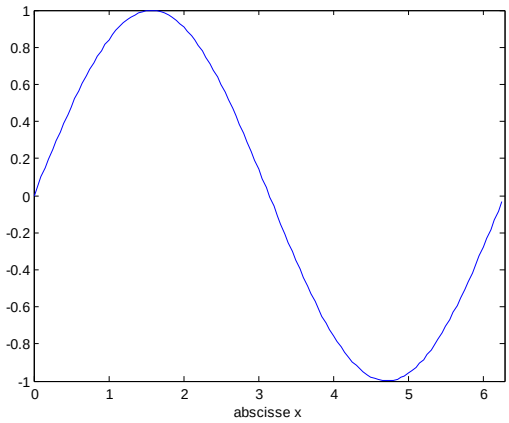
A la courbe ci-dessus, on peut ajouter les titres des axes et de la courbe en ajoutant, au script ci-dessus ou en exécutant dans la fenêtre de commande, les commandes suivantes:

```
xlabel('x')
ylabel('sin(x)')
title('sin(x) entre 0 et 2pi')
```

On peut aussi régler les échelles des axes des abscisses et des ordonnées des courbes 2D en utilisant la commande **`axis([`x_{min} x_{max} y_{min} y_{max}`])`**. Pour notre courbe, on peut ajouter la commande :

```
axis([0 2*pi -1 1])
```

On obtient alors la figure suivante où on remarque que la courbe se marie maintenant au cadre de la figure :



N.B :

`axis square`= axes x et y de même longueur

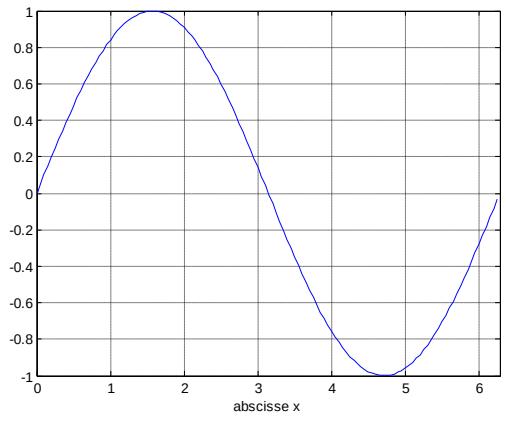
`axisequal`= pas de même longueur sur les axes x et y

`axis off`= rend les axes invisibles

`axis on`= rend les axes visibles (par défaut)

Finalement, on peut ajouter aussi un quadrillage à la courbe :

```
grid on
```



On peut utiliser la commande **`grid on`** ou tout simplement **`grid`**. Pour enlever le quadrillage, on utilise la commande :

```
grid off
```

Finalement, on peut changer l'épaisseur du trait de représentation de la figure en utilisant la commande plot suivante :

```
plot(x,y,'LineWidth',n)
```

Par défaut n = 1.

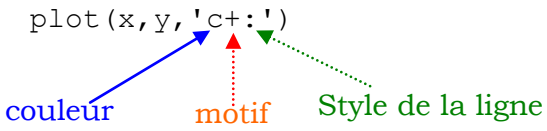
On peut aussi changer la couleur du fond de la courbe à l'aide de la commande :

```
whitebg('couleur')
```

'couleur' peut être : 'reed', 'blue', 'cyan', 'green', ….

Remarque :

Par défaut,les courbes sont tracées par une ligne continue de couleur bleue. Pour utiliser des couleurs, des motifs et des stylesde lignes différente, on peut utiliser la syntaxe générale de la fonction `plot` :



Le tableau suivant donne les différents styles, motifs et couleurs supportés par la commande `plot` :

Couleur		Marqueurs		Styles de lignes	
y	jaune	.	point	-	Lignesolide
m	magenta	o	cercle	:	pointillée
c	cyan	x	x	-.	Trait d'union-point
r	rouge	+	plus	- -	coupée
g	vert	*	étoile		
b	bleu	s	carré		
w	blanc	d	diamant		
k	noire	v	Triangle (bas)		
		^	Triangle (haut)		
		>	Triangle (droite)		

Exemple : Exécuter les commandes suivantes :

```
x=0:0.25:(2*pi);
y= sin(x);
plot(x,y,'r+:);
```

I.1.1 Texte dans la figure

La commande **text**permet d'écrire écrit un texte à une position précise sur la figure : **text(posx,posy,'un texte')** où posxet posysont les coordonnées du point de début de texte. Dans la courbe précédente, on peut insérer un texte à l'aide de la commande suivante :

```
text(4,0.8,'Courbe y = sin(x)')
```

La commande **gtext**('texte ici')permet d'écrire le texte 'texte ici' à une position choisie à l'aide de lasouris :

```
gtext('Courbe y = sin(x)')
```

I.1.2. Représentation de plusieurs courbes dans une même fenêtre graphique

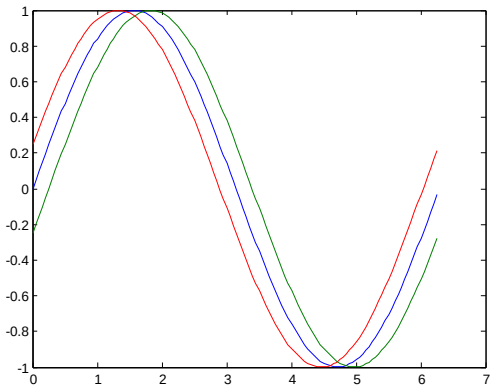
Dans certains cas, on est amené à tracer plusieurs courbes dans une même fenêtre graphique. Dans MATLAB, ceci peut se faire des deux manières suivantes :

a) Utilisation d'une seule commande`plot` :

Ecrire le script suivant :

```
x=0:0.05:(2*pi);
y1= sin(x);
y2=sin(x-0.25);
y3=sin(x+0.25);
plot(x,y1,x,y2,x,y3)
```

on obtient la courbe suivante :



On remarque que la dernière commande plot associe une couleur différente à chaque courbe. Un même résultat est obtenu en utilisant la commande :

```
plot(x,[y1;y2;y3])%Noter le point-virguleentre les yi
```

A la courbe obtenue ci-dessus, on peut ajouter une légende pour distinguer entre les différentes courbes. Ceci peut se faire en ajoutant la commande suivante :

```
legend('sin(t)','sin(t-0.25)','sin(t+0.25')
```

On peut aussi tracer les 3 courbes précédentes en utilisant des motifs, des couleurs et des styles de traits différents. Pour cela, on peut utiliser la commande suivante :

```
x=0:0.05:(2*pi);
y1= sin(x);
y2=sin(x-0.25);
y3=sin(x+0.25);
plot(x,y1,'k+:',x,y2,' g*-',x,y3,'md--')
legend('sin(t)','sin(t-0.25)','sin(t+0.25')
axis([0 2*pi -1 1])
```

b) Utilisation de la commande : `hold on`

On obtient le même résultat que ci-dessus, en exécutant le script suivant :

```
x=0:0.05:(2*pi);
y1= sin(x);
y2=sin(x-0.25);
y3=sin(x+0.25);
plot(x,y1,'k+:')
hold on
plot(x,y2,' g*-')
plot(x,y3,'md--')
legend('sin(t)','sin(t-0.25)','sin(t+0.25')
axis([0 2*pi -1 1])
hold off
```

La commande **`hold on`**permet grader l'ancienne courbe pour tracer dessus de nouvelles courbes. L'effet de cette commande est annulé à l'aide de la commande **`hold off`**.

