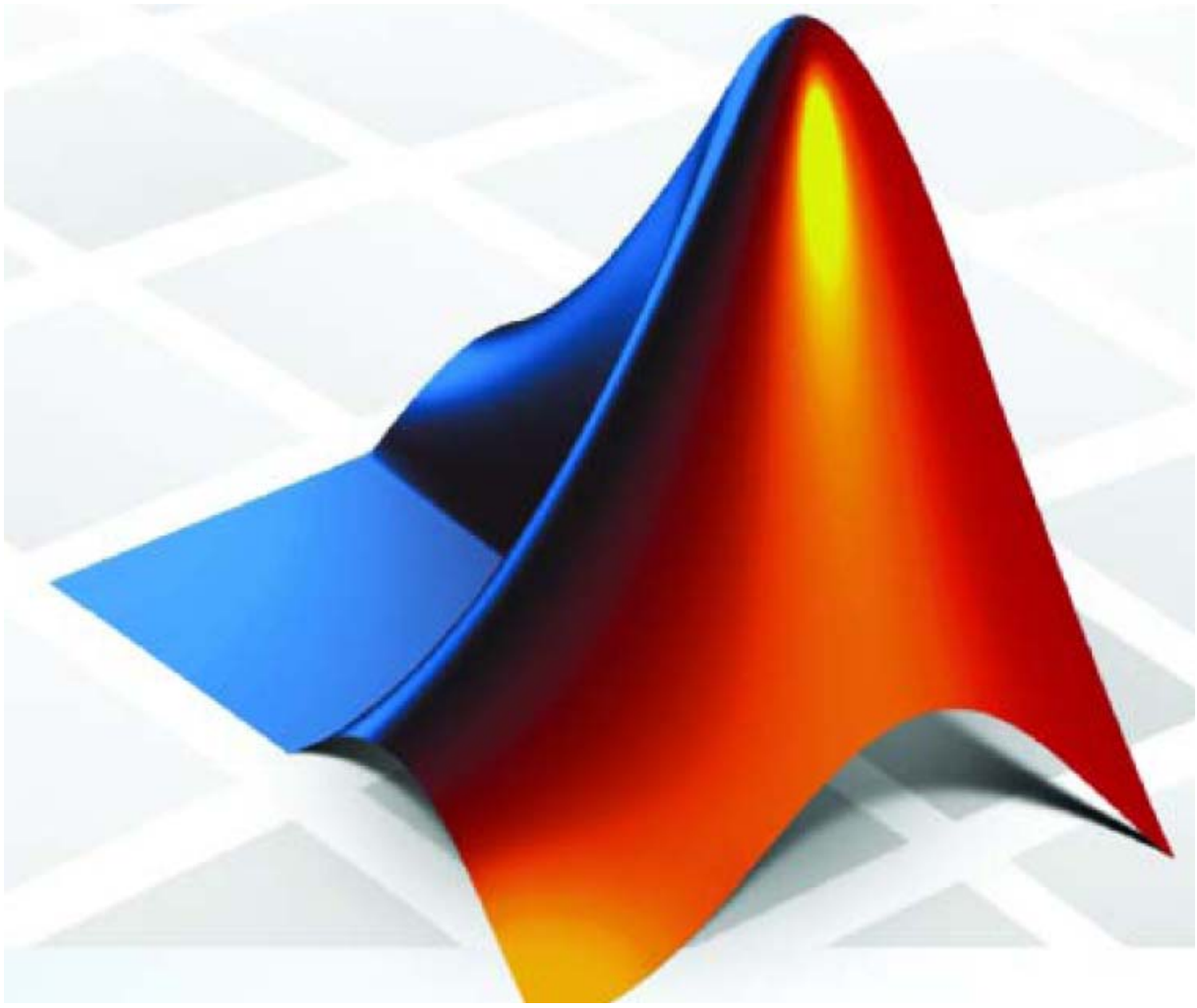
Université Sidi Mohammed Ben Abdallah
Faculté Des Sciences et Techniques Fès
Département Génie Electrique

LICENCE ETI

Electronique, Télécommunications et Informatique

Introduction à MATLAB et Simulink



Pr. Abdellah MECHAQRANE

Année Universitaire 2007-2008

Introduction

MATLAB (abréviation de **MATrix LABoratory**) est un langage de haute performance. Il intègre calcul, visualisation (graphiques) et programmation dans un environnement facile à utiliser et où les problèmes et les solutions sont exprimés en notation mathématique familière.

MATLAB est un système interactif dont les variables ou éléments de la base de données ne nécessitent pas de dimensionnement. Cela vous permet de résoudre beaucoup de problèmes techniques, notamment ceux à formulations vectorielle ou matricielle, en une fraction du temps qu'il ne faudrait pour écrire un programme dans un langage non-interactif tels que C ou Fortran.

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 : Traitement du signal, Traitement d'images, Télécommunications, Statistiques, Optimisation, Identification, Contrôle de systèmes, …

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 en langage MATLAB).

Avertissement :

Ce document a pour but de vous initier à Matlab et de vous montrer l'intérêt et la facilité de son utilisation. Cependant, un tel document ne peut en aucun cas présenter toutes les fonctionnalités de ce puissant logiciel. Il vous est donc vivement conseillé de consulter sa documentation disponible en ligne au site web :

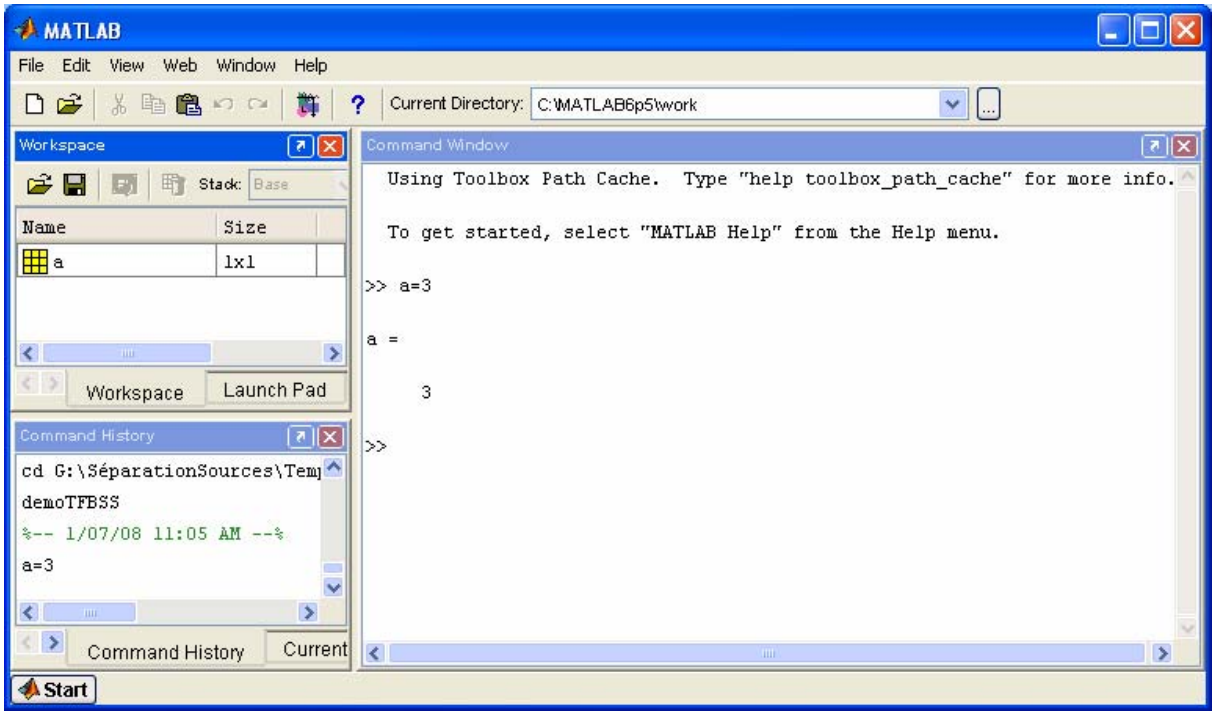
http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.shtml

I. Le système MATLAB

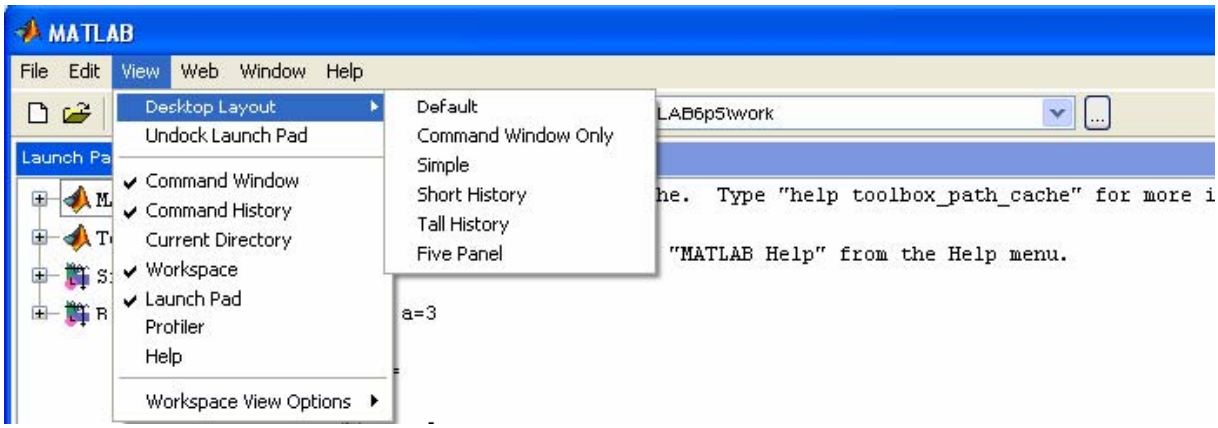
Le système MATLAB se compose de cinq parties principales:

- **Environment Development :**

Il s'agit d'un ensemble d'outils qui vous aident à utiliser les fonctions et fichiers MATLAB. Beaucoup de ces outils sont des interfaces graphiques. Les principaux outils sont : la fenêtre de commande (Command Window), un historique des commandes (Command History), un éditeur et un débogueur et les navigateurs pour visualiser l'aide, l'espace de travail, des fichiers, et le chemin de recherche (Search path).



L'apparence de la fenêtre principale peut être changée en utilisant la commande « View » de la barre d'outils :



➢ **La bibliothèque de fonctions mathématiques MATLAB** :

Il s'agit d'une vaste collection d'algorithmes de calcul allant des fonctions élémentaires comme la somme (sum), sinus (sin), cosinus (cos), …, à des fonctions plus sophistiquées comme l'inversion des matrices (inv), le calcul des valeurs propres et des vecteurs propres d'une matrice (eig), fonctions de Bessel (bessel), transformées de Fourier rapides (fft), ….

➢ **Le langage MATLAB** :

Il s'agit d'un langage matriciel de haut avec des déclarations de contrôle de boucles, des fonctions, des structures de données, des entrées / sorties et des fonctionnalités de programmation orientée objet.

Et le plus important est que Matlab est un **langage interprété** : toute commande tapée dans la fenêtre de commande est immédiatement exécutée (après la frappe de **return**). Il n'est pas nécessaire de compiler un programme avant de l'exécuter :

```
>> 2+2

ans = 4

>> 5^2

ans = 25
```

Quand vous ne spécifiez pas une variable de sortie, MATLAB utilise la variable *ans* (abréviation de answer (réponse)) pour stocker les résultats pour un éventuel calcul.