I.1.3. Représentation surdes fenêtres graphiques différentes

Pour garder la première figure et tracer dans une nouvelle fenêtre graphique, il suffit d'exécuter la commande **`"figure"`**. Exécuter le script suivant :

```
x = 0:0.05:(2*pi);
y1 = exp(-x).*sin(x);
y2 = exp(-2*x).*sin(x);
y3 = exp(-3*x).*sin(x);
y4 = exp(-4*x).*sin(x);
plot(x,y1)
figure% ici équivalente à la commande : figure(2)
```

```
plot(x,y2)
figure% ici équivalente à la commande : figure(3)
plot(x,y3)
figure% ici équivalente à la commande : figure(4)
plot(x,y4);
```

Dans ce script, la première commande `plot` (`plot(x,y)`) ouvre la première fenêtre graphique (`Figure1`). La commande"`figure`" sans argument d'entrée qui suit, permet d'ouvrir une fenêtre graphique nommée**`Figure2`**. La commande "`figure`" est ici équivalente à exécuter la commande : `figure(2)`.

Remarques :

- En utilisant la commande "`figure`" sans argument d'entrée, Matlabouvre une nouvelle fenêtre graphique dont le numéro est = numéro de la dernière fenêtre graphiqueouverte + 1. Si aucune fenêtre graphique n'est ouverte, la commande "`figure`" sans argument d'entrée, ouvre la première fenêtre "`Figure1`".
- Si l'on veut placer une courbe dans une fenêtre qui ne respecte pas l'ordre croissant de numérotation des fenêtres graphiques,on peut utiliser la commande "`figure`"en lui affectant le numéro voulu comme argument d'entrée. Par exemple, la commande "`figure(3)`" ouvre la fenêtre "`Figure3`"même si aucune fenêtre graphique n'est ouverte.

Les autres commandes `figure` et `plot`,dans le script ci-dessus, suivent les mêmes principes sus-décrits.

N.B :

- Lorsqu'on est sur une figure, toutes les commandes graphiques (plot, axis, legend, title, ...) s'appliquent sur cette figure. Pour faire des modifications dans une autre figure (Figure1 par exemple), on doit d'abord l'activer. L'activation d'une fenêtre graphique peut se faire de deux manières :
  - Cliquer sur la fenêtre graphique.
  - Exécuter la commande **`figure(n).`**

- La fermeture des fenêtres graphiques peut se faire des façons suivantes :
  - Fermeture de la fenêtre active : **`>> close`**
  - Fermeture de la fenêtre active de numéro n : **`>> close(n)`**
  - Fermeture de toutes les fenêtres actives : **`>> close all`**

I.1.4 Subdivision d'une fenêtre graphique : `subplot`

La commande "**`subplot`**" nous permet de subdiviser une fenêtre graphique en plusieurs fenêtres pouvant chacune recevoir des représentations graphiques.

Par exemple, on peut représenter les 4 courbes vues ci-dessus dans une seule fenêtre graphique en utilisant les commandes suivantes :

```
subplot(2,2,1), plot(x,y1)
subplot(2,2,2), plot(x,y2)
subplot(2,2,3), plot(x,y3)
subplot(2,2,4), plot(x,y4)
```

Remarquons que les échelles des axes des ordonnées sont différentes pour les 4 courbes. Ceci ne facilite pas l'interprétation physique et il souvent préférable de représenter les courbes sur les mêmes échelles.

Exercice *: Tracer sur une même échelle les courbes étudiées ci-dessus.*

Remarque :

La fonction **`subplot(m,n,p)`** ou **`subplot(mnp)`** subdivise la fenêtre graphique en **`mxn`** (m lignes et n colonnes) cadres séparés. Le $p^{ième}$ cadre est choisi de telle sorte que p=1 correspond à (m=1,n=1), p=2 →(m=1,n=2), ….

subplot(2,3,1)	subplot(2,3,2)	subplot(2,3,3)
subplot(2,3,4)	subplot(2,3,5)	subplot(2,3,6)

I.1.5. Autres fonctions analogues à plot

I.1.5.1. Fonction : `fplot`

La fonction **`fplot('fonct',[xminxmax])`** trace la fonction définie par "fonct" entre x_{min} et x_{max}. On peut aussi délimiter l'axe des ordonnées en précisant le y_{min} et le y_{max} :**`fplot('fonct',[xminxmaxyminymax])`**.

Exemples : Exécuter les commandes suivantes :

```
>>fplot('sin(x)', 2*pi*[-1 1])
>>fplot('[sin(x),cos(x)]',2*pi*[-1 1 -1/6 1/6])
>>fplot('[sin(x),cos(x),x.*sin(x)]',2*pi*[-1 1 -1 1/2])
```

I.1.5.2. Fonction :`ezplot`

`ezplot('func(x)')` trace la function`func(x)` dans le domaine $-2\pi<x<2\pi$.

ezplot('func(x)',[A B]) trace la function`func(x)` dans le domaine A<x<B.

ezplot('func(x)',[A B],FIG) trace la function`func(x)` dans le domaine A<x<B et la fenêtre graphique spécifiée par FIG.

Exemple : Exécuter la commande :

```
>>ezplot('sin(x)',[0 2*pi],figure(2))
```

I.2 Autres fonctions de représentation 2D

Parmi les fonctions de représentation 2D, on peut citer :

- **Fonction bar :bar(x,y)** traceles amplitudes y correspondant aux abscisses x à l'aide de barres verticales :

```
>> x=0:0.25:2*pi; y=sin(x); bar(x,y)
```

- **Fonction polar :**elle permet de représenter des fonctions en coordonnées polaires :

```
>> theta = 0:.1:2*pi;
>> r = 1 + cos(theta);
>> polar(theta,r)
```

- **Fonction stem : stem(x,y)** permet de représenter les ordonnées y comme des barres terminées par un cercle :

```
>>x=0:0.25:2*pi; y=sin(x); stem(x,y)
```

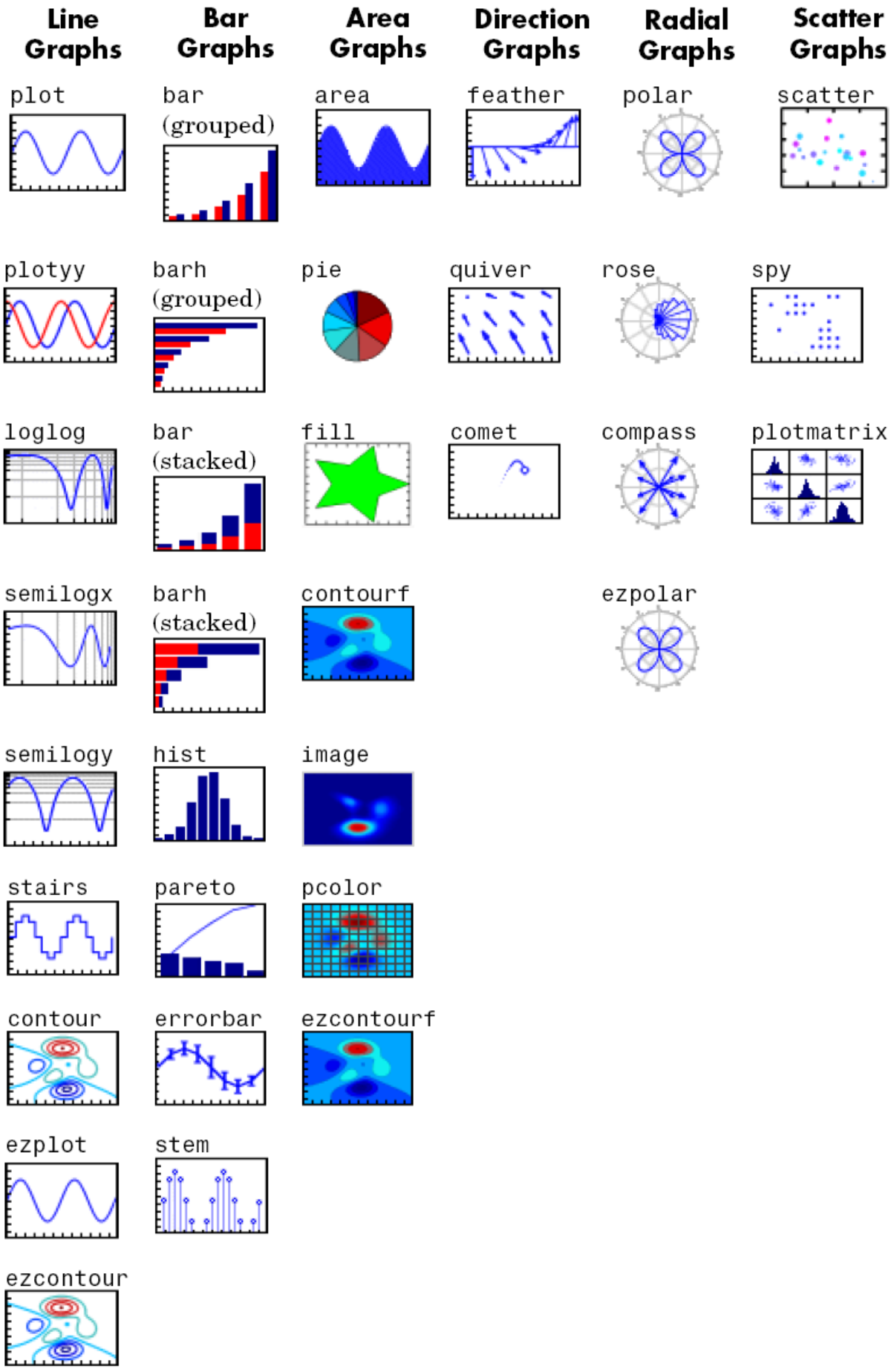
- **Fonction errorbar :errorbar(x,y,e)** trace `y` en fonction de `x` avec les segments d'incertitudes `[y+e  y-e]` :

```
>> x = 1:.5:10;
>>y = sin(x);
>>e = 0.25*std(y)*ones(size(x));
>>errorbar(x,y,e)
```

- **Fonction hist : hist(x,N)** trace un histogramme de N barres de la variable x. Par défaut, N=10.

```
>> y=randn(50000,1);
>>hist(y)
>>hist(y,20)
```

Les fonctions graphiques 2D sont données sur la figure ci-dessous :


Line Graphs
Bar Graphs
Area Graphs
Direction Graphs
Radial Graphs
Scatter Graphs
plot
bar (grouped)
area
feather
polar
scatter
plotyy
barh (grouped)
pie
quiver
rose
spy
loglog
bar (stacked)
fill
comet
compass
plotmatrix
semilogx
barh (stacked)
contourf
ezpolar
semilogy
hist
image
stairs
pareto
pcolor
contour
errorbar
ezcontourf
ezplot
stem
ezcontour

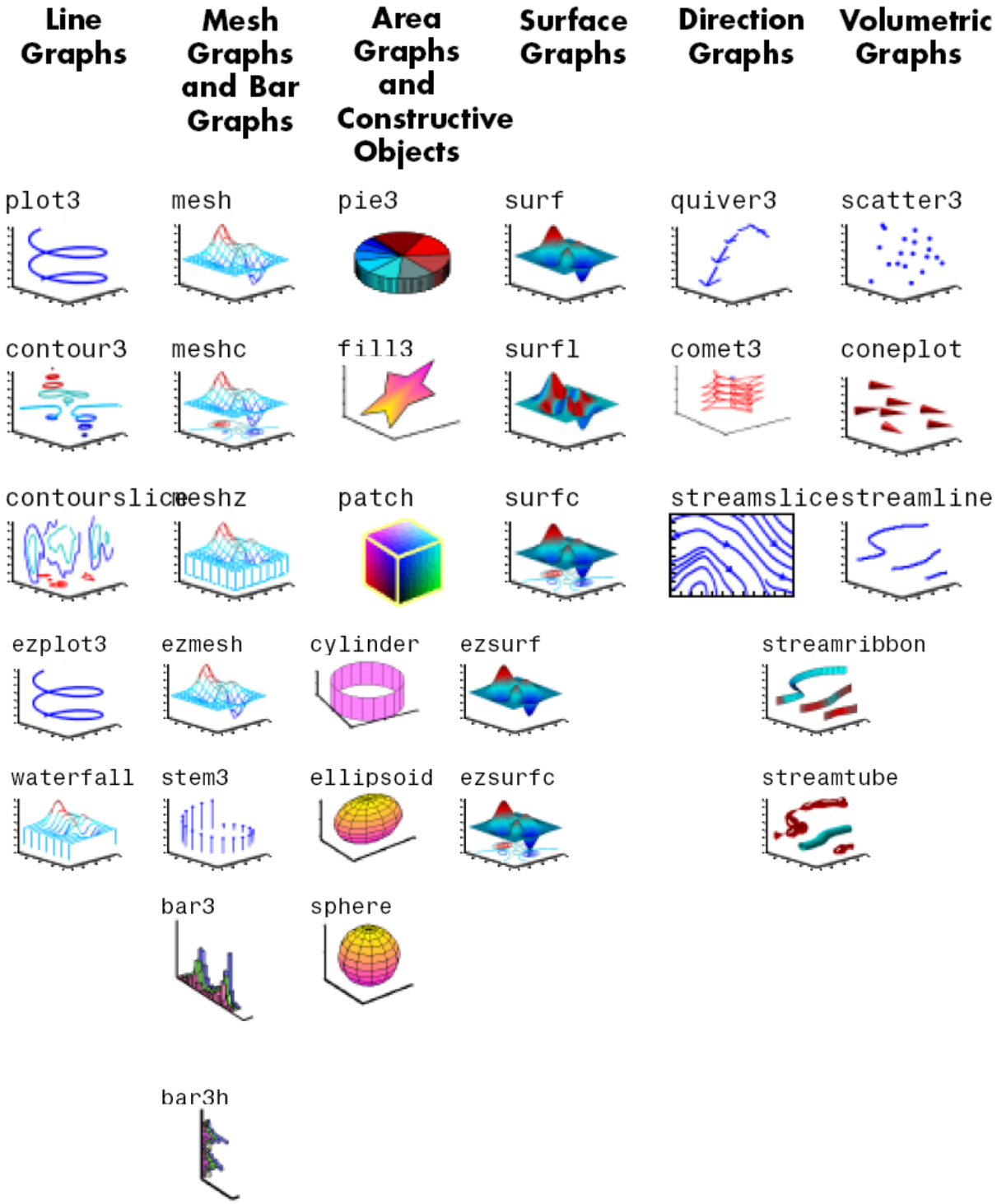
Utiliser le help de la fonction écrite sur l'image pour connaître son effet et la façon dont elle est utilisées.