- **Graphiques :**

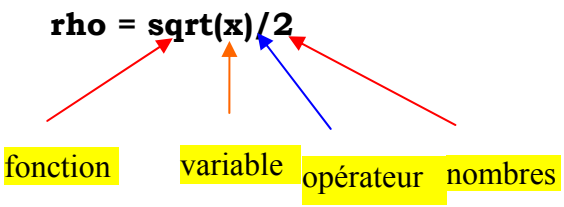
MATLAB possède une facilité extensive permettant d'afficher des vecteurs et des matrices sous forme de graphique. Il comprend des fonctions de haut-niveau pour la visualisation bidimensionnelles et tridimensionnelles de données, d'images, d'animations. Il comprend également des fonctions de bas niveau, ce qui vous permet de personnaliser entièrement l'apparence des graphiques ainsi que de construire des interfaces graphiques complètes pour vos applications MATLAB.

I.1 Les Expressions

Comme la plupart des autres langages de programmation, MATLAB fournit des expressions mathématiques, mais contrairement à la plupart des langages de programmation, la totalité de ces expressions impliquent des matrices. Les éléments constitutifs sont des expressions :

- Nombres
- Opérateurs
- Fonctions
- Variables

Exemples d'expressions :



I.2 Nombre

MATLAB utilise la notation décimale classique : un point pour séparer la partie entière de la partie décimale et un signe moins pour les nombres négatifs; l'absence du signe indique que le nombre est positif. Il utilise la lettre *e* pour spécifier une puissance de dix. Pour les nombres imaginaires MATLAB utilise indifféremment les suffixes i ou j. Exemples de nombres :

```
3              -99            0.0001

9.6397238    1.60210e-20    6.02252e23

1i             -3.14159j      3e5i
```

Tous les chiffres sont stockés en utilisant le **format long** spécifié par l'IEEE en virgule flottante standard. Les nombres en virgule flottante ont une précision finie de 16 chiffres décimaux significatifs et un intervalle de variation d'environ 10^{-308} à 10^{+308}.

Vous trouvez à la fin de l'annexe 3 l'inventaire des différents formats utilisés par MATLAB.

I.3 Opérateurs

MATLAB utilise les opérateurs usuels suivants pour manipuler les expressions :

Fonctions	**Définition**
+	Addition
-	Soustraction
*	Multiplication
^	Puissance
/	Division
\	Division à gauche
‘	Transposé conjugué

Il utilise aussi des opérateurs logiques et relationnels :

Fonctions	Définition
< >	différent de
<= >=	inférieur (supérieur) ou égal
==	égal
~=	différent
&	et logique
|	ou
~	complément logique (not)
xor	ou exclusif

I.4 Fonctions

MATLAB fournit un grand nombre de fonctions mathématiques élémentaires standard, y compris sin (sinus), cos (cosinus), tan (tangente), abs (valeur absolue ou module), sqrt (racine carrée), exp (exponentiel), …. La plupart de ces fonctions acceptent des arguments complexes. Pour obtenir une liste des fonctions mathématiques élémentaires, tapez :

```
>> help elfun
```

MATLAB fournit également de nombreuses fonctions mathématiques plus avancées, y compris les fonctions de Bessel et gamma. Pour obtenir une liste de ces fonctions, tapez :

```
>> help specfun
```

Plusieurs fonctions spéciales fournissent des valeurs de constantes utiles :

Fonctions	Définition
pi	3.14159265...
i	Nombre imaginaire $\sqrt{-1}$
j	Comme i
eps	Précision relative des nombres réels (distance entre 1.0 et le nombre réel flottant le plus proche), 2^{-52}
realmin	Le plus petit nombre réel, 2^{-1022}
realmax	Le plus grand nombre réel, $(2\text{-eps})^{1023}$
inf	Infinie
NaN	Not-a-Number (ce n'est pas un nombre) : 0/0=NaN
date	Date

I.5 Variables

Matlab travaille essentiellement avec un seul type d'objet : une matrice (de dimension NxM). Une matrice 1×1 est interprétée comme un scalaire, celle ayant une seule ligne (1xN) ou une seule colonne (Nx1) comme un vecteur.

Ainsi, dans Matlab, il n'est pas nécessaire de déclarer le type (réelle, complexe, ...) ou la dimension (scalaire, matrice) de la variable que l'on manipule, y compris les tableaux. Lorsque MATLAB rencontre un nouveau nom de variable, il crée automatiquement la variable et alloue la capacité appropriée de stockage. Si la variable existe déjà, MATLAB modifie son contenu et, si nécessaire, il alloue une nouvelle capacité stockage.

Les noms des variables sont composés d'une lettre, suivie d'un nombre quelconque de lettres, chiffres ou de sous-tirets. MATLAB utilise seulement les 31 premiers caractères d'un nom de variable et établit une distinction entre majuscules et minuscules : A et a ne sont pas la même variable. Pour visualiser la matrice affectée à une variable, il suffit d'entrer le nom de la variable.

Aussi, il faut éviter de mettre dans le nom des variables des opérateurs Matlab tels que -, +, *, /, \ , &, |, ….

I.5.1 Variables réelles

En Matlab, chaque nombre réel est représenté en virgule flottante avec une double précision, c'est-à-dire 8 octets.

Par exemple,

```
>> num_students = 25
```

crée une variable (matrice 1x1) nommée num_students et stocke la valeur 25 dans le seul élément.

```
>> x = pi/4
```

assigne à la variable x la valeur pi/4.

Remarques :

- Un point virgule « **;** » à la fin de la ligne permet de ne pas afficher ce résultat. 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 taper plusieurs commandes par ligne, séparées par une virgule ou un point virgule.

```
>> x = pi/2 ; y = sin(x) ;
```

- Lorsqu'une ligne de commande est trop longue on peut l'interrompre par trois points « **…** » et la poursuivre à la ligne suivante.
- On peut aussi mettre des commentaires dans une ligne de commande à l'aide du signe " **%** ".

I.5.2 Variables complexes

Matlab travaille indifféremment avec des nombres réels ou complexes. Par défaut, les variables i et j sont assignées à la valeur complexe. Naturellement, si vous redéfinissez la variable i ou j avec une autre valeur elle n'aura plus la même signification.

```
>> j = 3 + 2*i

    j = 3.0000 + 2.0000i
```

Les fonctions usuelles de manipulation des nombres complexes sont prédéfinies dans Matlab : real, imag, abs, angle (en radian), conj.

```
>> r = abs(j) ;

>> theta = angle(j) ;

>> y = r*exp(i*theta) ;
```

I.5.3. Vecteurs

Les vecteurs et matrices sont souvent utilisés dans Matlab pour représenter les données.

Rapelons que dans Matlab, toute variable est une matrice (scalaire : matrice 1x1, vecteur : matrice 1xN ou Nx1).

I.5.3.1 Vecteurs

- Un **vecteur ligne** peut être créé de la façon suivante :

```
>> vl=[1, 2, 3]
```

ou

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

C'est-à-dire que, dans un vecteur ligne, les éléments sont séparés par une virgule ou un espace.

- Un **vecteur colonne** peut être créé de la façon suivante :

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

C'est-à-dire que, dans un vecteur colonne, les éléments sont séparés par un point virgule.

Remarque :

- Les éléments d'un vecteur (ou plus généralement d'une matrice) peuvent être des expressions mathématiques de Matlab :

```
>> x = [ -1.3, sqrt(3), (1+2+3)*4/5, 2*exp(i) ]
```

- Les éléments d'un vecteur peuvent être référencés par leurs indices :

```
>>x(2)
     ans =
          1.7321
>> x(4)
     ans =
          1.0806 + 1.6829i
```

- On peut avoir des informations sur la longueur d'un vecteur à l'aide de commande « length » :

```
>> length(x)
```

- On peut avoir le transposé ou le transposé conjugué d'un vecteur :

```
>> x' % Transposée conjuguée de x

>> x.' % Transposée de x
```

I.5.4 Matrices

Pour créer une matrice dans MATLAB, on doit suivre les conventions de base suivantes :

- ☞ séparer les éléments d'une ligne avec des espaces ou des virgules.
- ☞ Utilisez un point-virgule « **;** » pour indiquer la fin de chaque ligne.
- ☞ Entourez l'intégralité de la liste des éléments avec des crochets **[ ]**.

Exercice **:** *Créer dans la fenêtre de commande MATLAB, la matrice suivante :*

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

Une fois que vous avez écrit la matrice A (en respectant les consignes ci-dessus), et valider en appuyant sur la touche « Entr » du clavier. Aussitôt, la matrice A est mémorisée automatiquement dans l'espace de travail MATLAB (MATLAB workspace).

I.5.4.1 Quelques opérations sur les matrices

On peut se référer à la matrice ci-dessus simplement par la lettre A. Essayer alors les commandes suivantes :

```
>> sum(A)
```

La commande *sum(A)* permet de calculer un vecteur ligne contenant la somme des colonnes de A.

Qu'en est-il de la somme des lignes ? Puisque MATLAB a une préférence pour le travail avec les colonnes d'une matrice, donc la façon la plus simple pour obtenir la somme des lignes est de ***transposer la matrice***, puis calculer la somme comme ci-dessus et de transposer ensuite le résultat. L'***opération de transposition*** est réalisée par une apostrophe, « **'** ». Cette

opération inverse la matrice sur sa diagonale principale et transforme un vecteur ligne en un vecteur colonne.

Exercice *: a) Calculer la somme des lignes de la matrice A.*

b) Calculer la somme de la diagonale de A (diag(A)).

L'autre diagonale, la soi-disant antidiagonale, n'est pas si importante mathématiquement, de sorte que MATLAB ne dispose pas d'une fonction préprogrammée pour elle. Mais une fonction initialement destinée à être utilisées dans les graphiques, `fliplr`, renverse une matrice de gauche à droite.

Exercice *: Calculer la somme de l'antidiagonale de la matrice A. Conclure. Savez-vous quel nom est donné à une matrice telle que A ?*

Les sections suivantes continueront à utiliser la matrice A pour illustrer les capacités supplémentaires MATLAB.

I.5.4.2 Indexation des éléments d'une matrice

L'élément dans la ligne i et la colonne j de A est noté A(i, j). Par exemple, A(4,2) est le nombre de la quatrième rangée et de la deuxième colonne. Pour notre carré magique, A(4,2) est 15. Ainsi, pour calculer la somme des éléments dans la quatrième colonne de A, on peut écrire :

```
>> A(1,4) + A(2,4) + A(3,4) + A(4,4)
```

Mais ceci n'est pas la façon la plus élégante pour calculer la somme d'une colonne unique (imaginer le cas d'une très grande matrice. Nous allons voir ci-dessous comment calculer la somme d'une colonne (ou d'une ligne) d'une matrice d'une façon plus simple.

Il est également possible de se référer à des éléments d'une matrice avec un seul indice, A (k). Dans ce cas la matrice est considérée comme un long vecteur colonne formé de colonnes de la matrice originelle. Donc, pour notre

carré magique, A (8) est une autre manière de se référer à la valeur stockée dans A (4,2).

En résumé, on a deux façon d'indexer une matrice, soit pour la matrice A :

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

Exercice *: Créer une matrice B contenant uniquement les éléments centraux de A, c'est-à dire :*

$$B = \begin{bmatrix} \cancel{16} & \cancel{3} & \cancel{2} & \cancel{13} \\ \cancel{5} & \mathbf{10} & \mathbf{11} & \cancel{8} \\ \cancel{9} & \mathbf{6} & \mathbf{7} & \cancel{12} \\ \cancel{4} & \cancel{15} & \cancel{14} & \cancel{1} \end{bmatrix}$$

Si vous essayez d'utiliser un élément extérieur à la matrice, Matlab vous envoie une erreur. Exemple :

```
>> t = A(4,5)
```

??? Index exceeds matrix dimensions.