Pour obtenir la liste des fonctions de représentation graphique 2D disponibles dans MATLAB, tapez :

```
>>help graph2d
```

I.2 Graphes 3D

La figure ci-dessous illustre les fonctions graphiques 3D de MATLAB.



Dans les paragraphes qui suivent nous contemplerons quelques une des fonctions de la figure ci-dessus.

a) **plot3**

Exécuter les instructions suivantes :

```
>> t = linspace(0, 10*pi);
>>plot3(sin(t), cos(t), t)
>>xlabel('sin(t)'), ylabel('cos(t)'), zlabel('t')
>>grid on
```

b) **mesh**("mesh" (maillage)

Pour tracer une fonction *g(x,y)* pour $x \in [xmin, xmax]$ et $y \in [ymin, ymax]$, on procède de la manière suivante :

1) Définition des vecteurs x et y,

```
>> x=-2:.2:2 ;  y= -2:.2:2 ;
```

2) Création d'un maillage du domaine [xmin, xmax]× [ymin, ymax] par la commande **meshgrid** :

```
>> [X,Y] = meshgrid(x, y);
```

3) Evaluation de la fonction aux nœuds de ce maillage :

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

4) Affichage de la surface grâce à la commande **mesh** :

```
>>mesh(X, Y, Z)
```

La fonction **mesh** affiche les surfaces à l'aide de lignes en couleur.

La commande **meshc** permet d'afficher les lignes de niveau sous la surface dans le plan $Z=Z_{min}$:

```
>>meshc(X,Y,Z)
```

La commande **contour**permet de tracer les lignes de niveau de la fonction :

```
>>contour(X,Y,Z)
```

On peut aussi afficher les labels des lignes de niveau :

```
>>[C,h] = contour(X,Y,Z);
>>clabel(C,h);
```

On peut imposer le nombre de lignes de niveau N en utilisant la commande **contour(X,Y,Z,N)** :

```
>>contour(X,Y,Z,20)
```

c) **surfl** (Surface avec illumination)

```
>>clf
>>surfl(X,Y,Z)% graphique avec illumination
>>shadinginterp% meilleure interpolation
>>colormappink%choix d'une palette de couleurs prédéfinie
>>view(-37.5+60,30)%changement de point de vision:view(azimut,élévation)
```

N.B :

- view(0,90); % Met le graphe 3D "à plat", par view(-37.5,30).
- Colorbar% Ajoutel'échelle des couleurs
- axis equal;% Échellesisométriques
- axis tight;% Ajuste les limites des axes aux valeurs des données
- axis([xminxmaxyminymaxzminzmax])% Impose les limites du tracé en x,y, et z
- caxis([-0.1 0.1]); colorbar;% Définit les plages de la colormap et réactualise la colormap
- caxis auto; colorbar; % Retour aux valeurs par défaut

Autres exemples :

Utiliser les fonctions Matlab : plot3, meshc et surfc pour tracer les fonctions suivantes:

1) $z = \frac{xy(x^2-y^2)}{x^2+y^2}, \quad -4 \le x \le 4, \quad -3 \le y \le 3$
2) figure(1) , contour3(x,y,z), figure(2),mesh(x,y,z), figure(3), surfc(x,y,z), figure(4), surfl(x,y,z), shadinginterp, colormap hot :

$$z = \frac{-10}{3+x^2+y^2}, \quad -3 \le x \le 3, \quad -3 \le y \le 3$$

3)

```
vx=-4:0.2:4;
vy=-3:0.2:3;
[x,y]=meshgrid(vx,vy);
z=x.*y.*(x.^2-y.^2)./(x.^2+y.^2+eps);
```

```
plot3(x,y,z),grid on

figure

meshc(x,y,z)

view(-37,15)

figure

surfc(x,y,z)

view(-47,25)

[X,Y] = meshgrid(-8:.5:8);

R = sqrt(X.^2 + Y.^2) + eps;

Z = sin(R)./R;

mesh(X,Y,Z,'EdgeColor','black')
```

Il est possible de créer une surface transparente en utilisant la commande hidden off. Voici le même exemple avec la commande surf.

```
surf(X,Y,Z)

colormaphsv

colorbar
```

4)

```
t=-7:.001:7;plot3(exp(-t/10).*sin(t), exp(-t/10).*cos(t), exp(-t)),grid
```

Partie IV

Boucles & Contrôle de flux

IV.1. Boucles de contrôle

IV.1.1 BOUCLEFOR

On peut créer une boucle en utilisant **for ... end**.

On peut aussi réaliser des boucles FOR imbriquées.

Exemples :

- Boucle FOR simple : Tracer plusieurs courbes cosinus dans la même fenêtre graphique

```
x = -2*pi:pi/50:2*pi;
hold on;
for i = 0:pi/8:2*pi
plot(x,cos(x+i))
end
hold off;
```

- **Deuxboucles FOR :**

```
for i=1:5
for j=1:20
amp=i*1.2;
wt=j*0.05;
v(i,j)=amp*sin(wt);
end
end
```

Remarque :

Matlab est un logiciel dédié au calcul matriciel. Les boucles for sont consommatrice du temps d'exécution et il faut les éviter autant que possible. L'exemplesuivantillustrececi :

```
>>tic,for i=1:500;for j=1:2000; amp=i*1.2; wt=j*0.05; v(i,j)=amp*sin(wt); end; end, toc
>>tic,i=1:500;amp=i*1.2;j=1:2000;wt=0.05*j;v2=amp'*sin(wt);toc
```

IV.1.2 BOUCLE WHILE

On peut créer une boucle en utilisant **while ... end**.