En revanche, si vous stockez une valeur dans un élément en dehors de la matrice, la taille augmente pour accueillir le nouveau venu.

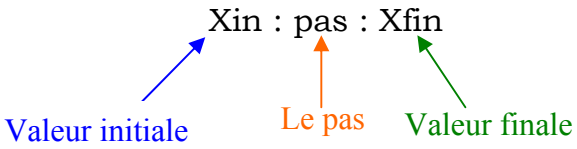
Exercice *: Appliquer et commenter les commandes suivantes :*

```
>> X = A;

>> X(4,5) = 17
```

I.5.4.3 L'opérateur « : »

« : » est l'un des plus importants opérateurs de MATLAB. Il se produit dans plusieurs formes différentes. Sa syntaxe est :



Exercice *: Appliquer et commenter les commandes suivantes :*

```
>> 1:10

>> 0:pi/4:pi

>> 100:-7:50
```

Une autre fonction MATLAB permet de générer des valeurs régulièrement espacées dans un intervalle donné, c'est la fonction **`linspace`** :

`linspace(x,y)` : génère 10 valeurs régulièrement espacées entre x et y.

`linspace(x,y,N)` : génère N valeurs régulièrement espacées entre x et y.

L'opérateur **`:`** est souvent utilisé pour se référer à des parties d'une matrice. Ainsi :

```
>> y = A(1:k, j)
```

représenterait un vecteur colonne contenant les éléments de la 1ère à kième ligne de la jème colonne.

Remarque : lorsqu'on veut utiliser tous les éléments d'une ligne ou d'une colonne, il suffit d'écrire :

`>> y = A( : , j )`	est le vecteur colonne constitué par tous les éléments de la colonne j de la matrice A
`>> y = A( j , : )`	est le vecteur ligne constitué par tous les éléments de la ligne j de la matrice A

Exercice *: a) Calculer la somme des éléments dans la quatrième colonne de la matrice A.*

Le dernier élément d'une ligne ou d'une colonne peut être indexé par la variable **end**. Essayer les commandes suivantes :

```
>> A(:,end)

>> A(end,end-1)
```

I.5.4.4 Suppression de lignes et de colonnes

Vous pouvez supprimer des lignes et des colonnes d'une matrice en utilisant juste une paire de crochets.

Pour conserver notre matrice A d'origine, on effectuera les calculs sur une matrice X, telle que :

```
>> X=A ;
```

Pour supprimer la 2ème colonne de la matrice X, il suffit d'exécuter la commande suivante :

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

Exercice *: a) Supprimer la 3ème colonne de la matrice X.*

b) supprimer les 1ère et 2ème colonnes de la matrice X=A.

c) supprimer les 3ème et 4ème lignes de la matrice X=A.

Si vous supprimez un élément unique à partir d'une matrice, le résultat n'est plus une matrice. Ainsi, une expression telle que :

```
>> X(1,2) = []
```

Entraînera une erreur.

I.5.4.5 Opérations sur les matrices

Le tableau ci-dessous représente les principales opérations mathématiques que l'on peut réaliser à l'aide de MATLAB :

Opérations	Forme dans Matlab	Commentaires
Addition	A+B	Dimensions doivent être les mêmes
Soustraction	A-B	Dimensions doivent être les mêmes
Multiplication de matrices (élément par élément)	A.*B	La multiplication se fait élément par élément. Les deux matrices doivent être de mêmes dimensions, ou l'une d'elles peut être un scalaire.
Multiplication de matrices	A*B	Le nombre de colonnes dans "a" doit être le même que le nombre de rangées dans "b"
Division de matrices (élément par élément)	A./B	La division se fait élément par élément. Les deux matrices doivent être de mêmes dimensions, ou l'une d'elles peut être un scalaire.
Division de matrices	A/B	Équivalent à "a * inv(b)"
Exposant sur matrices	A.^B	Se fait élément par élément

Exercice : *a) Calculer le produit de la matrice A par elle même.*

b) Calculer le produit élément par élément de la matrice A.

b) Multiplier la 3ème colonne de A par 2.

I.5.4.6 Matrices particulières

Le tableau ci-dessous décrit quelques fonctions MATLAB permettant de générer quelques matrices particulières :

Fonctions	Définition
zeros(n)	Génère une matrice de zéros de grandeur nxn
zeros(n,m)	Génère une matrice de zéros de grandeur nxm
ones(n)	Génère une matrice de "1" de grandeur nxn
ones(n,m)	Génère une matrice de "1" de grandeur nxm
eye(n)	Génère une matrice identité de grandeur nxn
eye(n,m)	Génère une matrice identité de grandeur nxm
rand(n,m)	Matrice nxm formée d'éléments aléatoires uniformément distribués
randn(n,m)	Matrice nxm formée d'éléments aléatoires normalement distribués
magic(n)	Crée un carré magique de dimension n

Exercice **:** *Essayer les commandes suivantes :*

```
>> Z = zeros(2,4)
>> F = 5*ones(3,3)
>> R = randn(4,4)
```

Ré-exécuter la dernière instruction (R=…). Comparer la nouvelle matrice avec l'ancienne. Qu'est ce que vous constatez ? Pouvez-vous justifier cette différence ?

Pour connaître les dimensions (lignes, colonnes) d'une matrice, on utilise la commande MATLAB : **size** (exemple : size(R))

I.5.4.7 Concaténation

La concaténation est le processus d'assemblage de petites matrices pour en faire une plus grande.

Exercice **:** *Utiliser la matrice A et construisez la matrice B définie par :*

```
B = [A  A+32; A+48  A+16]
```

Quelles sont les dimensions de la matrice B ?

I.5.4.8 Algèbre linéaire

Les opérations mathématiques sur les matrices font l'objet de l'algèbre linéaire.

Le symbole de multiplication, * , désigne la multiplication matricielle, c'est-à-dire les produits entre les lignes de la première matrice et les colonnes de la deuxième.

Exercice *: Soit la matrice :*

```
K=A'*A
```

Quelle est la particularité de la multiplication d'une matrice par son transposé ?

Exercice *: Calculer le déterminant de la matrice A :*

```
d=det(A)
```

Lorsque le déterminant d'une matrice est nul, cette matrice est dite singulière et elle n'admet pas de matrice inverse. Si vous essayez de calculer l'inverse avec :

```
>> X = inv(A)
```

vous obtiendrez le message d'avertissement :

```
Warning: Matrix is close to singular or badly scaled.
```

L'annexe 1 présente quelques opérations mathématiques intéressantes sur les matrices.

I.6 Manipulation des données

I.6.1 Construction de tableaux - Données Multidimensionnelles

Supposons que n est le vecteur colonne :

```
>> n=(0:9)'
```

Alors

```
>> pows = [n  n.^2  2.^n]
```

Retourne un tableau contenant n dans la 1ère colonne, le carré de n (n^2) dans la 2ème colonne et 2 à la puissance n (2^n) dans la 3ème colonne.

Les fonctions mathématiques élémentaires agissent sur des tableaux élément par élément. Ainsi :

```
>> pows1 = pows + 1
```

ajoute 1 à chacun des éléments du tableau pows, et :

```
>> log(pows1)
```

Calcule le logarithme népérien de chacun des éléments du tableau pows1.

I.6.2 Analyse statistique

MATLAB effectue l'analyse statistique des données suivant les colonnes. Chaque colonne d'un jeu de données représente une variable, et chaque ligne une observation. Le (i, j)ème élément est la ième observation de la jème variable.

Comme exemple, considérons un ensemble de données avec trois variables:

- La fréquence des battements cardiaques
- Poids
- Nombre d'heures d'exercice par semaine

Pour cinq observations, on a le tableau suivant :

```
D = [   72          134         3.2
        81          201         3.5
        69          156         7.1
        82          148         2.4
        75          170         1.2 ]
```

La première ligne contient la fréquence cardiaque, le poids et le nombre d'heures d'exercice pour le patient 1, la deuxième rangée contient les données du patient 2, et ainsi de suite. Maintenant, vous pouvez appliquer de nombreuses fonctions d'analyse de données MATLAB à cet ensemble de données. Par exemple, pour obtenir la moyenne et l'écart-type de chaque colonne, utilisez les fonctions :

```
>> mu = mean(D),

>> sigma = std(D)
```

Pour obtenir la liste des fonctions d'analyse des données disponibles dans MATLAB, tapez :

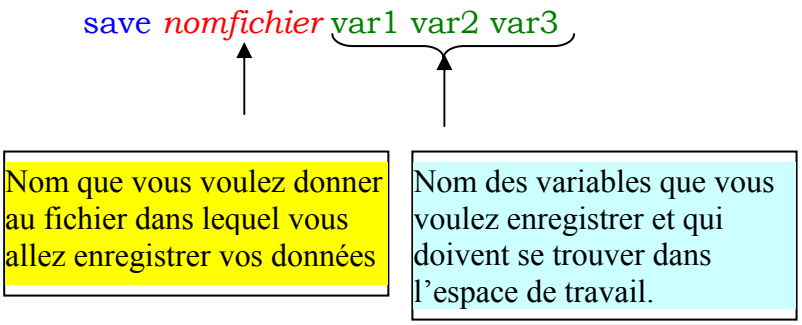
```
>> help datafun
```

Pour obtenir la liste des fonctions d'analyse statistique, tapez :

```
>> help stats
```

I.6.3 Enregistrement et rappel des données

Pour enregistrer des données à l'aide de Matlab, on utilise la commande **save** dont la syntaxe est la suivante :



Les données var1, var2 et var3 seront enregistrées dans le fichier « nomfichier » qui aura une extension « **.mat** » qui est l'**extension des fichiers de données enregistrés sous Matlab**.

Remarque :

Avant de procéder à l'enregistrement, on vérifier d'abord que les variables sont dans l'espace de travail. Ceci peut se faire à l'aide des commandes :

```
>> who
```

qui renvoie le nom des variables contenues dans l'espace de travail ou

```
>> whos
```

qui renvoie le nom de ces variables, leur taille, leur occupation en mémoire et leur classe.

Exercice **:** Enregistrer les matrices A et D dans un fichier appelé « myfirstmat ».

La commande :

```
>> clear X
```

Permet d'effacer la variable X de l'espace de travail. La commande :

```
>> clear
```

Efface quant-à elle toutes les variables contenues dans l'espace de travail.

Exercice **:** Effacer ensuite A de l'espace de travail. Vérifier qu'elle est bien effacée. (Calculer m=mean(D), sd=std(D) et d=det(D)).

Quelles sont les variables présentes dans l'espace de travail ?

Effacer toutes ces variables de l'espace de travail.

Lorsque vous aurez besoin des données enregistrées dans un fichier .mat, il suffit d'utiliser la commande **load** suivie du nom du fichier de données pour les recharger dans l'espace de travail. La syntaxe est la suivante :

```
load nomdufichier
```

Pour les fichiers .mat, il n'est pas nécessaire de préciser l'extension .mat.

Exercice : Recharger les données enregistrées dans le fichier « myfirstmat » dans l'espace de travail.

Quelles sont alors les variables présentes dans l'espace de travail ?

Remarque :

Matlab peut lire plusieurs formats de données:

load nomfichier.dat ; % données enregistrées sous le format .dat

load nomfichier.txt ; % données enregistrées sous le format .txt

Pour les fichiers .dat et .txt, il faut préciser l'extension.

xlsread('nomfichier') ;% lecture de données dans un fichier Excel.

imread('nomfichier', format) ;% lecture d'une image de format donnée.

Taper le **help** de ces fonctions pour connaître leur fonctionnement détaillé.

I.6.4 Matrices et scalaires

Matrices et scalaires peuvent être combinés de plusieurs façons différentes. Par exemple, un scalaire est soustrait à une matrice en le soustrayant de chaque élément. Il en est du même pour l'addition et pour la multiplication.

Exercice : *Pour centrer des données statistiques, il suffit d'en soustraire la valeur moyenne. A partir de notre matrice A, créer une matrice M dont la moyenne de chaque colonne est nulle.*

On peut attribuer un scalaire à l'ensemble des indices dans une gamme. Par exemple, la commande :

```
>> M(1:2,2:3) = 0
```

attribuera un zéro aux éléments compris entre les lignes 1 et 2 et les colonnes 2 et 3 de la matrice M.

I.6.5 Eléments répondant à une condition donnée

Dans la manipulation des données, on est parfois contraint de choisir uniquement des données particulières répondant à un critère (ou condition) donné. Dans MATLAB, ceci peut se faire à l'aide de la fonction **find**. Cette fonction détermine les indices (les positions dans la matrice) des éléments qui répondent à une même condition logique donnée. Dans sa forme la plus simple, `find` retourne un vecteur colonne. Transposer ce vecteur pour obtenir un vecteur ligne. Par exemple,

```
>> k = find(isprime(A))'
```

donne les indices des nombres premiers contenus dans la matrice A.

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

```
>> A(k)
```

Cette fonction utilise souvent les opérateurs logiques (§ I.3).

Exercice **:** *a) Trouver les éléments de A qui sont inférieurs à 7.*

b) trouver l'indice de l'élément de A qui est égal à 7.

c) Trouver les éléments de A qui sont inférieurs ou égal à 3 ou supérieurs ou égal à 13.

I.7 Chaîne de caractères

Matlab traite toutes les données sous forme matricielle, c'est-à-dire soit comme un tableau de nombres, soit comme un tableau de caractères. Par exemple, une phrase sans saut de ligne sera un vecteur de caractères.

Les chaînes de caractères sont délimitées par des apostrophes « ' » :

```
>> txt1 = 'Bonjour les amis'
>> txt1(1:8)
```

Pour récupérer une chaîne entrée par l'utilisateur au clavier, on utilise la fonction **input** avec l'argument **'s'** signifiant *string*. La commande input est utilisable en mode caractère (présence de l'argument **'s'**) ou numérique (absence de l'argument **'s'**).

```
>> txt2 = input('Tapez votre texte :    ','s')
```

L'exécution de l'instruction ci-dessus n'est accomplie qu'une fois la chaîne de caractères est écrite et validée par la touche « Entrée ».

L'affichage du texte est possible par les commandes **disp** et **sprintf** **:**

- **disp** permet d'afficher une variable ou une chaîne de caractères, ou toute combinaison licite des 2.

```
>> disp(A)
>> disp(txt2)
```

```
>> disp('Ceci est un texte')
```

Le mélange de variables numériques et de caractères se fait grâce à la fonction **num2str** (*number to string*) qui convertit un nombre en chaîne :

```
>> disp(['l''élément A(3,3) vaut : 'num2str(A(3,3))])
>> disp(['La matrice A vaut :' num2str(A)])
```

Cette dernière ligne n'est pas valide car la chaine de caractère est un vecteur ligne alors que la partie numérique est une matrice.

Pour résoudre ce problème, on utilise soit l'instruction :

```
>> disp( strvcat('La matrice A vaut :', num2str(A)) )
```

soit l'instruction :

```
>> disp('La matrice A vaut : '), disp(A)
```

La fonction **sprintf** est une fonction similaire à la fonction *printf* du langage C. Par rapport à la fonction **disp**, elle permet d'inclure plus facilement des résultats numériques au milieu de texte, et de choisir la précision d'affichage :

```
>> sprintf('A est une \nmatrice %dx%d',size(A,1),size(A,2))
```

Pour plus de détails sur cette fonction, consulter son help.

I.8 Contrôle de la fenêtre de commande et des entrées et sorties

Jusqu'ici, vous avez utilisé la ligne de commande MATLAB, tapé des fonctions et expressions et vous avez vu les résultats imprimés dans la fenêtre de commande. Cette section décrit :

- Les fonctions de **format**, pour contrôler l'apparence des valeurs affichées
- Suppression de l'affichage de la Sortie

- Entrer de longues déclarations
- Modification de la ligne de commande

I.8.1 La Fonction format

La fonction **format** contrôle le format d'affichage des valeurs numériques affichées par MATLAB. La fonction affecte seulement la façon dont les valeurs sont affichées, et non la façon dont MATLAB calcule ou enregistre ces valeurs. Pour contempler les différents formats ainsi que le résultat produit, on va considérer un vecteur x avec deux composantes de différentes amplitudes, soit :

```
>> x = [4/3 1.2345e-6];
```

pour voir l'affichage produit par le **format short**, il suffit de taper :

```
>> format short,x
```

Contempler l'effet des formats suivants :

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

I.8.2 Suppression de la réponse

Si vous tapez simplement une déclaration et appuyez sur la touche Retour ou Entrée, MATLAB affiche automatiquement les résultats à l'écran. Toutefois, si vous terminez la ligne par un point-virgule « **;** », MATLAB effectue le calcul mais n'affiche aucune sortie. Cela est particulièrement utile lorsque vous générez de grandes matrices ou lorsqu'on utilise de longs

programmes (l'affichage des résultats dans la fenêtre de commande ralenti l'd'exécution des programmes). Par exemple,

```
>> A1 = magic(6);
```

Matlab n'affiche pas A1, mais cette matrice est mémorisée dans l'espace de travail et elle peut être utilisée dans des calculs. Aussi, si vous voulez afficher A1, il suffit de taper son nom dans la ligne de commande et valider :

```
>> A1
```

I.8.3 Saisie d'une longue déclaration

Si une déclaration ne tient pas sur une seule ligne, utilisez trois points de suspension, **...**, suivi de « Retour » ou « Entrée » pour indiquer que la déclaration continue sur la ligne suivante. Par exemple,

```
>> s = 1 -1/2 + 1/3 -1/4 + 1/5 - 1/6 + 1/7 ...

                - 1/8 + 1/9 - 1/10 + 1/11 - 1/12;
```

Les espaces vides autour de =, + et - sont facultatifs, mais ils améliorent la lisibilité.

I.8.4 Touches de raccourci

Le tableau ci-dessous donne quelques raccourcis clavier qui facilitent la manipulation des instructions dans la fenêtre de commande.

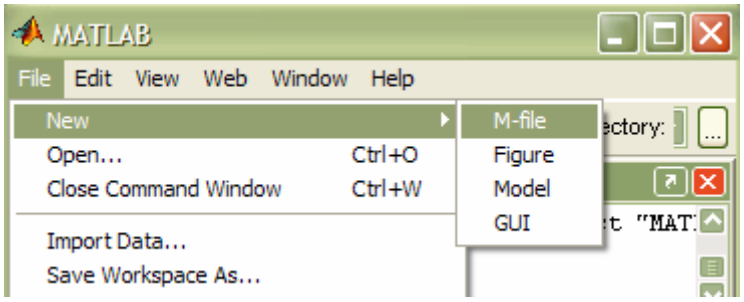
En particulier, pour revenir à une commande précédente sans la réécrire, il suffit d'appuyer une seule fois (s'il s'agit de la commande précédente) ou plusieurs fois (s'il s'agit d'une commande plus antérieure) sur la touche clavier (flèche) de déplacement vers le haut  .

Key	Control Key for MATLAB Standard (Emacs) Preference	Operation
↑	**Ctrl+P**	Recall *p*revious line. Works only at command line.
↓	**Ctrl+N**	Recall *n*ext line. Works only at the prompt if you previously used the up arrow or **Ctrl+P**.
←	**Ctrl+B**	Move *b*ack one character.
→	**Ctrl+F**	Move *f*orward one character.
Ctrl+→	None	Move *r*ight one word.
Ctrl+←	None	Move *l*eft one word.
Home	**Ctrl+A**	Move to beginning of current statement.
End	**Ctrl+E**	Move to *e*nd of current statement.
Ctrl+Home	None	Move to top of Command Window.
Ctrl+End	None	Move to end of Command Window.
Esc	**Ctrl+U**	Clear command line when cursor is at the prompt. Otherwise, move cursor to the prompt.
Delete	**Ctrl+D**	Delete character after cursor.
Backspace	**Ctrl+H**	Delete character before cursor.
None	**Ctrl+K**	Cut contents (*k*ill) to end of command line.
Shift+Home	None	Select from cursor to beginning of statement.
Shift+End	None	Select from cursor to end of statement.

II. Scripts et fonctions

MATLAB est un langage de programmation puissant aussi bien qu'un environnement de calcul interactif. Les fichiers qui contiennent un programme dans le langage MATLAB ont l'extension **.m** (en anglais, on les appellent les **M-files**). On crée des M-files en utilisant un éditeur de texte. Matlab possède son propre éditeur de texte qu'on ouvre :

- 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 fichiers Matlab : les **scripts et les fonctions**.

Avant de continuer, il est à signaler que pour que Matlab puisse exécuter un script ou une fonction, il faut qu'il connaisse son **chemin**.

On vous demande alors de :

- créer un répertoire D:\ETI (ou dans votre clé USB)
- ajouter ce répertoire au path de Matlab 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. Vous pouvez aussi indiquer à Matlab le chemin de votre répertoire en le plaçant dans la fenêtre « Current directory », ou en tapant l'instruction :

```
>> cd D:\ETI
```

Remarques :

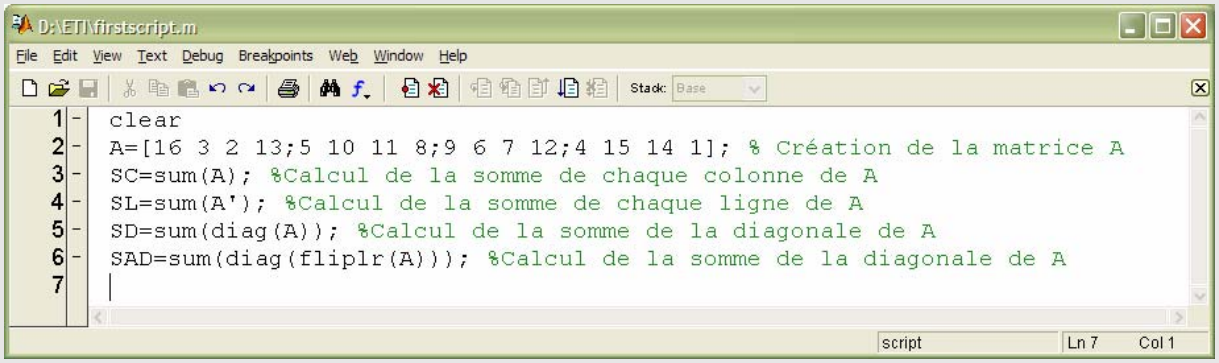
La commande **dir** retourne la liste des fichiers contenus dans le répertoire courant.

La commande **pwd** permet de connaître le nom du répertoire courant.

II.1 Scripts

Un script est un ensemble d'instructions MATLAB qui **joue le rôle de programme principal**. Si le script est écrit dans le fichier de nom **myscript.m** on l'exécute dans la fenêtre MATLAB en tapant son nom dans la fenêtre de commande (**>> myscript** et valider). **Même si Matlab est interactif** et permet de faire beaucoup de chose en tapant les instructions directement dans la fenêtre de commande, **il est toutefois utile d'utiliser un script**. 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 commande.

Exercice : Ouvrez l'éditeur de texte (ou de script) de Matlab. Reproduire alors le script ci-dessous.



Enregistrer le dans votre répertoire sous le nom « firstscript ». Exécuter le.