Syntaxe :

```
whilecondition
```

```
opérations à exécuter si la condition est vraie
end
```

a) Exemples :

```
n=1;
while n<7
n=n+1
end
```

```
lent=0;                                %pas de texte saisi
while lent==0                %tant que le texte n'est pas saisi
  texte=input('Saisir un texte:','s'); %demander de saisir un texte
lent=length(texte); %évaluer la longueur du texte
end
```

IV.1.3 Contrôle conditionnel : if, else, elseif, switch

*a) **if, else et elseif***

L'instruction **IF** ... **ELSEIF** ... **ELSE** permet de choisir plusieurs options.

if*condition*

opérations

elseif*condition*

opérations

else

opérations

end

La déclaration **`if`** évalue une conditionlogique et exécute un groupe d'opérations lorsque la condition est vraie. La commande **`elseif`**donne une autre condition à vérifier lorsque la condition de **`if`**n'est pas remplie. La commande **`else`** donne les opérations à exécuter lorsque les conditions précédentes (de **`if`** et de **`elseif`**(lorsqu'elle existe)) ne sont pas vérifiées. Un **`end`** doit terminer le groupe de commandes (un**`if`** doit obligatoirement être terminé par un **`end`**).

Exemple1 :

```
n=input('Donner un nombre positif  ');
if rem(n,3)==0
disp('Ce nombre est divisible par 3')
elseifrem(n,5)==0
disp('Ce nombre est divisible par 5')
else
disp('Ce nombre n''est pas divisible par 3 ou par 5')
end
```

Exemple2 : expression logique scalaire

```
a = 4;
b = 2;
if a>b %si a est strictement plus grand que b alors
diff = a-b; %calculer la difference a-b
disp('a plus grand que b'); %afficher un message
disp(['la difference est a-b = ',num2str(diff)]);
end%fin de la selection
disp('fin de la selection')
```

Exemple3 :

```
A = [1 6; 9 32];
B = [2 3; 3 4];
if B %si tous les éléments de B sont non nuls alors
A./B %faire la division (à droite) de A par B
end%fin de la sélection
disp('fin de la sélection')
```

Exercice :

Trouver le plus grand de deux nombres entiers positifs.

Remarque :

`if B`% équivaut à`if all(B(:))`: si tous les éléments de B sont non nuls.

Exemple 1:

```
a = input('Saisir le numérateur :');
b = input('Saisir le dénominateur :');
if b
d = a/b; %Si b est non nul alors faire la division de a par b
```

```
disp(['résultat de la division = ', num2str(d)]); %Afficher le
                                                  % resultat
else
disp('division par zero !'); %Si b est nul, afficher un
message                          % d'erreur : division par zero
end
```

Exemple2 : Comparaison de deux mots

```
A = input('Saisir le premier mot : ','s'); %par exemple 'matlab'
B = input('Saisir le deuxieme mot : ','s'); %par exemple 'MATLAB';
ifstrcmp(A,B)
disp('meme mot')
elseifstrcmpi(A,B)% Compare A et B en ignorant la casse
disp('meme mot avec des majuscules')
else
disp('mots differents')
end
```

b) switch et case

Un test conditionnel peut aussi être réalisé à l'aide de Les commandes **switch** et **case** permettent de vérifier un certain nombre de conditions suivant la valeur d'une variable donnée.

synthaxe : *switch variable,*

case valeur1, instruction1;

...

casevaleurn, instructionn;

end

Exemple 1 :

```
n=input('Donner un nombre positif  ');
switch n
 case 1,  disp('Method is linear');
case 2,  disp('Method is quadratic');
case 3,  disp('Method is cubic');
otherwise, disp('Unknown method.')
end
```

Exemple2 :comparer un mot

```
mot=input('Entrer l''un des mots: if,else,ouelseif : ','s');
switch mot
    case 'if', disp('instruction selective')
    case 'else', disp('instruction alternative')
    case 'elseif', disp('selection en cascade')
end
```

Exemple3 :séquenced'instructions communes

```
var = 2;
switchvar
case 1, disp('cas 1')
case {2, 3, 4},disp('cas 2, 3 ou 4')
case 5,disp('cas 5')
otherwise
disp('autrescas')
end
```

IV.2BREAK etCONTINUE

Les déclarations "**break**" et "**continue**" permettent un contrôle plus stricte de de la boucle. La déclaration «**break**» provoque la sortie prématurée d'une boucle. L'exécution continue à la ligne se trouvant juste après la fin de la boucle contenant **break**. Dans le cas de boucles imbriquées, **break** interrompt la boucle qui la contient. La syntaxe est simplement d'écrire le mot **break** dans la boucle là où vous souhaitez la briser.

Contrairement à **break**, la déclaration **continue**donne le contrôle à l'itération suivante dans une boucle **FOR** ou **WHILE**. La déclaration **continue**impose au programme de continuer la boucle sans exécuter le code se trouvant après elle. Dans le cas des boucles imbriquées, **continue** donne le contrôle à l'itération suivante de la boucle qui la contient.

Exemples :

- **Continue :**

```
for k=-10:10;if (k^2-50<0)
continue
end
val=k^2-50;
fprintf('\n k=%g val=%g',k,val)
end
```

- **Break**

```
for p=7:8
for q=3:5
for r=1:2
fprintf('%3.0f, %3.0f, %3.0f\n',p,q,r)
end
if q==4, break; end
end
end
```

IV.5.Autres déclarations

return

- provoque un retour au programme appelant (ou au clavier)
- ignore toutes les instructions qui suivent l'instruction **return**

error('message')

- provoque un retour au programme appelant (ou au clavier) et affiche un message d'erreur.

warning('message de mise en garde')

- permet d'afficher un message de mise en garde sans suspendre l'exécution du programme
  - warning off : ne pas afficher les messages de mise en garde
  - warning on : rétablir l'affichage des messages de mise en garde

pause

- permet de suspendre l'exécution du programme
- reprend l'exécution du programme dès que l'utilisateur enfonce une touche du clavier

pause(n)

permet de suspendre l'exécution de programme pendant **n**secondes

Partie V

Importation et exportation des données

I. Sauvegarde d'une session Matlab

La commande :

```
>>diary nomfich.txt;
```

crée une copie de la session Matlab et de toutes les commandes exécutées dans le fichier nomfich.txt (excepté les graphiques) :

```
>>diary nomfich.txt;
>> …
>> …
…
diary off;
```

Exemple :

```
>> diary session1.txt
>> A=[1 2 3; 4 5 6]
>> B=[7 8 9; 10 11 12]
>> C=[13 14; 15 16]
>> D='Bonjour!'
>> diary off
```

Ouvrez le fichier **`nomfich.txt`**dans un éditeur de texte et supprimer les lignes de texte superflues.

Remarque :

Le fichier de sauvegarde peut être créé dans n'importe quel éditeur de texte : word (**`diary nomfich.doc`**), éditeur Matlab (**`diary nomfich.m`**), etc.

II.1. Sauvegarde des variables dans un fichier '.mat'

La commande :

```
>> save
```

permet de sauvegarde toutes les variables dans l'espace de travail dans le fichier **matlab.mat** (par défaut, l'extension des fichiers données matlab est .mat).

La commande :

```
>> save data
```

sauvegarde toutes les variables dans l'espace de travail dans le fichier **data.mat**.

La commande :

```
>> save('data')
```

est équivalente à la commande : >> **save** data

La commande :

```
>> savedata A B
```

Sauvegarde les variables A et B dans le fichier **data.mat**

On peut aussi spécifier le format de sauvegarde. Voici les différents formats possibles dans Matlab :

'-mat' Binary MAT-file format (default).

'-ascii' 8-digit ASCII format.

'-ascii', '-tabs' Tab-delimited 8-digit ASCII format.

'-ascii', '-double' 16-digit ASCII format.

'-ascii', '-double', '-tabs' Tab-delimited 16-digit ASCII format.

II.2. Importation des fichiers `.mat`

On peut importer des données vers l'espace de travail de deux manières :

a) Utiliser l'assistant d'importation :

- facilité d'utilisation
- opère sur la plupart des fichiers de données numériques

Etape 1 :

`menu File → Import Data`

Pour les anciennes versions :

`menu File → Load Workspace`…

ou exécuter la commande :

```
>> uiimport
```

Etape 2 :

sélectionner un fichier→ clic sur le bouton 'Ouvrir'

Etape 3 :

Sélectionner les variables à importer puis cliquer sur le bouton 'Finish'.

b) Utilisation de la fonction Matlab `"load"`

Pour importer les variables ou données enregistrées dans un fichier **`.mat`** (**`my_data.mat`** par exemple), on exécute la commande :

```
>> load my_data
```

Les données sont alors rappelées vers l'espace de travail (sans être affichées dans la fenêtre de commande).

III. Sauvegarde et importation des fichiers ASCII

Il est aussi possible de sauvegarder les variables dans un fichier ASCII et lui donner un format désirée. Elles pourront ensuite être lues par d'autres logiciels(par exemple, EXCEL) et avec MATLAB (en utilisant aussi la fonction load).

Exemples :

```
>> save exsave1.txt A -ASCII
>> save exsave2.txt A B –ASCII% A et B doivent avoir le même nombre
                              % d'éléments dans chaque ligne
>> save exsave3.txt D -ASCII% (sauve le code ASCII équivalent aux
                            % caractères)
```

On peut aussi utiliser l'extension **.dat** (data) au lieu du **.txt**.

Pour importer un fichier ASCII, il suffit d'exécuter la commande par exemple:

```
>> load exsave2.txt
```

Dans ce cas, le contenu du fichier `exsave2.txt` sera contenu dans une seule variable nommée exsave2.

La commande :

```
>> A=load('exsave2.txt')
```

retourne le contenu du fichier `exsave2.txt` dans la variable A.

IV. Importation des données avec entête

Pour importer des données avec entête, la commande **"dlmread"** est très pratique.

Exemple :

A l'aide du bloc-notes, créer le fichiet**data.txt** contenant les données suivantes :

Month	High	Low	Average
1	7.42	1.02	4.22
2	10.48	2.21	6.35
3	13.36	3.64	8.50
4	15.83	5.20	10.51
5	19.54	8.29	13.92
6	23.23	11.55	17.39
7	26.51	13.57	20.04
8	26.74	13.77	20.26
9	23.63	11.02	17.32
10	17.82	7.19	12.51
11	11.48	4.19	7.83
12	7.55	1.53	4.54

Utiliser la commande suivante pour importer ces données dans l'espace de travail Matlab :

```
>> M = dlmread('data.txt','',1)
```

Représenter les courbes de variations des minima (symbol o noir), moyennes (symbol * vert) et des maxima (symbol + rouge).

V.1. Sauvegarde des donnéesbinaires

Les fonctions utilisées dépendent du format des données :

- **auwrite** : exporte des données audio dans le format '**.au**'.
- **wavwrite** : exporte des données audio dans le format'**.wav**'. Windows
- **aviwrite** : exporte des données audio-vidéo dans le format '**.avi**'
- **imwrite** : exporte les données images dans différents formats (**.jpeg,.tiff, .bmp, .giff,** …)
- **xlswrite** : exporte les données dans le format Excel'**.xls**'.

…

V.2. Importation des données binaires

- **auread :** importe des données audio de format Sun Microsystems '**.au**'
- **wavread :** importe des données audio de format Windows '**.wav**'
- **aviread :**importe des données audiovisuelles de format AVI '**.avi**'
- **imread :** importe des données images de différents formats
- **load :** importe des données variables Matlab de format '**.mat**'
- **xlsread :** importe des données de format classeur Excel

…

Enfin, MatLab sait utiliser les fonctions de manipulationdes fichiers type C. Il s'agit des fonctions classiques genre **fopen**, **fclose**, **fwrite**, **fseek**, **fprintf**, etc. On peut ainsi lire ou générer un format binaire ou n'importe quel format exotique.

Sauvegarde des graphiques

Une figure peut être sauvegardée sous plusieurs formats :

- sous un format propre àmatlab avec l'extension **.fig**. Pour cela, cliquer sur la commande "**Save as**" du menu **File** de la fenêtregraphique et entrer un nom de fichier avec l'extension **.fig**.Un tel fichier peut être visualisé en utilisant la commande **Open** du menu **File** d'une fenêtre graphique.
- sous un format PostScript en utilisant la commande **Export** du menu **File** d'une fenêtre graphique, ou, plus manuellement, en tapant **print -dps*nomfichier***. Dans cecas, un fichier **nomfichier.ps** est créé dans le répertoire courant. On peut visualiser les fichiers PostScript en utilisant le logiciel GhostView (gv).

Partie VI

Introduction à Simulink

Simulink est l'extension graphique de MATLAB permettant de représenter les fonctions mathématiques et les systèmes sous forme de diagramme en blocs, et de simuler le fonctionnement de ces systèmes.

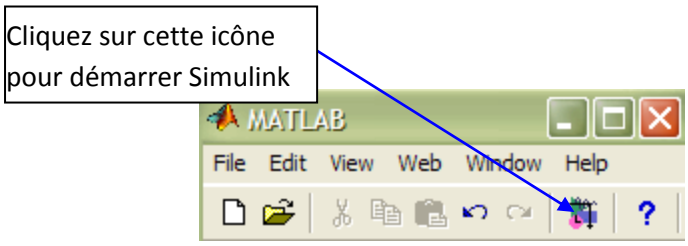
VI.1 Lancement de Simulink

Il y a deux manières de démarrer Simulink :