Afficher alors les résultats obtenus, c-à-d les valeurs des SC, SL, SD et SAD.

Remarque :

- Il est important de commencer un programme par l'instruction **clear**. Cette instruction effacera toutes les variables se trouvant dans l'espace. Ainsi, toutes les variables seront créées par le présent programme.
- 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, une ligne commentaire commence par « **%** ».

II.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
% Les commentaires se trouvant juste après la déclaration
% function et avant le corps de la fonction seront affichés quand
% on tapera la commande >> help mafonction.
```

help

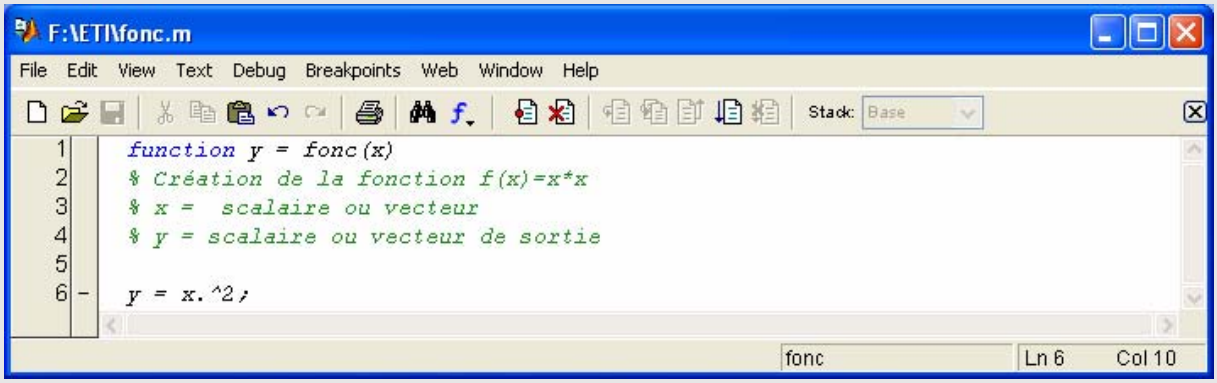
Corps de la fonction *contenant les instructions et les commandes*

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 que la fonction ayant pour nom **fonc** soit enregistrée dans un fichier du même nom **fonc.m** sans quoi cette fonction ne sera pas reconnue par MATLAB.

Exemple :

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



Enregistrer cette fonction. Tapez ensuite dans la fenêtre de commande :

```
» fonc(2)
```

On peut aussi placer le résultat dans une autre variable:

```
» b = fonc(5)
```

ou encore puisque l'on peut utiliser des vecteurs:

```
» v = [1 2 3]; r = fonc(v)
```

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

À la différence des fichiers SCRIPT, les variables à l'intérieur d'un fichier FUNCTION ne sont pas disponibles à l'extérieur. On dit qu'elles sont locales. Si on essaie d'obtenir la valeur des variables locales x et y :

```
» x
```

??? Undefined function or variable x.

» y

??? Undefined function or variable y.

Matlab ne les reconnaîtra pas.

On peut aussi utiliser notre fonction dans un script.

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%                   Fichier tracer.m               %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%  Partition de l'intervalle [0,5]

   x = 0 : 0.01 : 5.0 ;

%  Evaluation de la fonction sur tout le vecteur x

   y = fonc(x);

%  Graphique de la fonction

   plot(x,y, '-')
```

Exercice : Transformer le script « firstscript » en une fonction nommée « statmagic » recevant comme argument un nombre entier x, crée un carré magic de dimension x et retourne come argument de sortie SC, SL, SD et SAD.

Remarques :

a) Fonction comme argument d'une fonction :

Mentionnons l'existence de commandes permettant d'évaluer une chaîne de caractères comme une commande Matlab. La commande **eval** s'utilise ainsi :

```
>> comd='s1=sin(pi/2)'

>> eval(comd)
```

b) Arguments d'entrée et de sortie :

Il n'est pas obligatoire de fournir tous les arguments d'entrée et de sortie lors de l'appel d'une fonction, mais ceux que l'on omet doivent être les derniers des listes d'entrée ou de sortie. Ainsi, supposons que *mafonction* soit une fonction à 2 arguments d'entrée et 2 arguments de sortie, on peut alors effectuer l'appel suivant :

[*vars1***]=** mafonction**(***vare1***)** mais pas l'appel : **[***vars2***]=** nomfonction**(***vare2***)**.

Bien entendu, il faut gérer l'omission des arguments d'entrée. Matlab met à votre disposition la variable **nargin** qui indique le nombre d'arguments en entrée lors de l'appel de la fonction. Voici un exemple de calcul d'un produit scalaire ou d'une norme au carrée, illustrant son emploi :

```
function r = psnorm2(a,b) (a,b sont des vecteurs colonnes)

if (nargin==1)

r=a'*a;

elseif (nargin==2)

r=a'*b;

end
```

psnorm2(u,v) renvoie le produit scalaire et *psnorm2(u)* renvoie la norme au carrée de *u*. Remarquer que *psnorm2(u,u)* renvoie également la norme au carré de *u*.

Bien qu'il soit parfois dangereux d'utiliser cette possibilité, il est important de la connaître car de nombreuses fonctions natives de Matlab en font usage.

c) Obtenir la description d'une fonction

La commande **help** permet d'obtenir de l'aide sur une fonction. Par exemple pour obtenir l'aide sur la fonction sum (somme) tapez :

```
>> help sum
```

Quand on ne connaît pas le nom de la fonction à utiliser, la commande **lookfor** permet de faire une recherche par mot clé. Par exemple :

```
>> lookfor Fourier
```

affiche le nom de toutes les fonctions qui sont en rapport avec le mot « Fourier ». Le mot après lookfor doit être en anglais.

d) Editer un script ou une fonction

La commande **type** permet d'afficher le contenu d'un M-file dans la fenêtre de travail. Par exemple, tapez:

```
>> type firstscript
```

La commande **edit** permet d'afficher le contenu d'un M-file dans l'éditeur de texte Matlab. Par exemple, tapez:

```
>> edit firstscript
```

- La commande **delete** permet de détruire fichier et objet graphique.

II.3 Boucles & Contrôle de flux

II.3.1 Boucles de contrôle

II.3.1.1 BOUCLE FOR

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

On peut aussi réaliser des boucles FOR imbriquées.

Exemples :

- **Boucle FOR simple :**

```
for i=1:100

   wt = 24*i*0.01;

   x(i)=12.5*cos(wt+pi/6);

end
```

- **Deux boucles 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
```

II.3.1.2 BOUCLE WHILE

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

Syntaxe :

```
while condition

  opérations à exécuter si la condition est vraie

end
```

Exemple :

```
n=1;

while n<100

   x=n*0.05;

   y(n)=5.75*cos(x);

   z(n)=-3.4*sin(x);

   n=n+1;

end
```

II.3.2 Contrôle conditionnel : if, else, elseif, switch

II.3.2.1 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 condition logique 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**).

Exemple :

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

II.3.2.2 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;

...

case valeurn, instructionn;

end

Exemple :

```
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
```

II.3.3 BREAK et CONTINUE

Les déclarations "**break**" et "**continue**" permettent un contrôle plus stricte 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.

II.3.4 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.

II.4 LES POLYNOMES (voir annexe 4)

Matlab représente un polynôme comme un **vecteur ligne** qui contient les coefficients des termes rangés selon les puissances décroissantes.

```
>> p = [ 1 -12 0 5 0] % représente x^4 -12x^3 + 5x.
```

II.4.1 Racines d'un polynôme.

```
>> r = roots(p) % fournit les racines du polynôme donné ci-dessus.
>> p = poly(r) % est l'opération inverse qui redonne les coefficients rangés
                %selon les puissances décroissantes.
```

II.4.2 Addition.

On additionne les polynômes a et b en écrivant **d = a+b** s'ils sont de même degré. Sinon il faut compléter celui qui a le degré le plus faible avec des zéros.

II.4.3 Multiplication.

La fonction **conv(a,b)** réalise le produit des polynômes a et b. On renvoie le résultat dans c en écrivant `c = conv(a,b)`. Ce produit est obtenu par **conv**olution des vecteurs a et b.

II.4.4 Division.

La fonction **deconv( )** permet la division.

```
>> [q,r] = deconv(c,b) % divise c par b. q est le quotient et r le reste.
```

II.4.5 Dérivation.

La fonction **polyder( )** fournit la dérivée d'un polynôme.

```
>> h = polyder(g) % donne h dérivée de g.
```

II.4.6 Evaluation.

La valeur prise par le polynôme pour une valeur de la variable est fournie par la fonction **polyval( )**.

II.5 Intégration.

Les fonctions **trapz**, **quad** et **quad8** calculent l'aire comprise entre la courbe et l'axe horizontal.

Exemple : Définir la fonction g1.m :

```
function y=g1(x);
y=x.*x;
```

puis écrire le programme:

```
clear all
x=0:0.1:1;
y=g1(x);
aire1=trapz(x,y)
aire2=quad('g1',0,1)
```

II.6 Dérivation.

On évitera autant que possible la fonction **diff** qui dérive en faisant la différence entre éléments voisins dans le tableau des valeurs expérimentales, toujours faussées par le bruit.

Il est plus astucieux de remplacer la courbe par une bonne approximation polynomiale avec **polyfit** et de dériver celle-ci avec **polyder**.

II.7 Equations différentielles.

De nombreux systèmes physiques sont décrits par des équations différentielles, linéaires ou non. Pour intégrer une équation différentielle d'ordre n, on se ramène à un système de n équations du premier ordre, ce qui permettra d'utiliser les fonctions **ode23**, **ode45** **...** (ordinary differential equations) qui fourniront la solution cherchée.

III. Graphiques

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

Des outils interactifs vous permettent de manipuler les graphiques et 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.

III.1 Graphiques 2D

III.1.1 Fonction `plot`

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

La commande **`plot(x,y)`** permet de tracer le vecteur y en fonction du vecteur x. En toute rigueur, il faut définir auparavant les vecteurs x et y qui doivent être de même dimension.

Exercice *: Ecrire et commenter les lignes du script suivant :*

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

L'enregistrer et l'exécuter.

Au graphe obtenu, on peut ajouter la légende des axes en exécutant les commandes (ou en l'ajoutant dans le script) :

```
>> xlabel('abscisse');

>> ylabel('ordonnee');
```

On peut aussi donner un titre à la courbe à l'aide de la commande :

```
>> title ('Exemple de graphique');
```

Et on peut aussi ajouter un quadrillage :

```
>> grid on
```

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

```
>> grid off
```

Dans certains cas, on est amené à **tracer plusieurs courbes** dans une même figure. Dans MATLAB, ceci peut se faire des **deux manières** suivantes. La première est d'**utiliser une seule instruction plot**. Modifier le script précédent en introduisant les instructions suivantes :

```
y1 = exp(-2*x).*sin(x);
y2 = exp(-3*x).*sin(x);
y3 = exp(-4*x).*sin(x);
plot(x,y,'-',x,y1,'-.',x,y2,'*',x,y3,'o');
```

Remarque : On peut ajouter une légende à cette courbe, en utilisant la commande suivante :

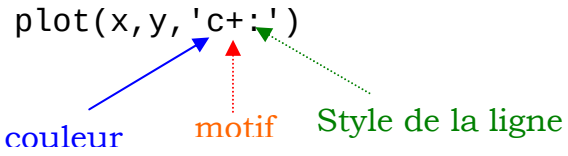
```
>> legend('courbe y', 'courbe y1', 'courbe y2', 'courbe y3')
```

La deuxième manière de tracer plusieurs courbes dans une même figure consiste à **tracer les figures une par une**. Mais dans ce cas un problème se pose : 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 va utiliser cette même fenêtre et va effacer l'ancienne courbe pour tracer la récente. Pour grader l'ancienne courbe et tracer dessus la nouvelle courbe, il faut utiliser la commande MATLAB **hold on** (l'effet de cette commande est

annulé à l'aide de la commande **`hold off`**). Ainsi, on peut obtenir la même figure qu'auparavant en utilisant les commandes (vous pouvez réenregistrer le script précédent sous un autre nom et remplacer les instructions précédentes par celles qui suivent) :

```
>> plot(x,y,'-') ;
>> hold on
>> plot(x,y1,'-.')
>> plot(x,y2,'*')
>> plot(x,y3,'o');
>> hold off
```

Mais ici, vous remarquez que **toutes les courbes sont tracées en bleu** (couleur choisie par défaut pour le traçage des courbes). Pour utiliser des couleurs différentes 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	-	Ligne solide
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)		

Signalons aussi que l'on peut **tracer les courbes sur des fenêtres graphiques séparées** à condition de créer ces fenêtres à l'aide de la commande figure :

```
>> figure(1), plot(x,y), title('y');

>> figure(2), plot(x,y1), title('y1');

>> figure(3), plot(x,y2), title('y2');

>> figure(4), plot(x,y3), title('y3');
```

Si vous préférez **tracer plusieurs courbes, dans la même fenêtre,** mais dans des cadres séparés, alors on utilise la fonction MATLAB **subplot** :

```
>> subplot(2,2,1), plot(x,y), title('y');

>> subplot(2,2,2), plot(x,y1), title('y1');

>> subplot(2,2,3), plot(x,y2), title('y2');

>> subplot(2,2,4), plot(x,y3), title('y3');
```

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. Ceci peut se réaliser en utilisant la commande **axis** dont la syntaxe est :

axis([Xmin Xmax Ymin Ymax])

où Xmin, Xmax : les graduations minimales et maximales de l'axe des x

Ymin, Ymax : les graduations minimales et maximales de l'axe des y

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

axis('square') présente le graphe dans un carré au lieu du rectangle habituel.

On peut aussi utiliser la fonction **fplot**:

```
>> g = '2*exp(-x).*sin(x)';
>> fplot(g,[0 8])
```

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^{ème} 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)**

II.1.2 Autres fonctions de représentation 2D

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

- **Fonction polar :** permet de **représenter des fonctions en coordonnées polaires.** Ceci présente un grand intérêt pour les diagrammes de directivité. Par exemple, ici un diagramme de directivité de type cardioïde.

```
>> 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**. Essayer les commandes

```
>> stem (x,y)
```

et

```
>> stem (theta,r)
```

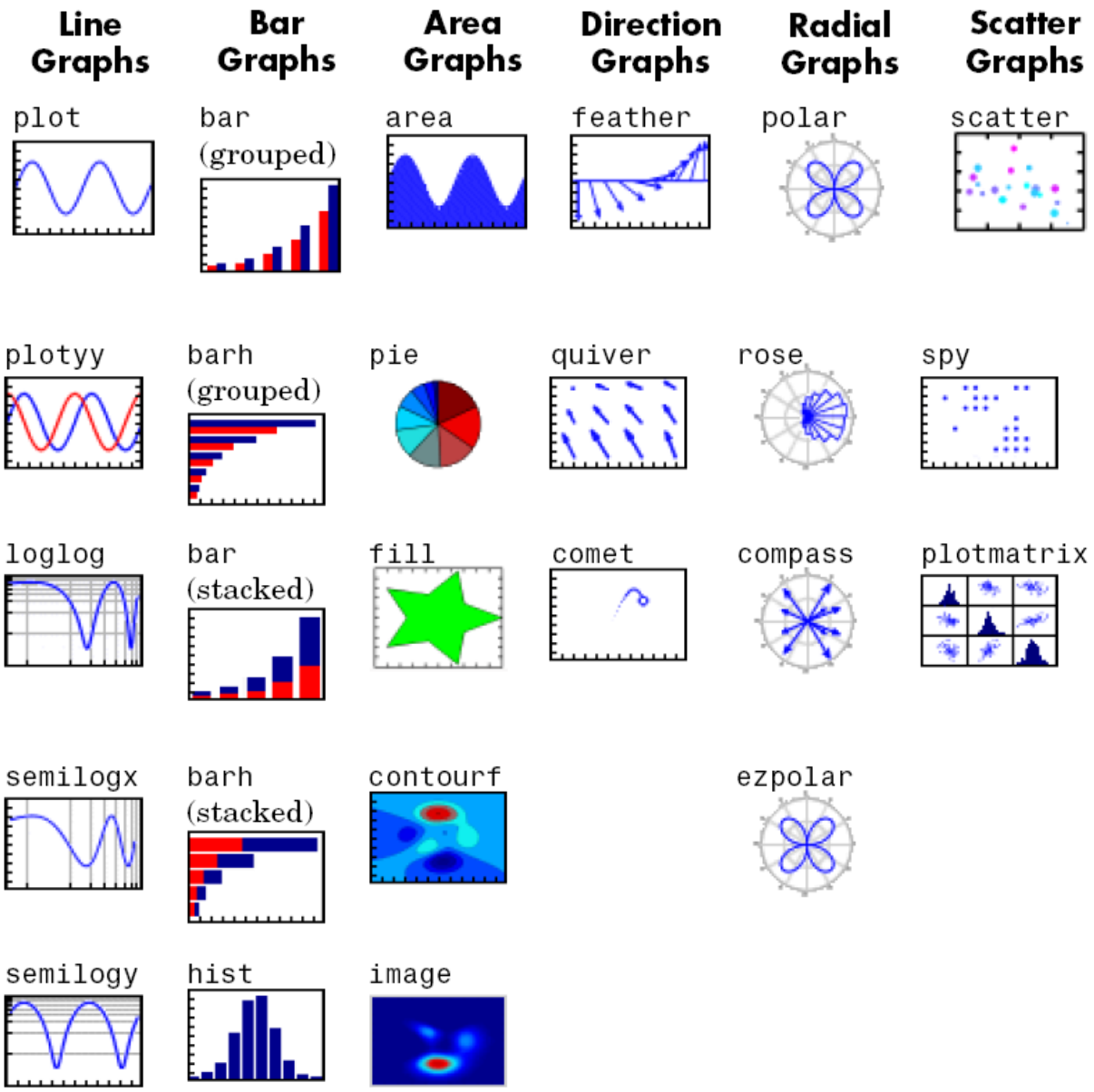
- **Fonction hist : hist(x,N) présente un histogramme de N barres** de la variable x. Par défaut, N=10. Essayer les commandes suivantes :

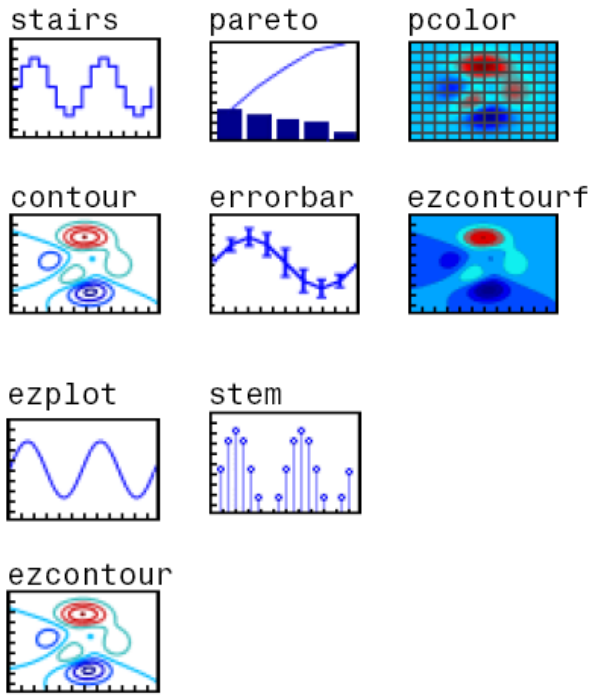
```
>> y=randn(50000,1);

>> hist(y)

>> hist(y,20)
```

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





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

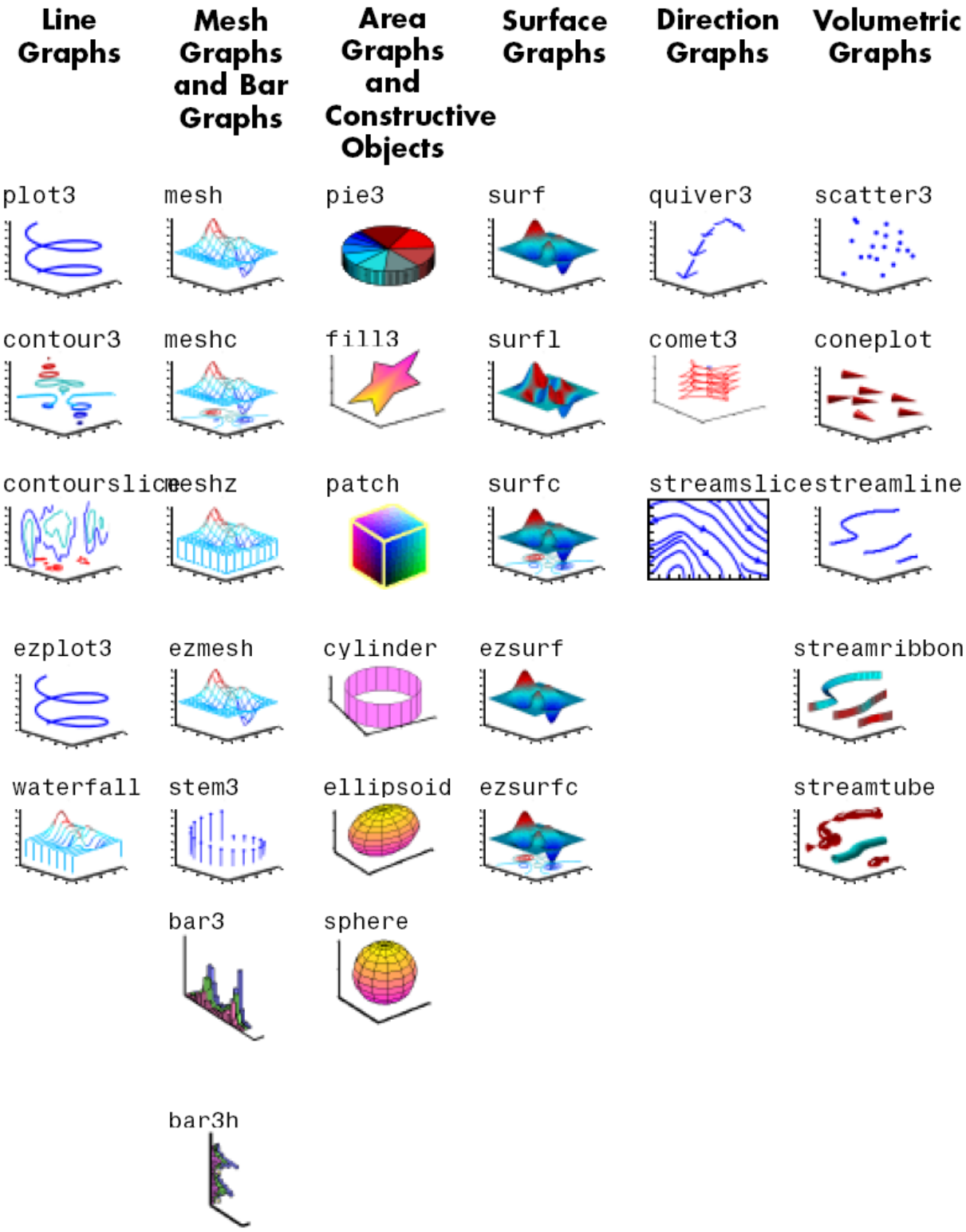
La commade **close all** permet de fermer toutes les fenêtres graphiques.
La commade **close(n)** permet de fermer la fenêtre graphique n.