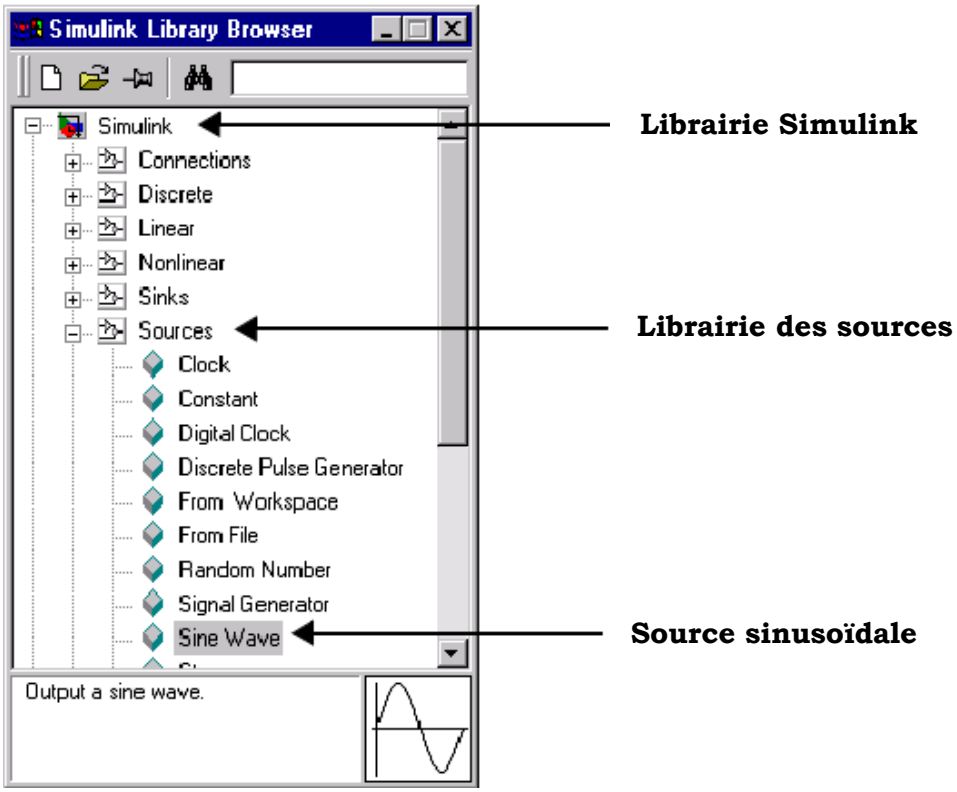
1) Dans la fenêtre de commande Matlab, exécuter la commande :

```
>> simulink
```

2) Dans la fenêtre de commande de Matlab:



Vous verrez alors apparaître la fenêtre suivante :



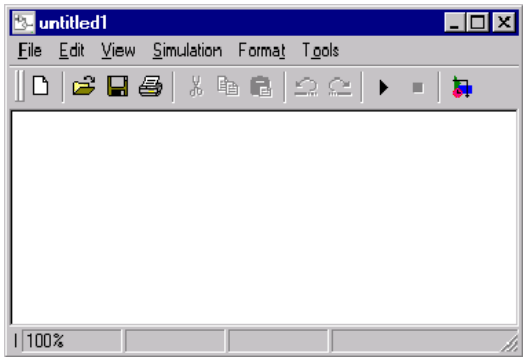
Remarque : L'apparence des fenêtres varie selon la version du logiciel Matlab utilisée.

IV.2 Construction d'un modèle Simulink

Pour créer un nouveau modèle, on ouvre une fenêtre de modèle Simulink. Pour cela,on peut soit :

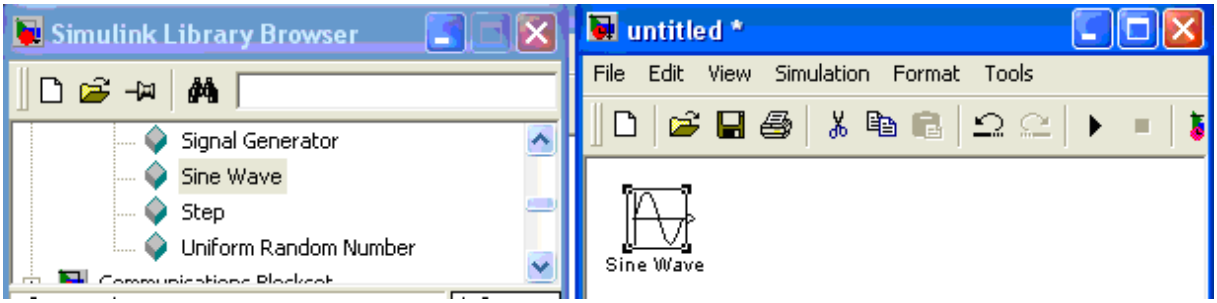
- cliquez sur l'icône  (de la fenêtre "Simulink Library Browser") ou
- ouvrez File→New→Model ou
- utilisez le raccourci Ctrl+N.

Une nouvelle fenêtre de modèle Simulink apparaît :



IV.2.1 Méthode de placement d'un composant

Ouvrez une nouvelle fenêtre de travail Simulink comme c'est indiqué ci-dessus. Dans la fenêtre « Simulink Library Browser » cliquez sur la librairie« Simulink » puis sur la sous-librairie« Sources », sélectionnez alors le composant « Sine Wave » et en maintenant l'appui sur le bouton gauche de la souris, faite glisser l'élément dans la fenêtre de travail et relâchez le bouton. Vous obtenez alors le résultat



Glissez ensuite dans la même fenêtre, le composant « Scope » se trouvant dans la sous-librairie « Simulink\Sinks ».

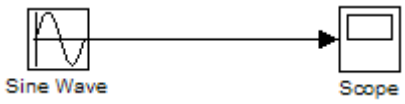
IV.2.2 Réalisation des connexions

Nous allons maintenant connecter la source sinusoïdale au scope. Pour réaliser la connexion entre des composants on procède de la manière suivante:

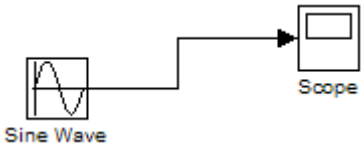
On sélectionne avec la souris, le symbole **>** situé sur un composant, on maintient l'appui sur le bouton et on tire le lien vers le symbole **>** de l'autre composant.

On peut aussi le faire de la manière la plus simple suivante : cliquer sur le composant de départ (de symbole **>**) et en maintenant la touche Ctrl du clavier enfoncée, cliquer sur le composant destination (de symbole **<**)

Réaliser alors la connexion suivante :



Si les deux composants ne sont pas alignés, la connexion entre eux n'est pas une droite. Ceci se fait automatiquement, toutefois on peut le réaliser nous même car il suffit de relâcher le bouton de la sourie, de maintenir à nouveau appuyé tout en se déplaçant dans la nouvelle direction :



Remarque :

- Il faut toujours vérifier que la connexion est bien établie :

Connexion bien établieConnexion mal établie

- Vous pouvez cliquez deux fois en importe quel point de la fenêtre de travail pour ajouter du texte.
- Pour changer le nom d'un composant, il suffit de cliquer une seule fois sur son nom d'origine puis entrer le nouveau nom.

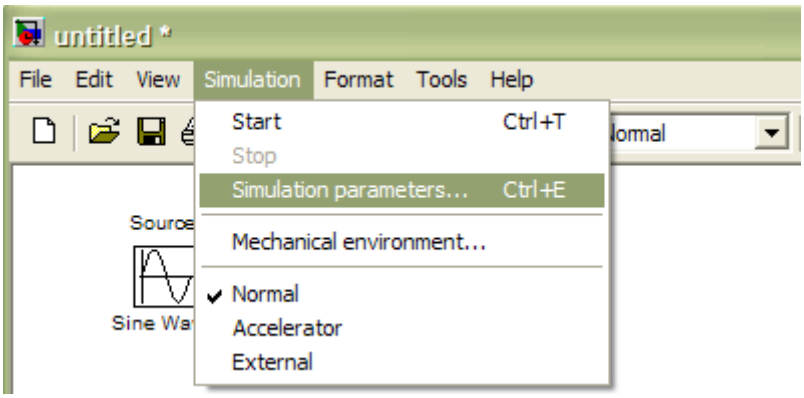
IV.2.3 Paramétrage des composants

En double cliquant sur un composant, on ouvre sa fenêtre de paramétrage. Cette fenêtre contient souvent une description du fonctionnement du composant.