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

```
>> help graph2d
```

II.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 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

>> shading interp % meilleure interpolation

>> colormap pink %choix d'une palette de couleurs prédéfinie

>> view(-37.5+60,30)% changement point de vision: view(azimut, elevation)
```

N.B :

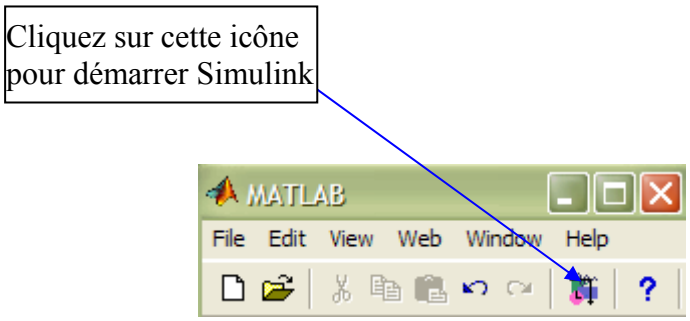
- view(0,90); % Met le graphe 3D "à plat", par défaut view(-37.5,30).
- colorbar % Ajoute l'échelle des couleurs
- axis equal; % Échelles isométriques
- axis tight; % Ajuste les limites des axes aux valeurs des données
- axis([xmin xmax ymin ymax zmin zmax]) % Impose les limites du tracé en x,y, et z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

IV. Introduction à Simulink

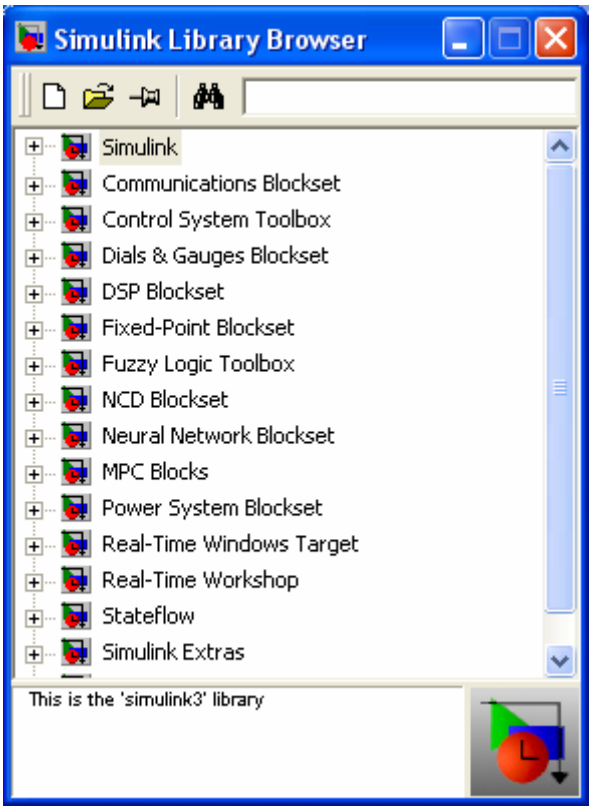
Simulink est l'extension graphique de MATLAB permettant de représenter les fonctions mathématiques et les systèmes sous forme de diagramme en blocs, et de simuler le fonctionnement de ces systèmes.

IV.1 Lancement de Simulink

Dans la fenêtre de commande de Matlab:



Vous verrez alors apparaître la fenêtre suivante :



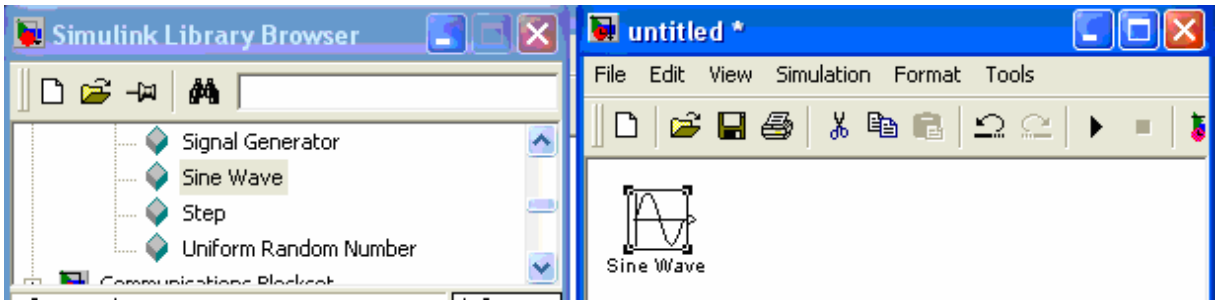
Remarque : L'apparence des fenêtres varie selon la version du logiciel Matlab utilisée.

Pour ouvrir une nouvelle fenêtre de travail Simulink, cliquez sur l'icône  (de la dernière fenêtre ci-dessus) ou ouvrez File→New→Model ou utilisez le raccourci Ctrl+N.

IV.2 Construction d'un modèle dans la fenêtre de travail

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 de travail, le composant « Scope » se trouvant dans la sous-librairie « Simulink\Sinks ».

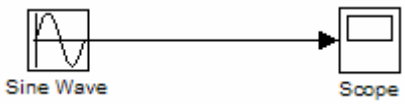
IV.2.2 Réalisation des connexions

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

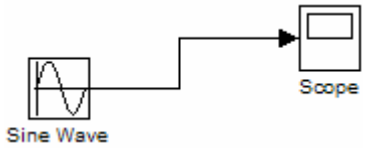
On sélectionne avec la souris, le symbole **>** situé sur un composant, on maintient l'appui sur le bouton et on tire le lien vers le symbole **>** de l'autre composant.

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 :

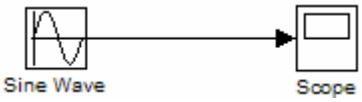


Si les deux composants ne sont pas alignés, la connexion entre eux n'est pas une droite. Ceci se fait automatiquement, toutefois on peut le réaliser nous même car il suffit de relâcher le bouton de la sourie, de maintenir à nouveau appuyé tout en se déplaçant dans la nouvelle direction :



Remarque :

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



Connexion bien établie

Connexion 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.

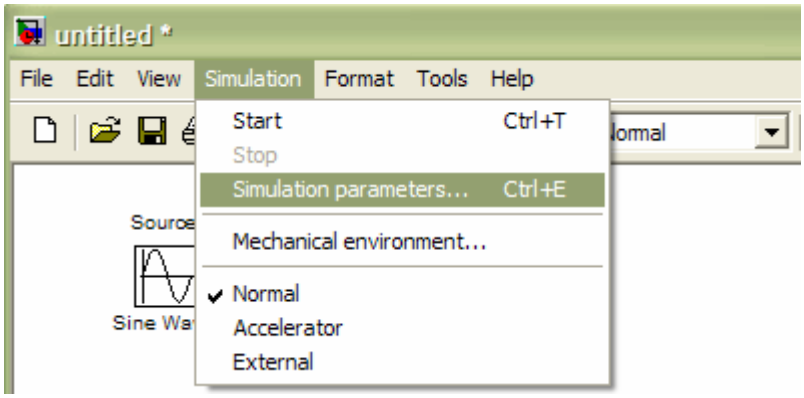
IV.2.3 Paramétrage des composants

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.



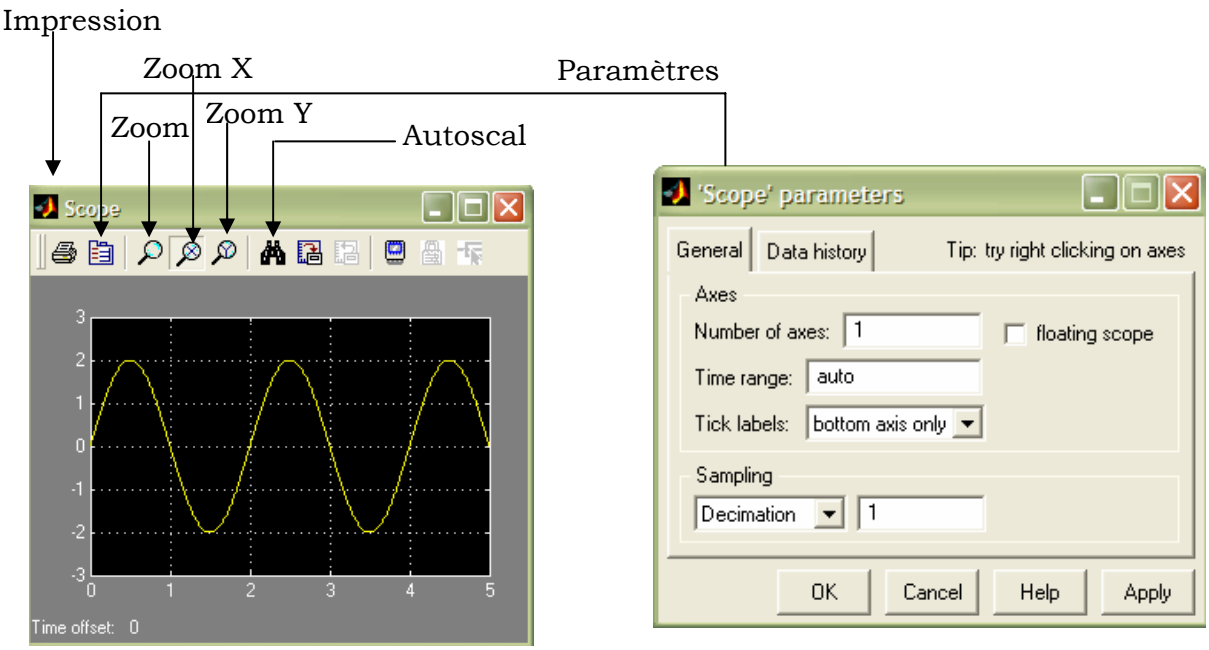
IV.2.4.2 Lancement de la simulation

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 :



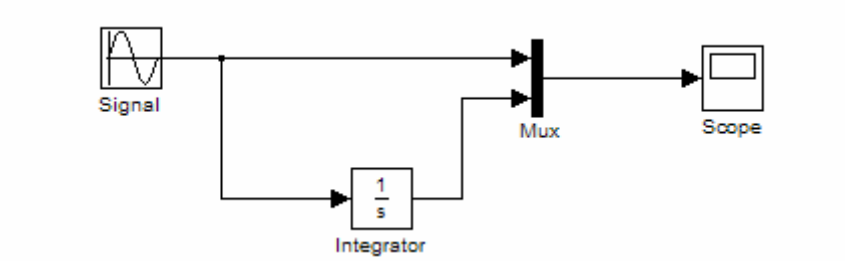
L'icône « Autoscal » permet de marier la courbe à la fenêtre de l'écran. Les icônes du Zoom permettent ensuite de zoomer soit sur les deux axes ou sur chacun d'entre eux.

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.

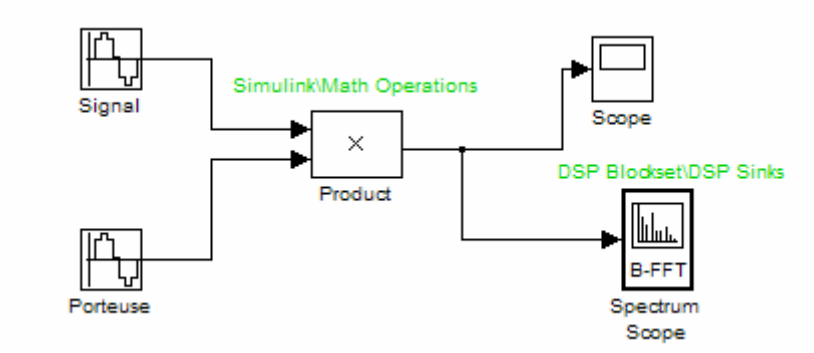
Dans la fenêtre « Simulation parameters » laisser l'instant de départ « Start time » à sa valeur par défaut (0) et régler l'instant d'arrêt « Stop time » à 5s puis valider en cliquant sur Ok. Simuler et consulter le Scope.

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 :

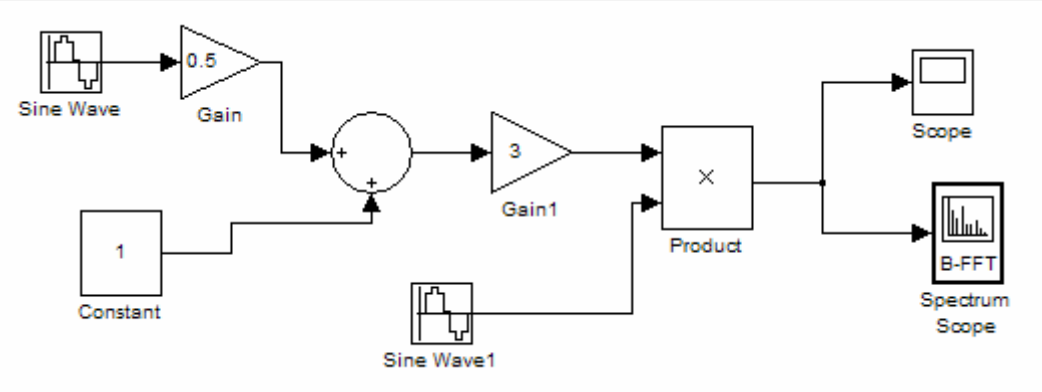


Les paramètres des différents composants sont les suivants :

- Source Signal : Amplitude=1 ; Fréquence=10rad/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.

Exercice : Réaliser le modèle Simulink suivant :



Garder tous les paramètres de l'exercice précédent. Simuler.

Annexe 1 — Les Matrices

Opérations sur les matrices