IV.2.4 Simulation

IV.2.4.1 Paramétrage de la simulation

Pour ouvrir la fenêtre de paramétrage de la simulation vous pouvez cliquer sur « Simulation » dans la barre de menus de la fenêtre de travail puis sur « Simulation parameters » ou cliquer simplement sur le raccourci Ctrl+E.



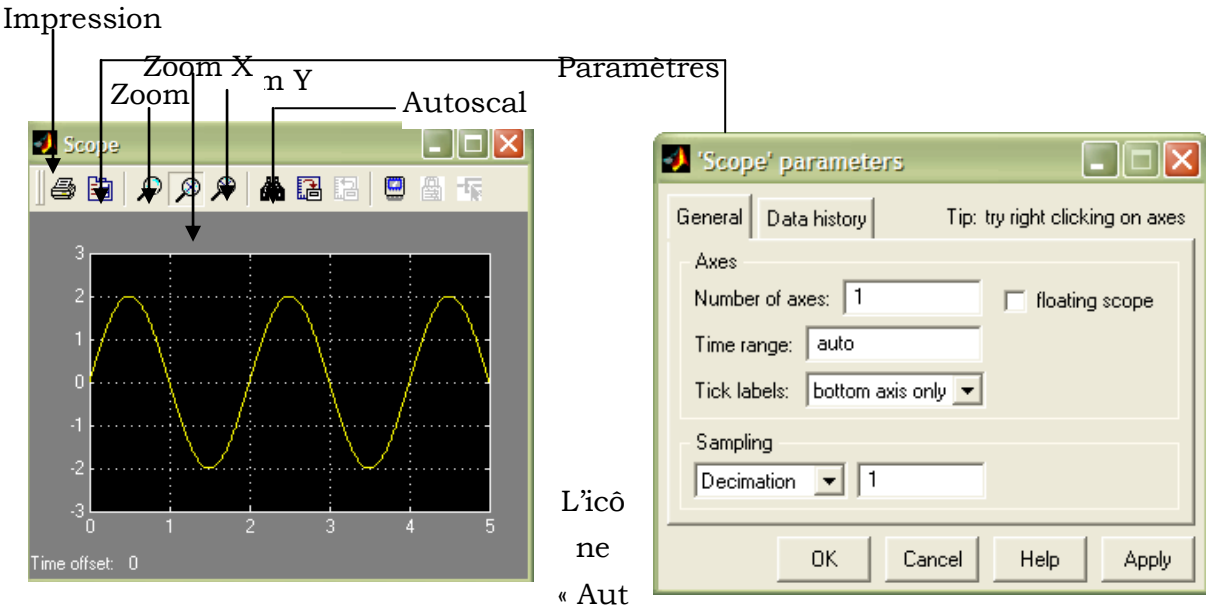
IV.2.4.2 Lancement de la simulation

Une fois le modèle construit, vous pouvez lancer la simulation à partir du menu « Simulation » → « Start » ou par le raccourci Ctrl+T, ou encore en cliquant l'icône  de la barre d'outils.

Un bip indique la fin de la simulation.

IV.2.4.3 Paramètres du Scope

Les principaux paramètres du Scope sont décrits sur la figure ci-dessous :



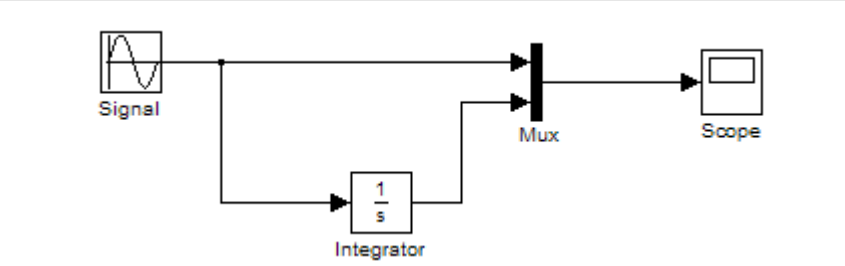
L'icône « Autoscal » permet de marier la courbe à la fenêtre de l'écran. Les icônes du Zoom permettent ensuite de zoomer soit sur les deux axes ou sur chacun d'entre eux.

Pour les paramètres du scope, on utilise surtout le paramètre « Number of axes » qui nous permet de subdiviser l'écran en plusieurs fenêtres.

Exercice : Double cliquez sur le composant « Signe Wave ». Régler alors son amplitude à 2 et sa fréquence à 0.5 Hz. Laisser les autres paramètres à leurs valeurs par défaut.

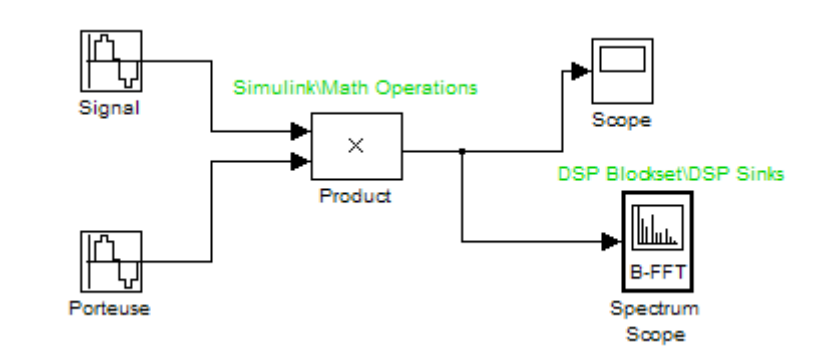
Dans la fenêtre « Simulation parameters » laisser l'instant de départ « Start time » à sa valeur par défaut (0) et régler l'instant d'arrêt « Stop time » à 5s puis valider en cliquant sur Ok. Simuler et consulter le Scope.

Exercice : Pour trouver un composant, on peut introduire son nom dans la barre de recherche de la fenêtre « Simulink Library Browser » et valider. Réaliser alors le modèle Simulink suivant :



Laisser tous les paramètres fixés par défaut et simuler. (Pour réaliser un nœud dans un circuit, sélectionner la connexion dans laquelle vous voulez placer le nœud, placer le curseur dans l'endroit du nœud, maintenir la touche Ctrl du clavier enfoncée et maintenant l'appui sur le bouton gauche de la sourie tirer le lien vers le symbole **>** de l'autre composant).

Exercice : Réaliser le modèle Simulink suivant :



Les paramètres des différents composants sont les suivants :

- Source Signal : Amplitude=1 ; Fréquence=100rad/sec ; Sample time=0.001.
- Source Porteuse : Amplitude = 1 ; Fréquence = 1000 rad/sec ; Sample time= 0.001.
- Spectrum Scope :Buffer size = 128; Buffer overlap = 64; Frequency units = Hertz; Frequency range = [-Fs/2 Fs/2]; Amplitude scaling = Magnitude.

Choisissez un temps de simulation de 10s. Simuler et consulter les Scopes.