```
>> A'  % Transposée conjuguée de A
>> A.'  % Transposée de A
>> 2*A  % Produit par un scalaire
>> A*B  % Produit de deux matrices (de dimensions % cohérentes)
>> A^p  % Elève la matrice carrée A à la puissance p
>> inv(A)  % Inversion d'une matrice carrée inversible  (message d'alerte éventuel)
>> A.*B  % Produit élément par élément de deux matrices
```

Attention : `A*A` est différent de `A.*A`.

```
>> X = A\B  % Donne la solution de A*X = B
>> X = B/A  % Donne la solution de X*A = B
>> X = A./B  % Division éléments par éléments
>> inv(A)  % Inversion d'une matrice carrée inversible (message d'alerte éventuel)
>> poly(A)  % Polynôme caractéristique de A
>> det(A)  % Déterminant de A
>> flipud(A)  % Retournement vertical de A
>> fliplr(A)  % Retournement horizontal de A
>> rot90(A)  % Rotation de A de 90 degrés
>> B = reshape(B,L,C)  % Remise en forme de B avec L lignes et C colonnes
```

Exemple : `>> B = [ 1 2 3 4 5 6 ]; B = reshape(B,2,3)`

```
B = 1 3 5
    2 4 6
```

```
>> [V, D] = eig(A)  % Renvoie les valeurs propres et les vecteurs propres de A
                    % (eigenvalues, eigenvectors)
```

Construction de matrices particulières

toeplitz permet de construire des matrices de Toeplitz :

```
>> c = [ 1 2 3 4 5 ];
>> l = [ 1.5 2.5 3.5 4.5 5.5 ];
>> toeplitz(c,l)
```

vander permet de construire des matrices de Vandermonde :

```
>> x = 2:5;
>> vander(x)
```

Décomposition de matrices

```
>> [U, S, V] = svd(X); % décomposition en valeurs singulières
```

☞ Elle produit une matrice diagonale *S* de même dimension que *X*, avec les éléments diagonaux non nuls rangés en ordre décroissant, et des matrices unitaires *U* et *V* telles que *X* = *U**S***V*'.

```
>> R = chol(X); % réalisé la factorisation de Cholesky
```

☞ Elle utilise uniquement la partie triangulaire supérieure de *X* (et considère que la partie triangulaire inférieure de *X* est la transposée conjuguée de la précédente). Elle produit un message d'erreur si *X* n'est pas définie positive, sinon elle produit la matrice triangulaire supérieure *R* telle que *R'*R=X*.

```
>> [Q,R] = qr(X); % réalise la factorisation QR
```

☞ produit une matrice triangulaire supérieure *Q* de même dimension que *X* et une matrice unitaire *R* telle que *X = Q*R*.

```
>> [L, U] = lu(X); % réalise une factorisation LU (Lower, Upper)
```

☞ exprime une matrice carrée *X* sous forme d'une matrice triangulaire supérieure *U* et d'une permutation d'une matrice triangulaire inférieure *L*, telles que *X = L*U*.

Matrices particulières

Fonctions	Définition
zeros(n)	Génère une matrice de zéros de grandeur nxn
zeros(n,m)	Génère une matrice de zéros de grandeur nxm
ones(n)	Génère une matrice de "1" de grandeur nxn
ones(n,m)	Génère une matrice de "1" de grandeur nxm
eye(n)	Génère une matrice identité de grandeur nxn
eye(n,m)	Génère une matrice identité de grandeur nxm
length(X)	Retourne la longueur du vecteur X
size(X)	Retourne les dimensions (lignes, colonnes) du vecteur X

Matrices creuses (help sparfun)

Fonctions	Définition
sparse	création d'une matrice creuse
full	conversion d'une matrice creuse
issparse	retourne 1 si la matrice est creuse
speye	matrice identité creuse

Informations sur les matrices

Fonctions	Définition
det	déterminant
norm	normes 1, 2 etc. de la matrice
trace	trace de la matrice
cond	conditionnement

rank	rang
triu	partie triangulaire supérieure
tril	partie triangulaire inférieure
sort	arrangement par ordre croissant

Annexe 2 Signaux Aléatoires

Génération aléatoire

La génération de processus aléatoires est possible sous *Matlab* avec les deux fonctions qui suivent :

Fonctions	Définition
rand	Génération de valeurs aléatoires : **distribution uniforme**
randn	Génération de valeurs aléatoires : **distribution gaussienne**

Analyse des données

Fonctions	Définition
median	valeur médiane
max	maximum
min	minimum
std	écart type (standard deviation)
mean	moyenne
corrcoef	coefficient de corrélation
prod	produit des éléments
sum	somme des éléments

Annexe 3 Mathématiques, géométrie

Fonctions	Définition
abs(x)	valeur absolue ou module d'un nombre complexe
acosh(x)	fonction inverse du cosinus hyperbolique
asin(x)	fonction inverse du sinus
asinh(x)	fonction inverse du sinus hyperbolique
atan(x)	fonction inverse de la tangente
atanh(x)	fonction inverse de la tangent hyperbolique
conj(x)	complexe conjugué
cos(x)	cosinus
cosh(x)	cosinus hyperbolique
exp(x)	fonction exponentielle
floor(x)	partie entière dans le cas d'un réel positif
imag(x)	partie imaginaire
log(x)	fonction logarithme népérien
log10(x)	fonction logarithme en base 10
real(x)	partie réelle
sign(x)	fonction signe
sin(x)	sinus
sinh(x)	sinus hyperbolique
sqrt(x)	racine carrée
tan(x)	tangente
tanh(x)	tangente hyperbolique

Les fonctions d'arrondis

Fonctions	Définition
round(x)	entier le plus proche de x
ceil(x)	arrondi par excès
floor(x)	arrondi par défaut
fix(x)	arrondi par défaut un réel positif et par excès un réel négatif

Valeurs spéciales prédéterminées dans Matlab

Fonctions	Définition
Inf	Ceci représente ∞ causéepar une division par zéro
NaN	"Not-a-Number", causé par une opération mathématique indéterminée. Par exemple, "0/0"
clock	Génère la date et le temps dans la forme d'un vecteur-ligne de 6 éléments correspondant à: année, mois, jour, heure, minute et secondes.
ans	Variable générée par Matlab pour sauvegarder un résultat si l'utilisateur n'en a pas créé une dans le programme

Précision des résultats affichés

Format	Résultat	Exemple
format short	4 chiffres après la virgule (par défaut)	1.1234
format long	14 chiffres après la virgule	1.12345678910111
format short e	4 chiffres après virgule + exposant	1.1234e+001
format short g	5 chiffres en tout avec ou sans exposant	12.126
format long e	15 chiffres après virgule + exposant	1.233333333333333e+043
format long g	15 chiffres après virgule au total, avec ou sans exposant	1.23333333333333e+043
format bank	"dollars et sous" format	9.75
format hex	affiche les bits en format hexadécimal	4028b0fcd32f707a
format +	seulement les signes sont affichés	+

Annexe 4 Fonctions de filtrage et de convolution

```
>> Y = filter(B,A,X);
```

☞ Elle construit le vecteur Y tel que :

$$y(n) = b(1)*x(n) + b(2)*x(n-1) + ... + b(nb+1)*x(n-nb) - a(2)*y(n-1) - ... - a(na+1)*y(n-na)$$

Cela correspond donc à un filtrage par un filtre ARMA.

- La fonction ***roots*** qui calcule les zeros d'un polynôme, peut lui être associée afin de trouver les zéros et les pôles d'un ARMA ; Par exemple le polynôme $P(z)=z^3-6z^2-72z-27$ est représenté par le vecteur :

```
>> p=[1 -6 -72 -27];
```

dont les racines sont données par :

```
>> r=roots(p)

r = 12.1229 -5.7345 -0.3884
```

- Pour calculer la valeur du polynôme en certains points on utilise la fonction ***polyval*** :

```
>> polyval(p,[1,exp(j*pi/4),exp(j*pi/2)])

ans = 1.0e+002 *
     -1.0400 -0.7862 - 0.5620i -0.2100 - 0.7300i
```

- La fonction ***poly*** effectue l'opération inverse de *roots* : étant donné un polynôme unitaire défini par ses racines, *poly* le développe en donnant les coefficient de ses monômes en puissance décroissante.

```
>> p=poly(r)

p = 1.000 -6.000 -72.000 -27.000
```

☞ D'autre part, si l'argument de *poly* est une matrice, la fonction renvoie les coefficients du polynôme caractéristique de cette matrice.

- La fonction ***conv*** effectue elle la convolution de deux vecteurs :

```
>> Z = conv(X,Y);
```

☞ Cette convolution est complète dans le sens où la longueur de Z est la somme des longueurs de X et Y moins 1. Elle peut aussi être interprétée comme le développement du produit de deux polynômes.

- En dimension 2 les fonctions correspondant à *filter* et *conv* sont ***filter2*** et ***conv2***.

- Une fonction fournie mais dont l'utilisation doit vous sembler délicate est la fonction ***deconv***. Elle est sensée effectuer une déconvolution, ce qui peut sembler ambitieux (pour ceux qui ont suivi un cours sur les problèmes inverses). Néanmoins cette fonction est utile pour la division de polynômes.

Opérations sur les Polynômes

Fonctions	Définition
roots	Calcul des racines connaissant les coefficients
poly	Calcul des coefficients connaissant les racines
polyval	Calcul de P(x)
polyfit	Identification des coefficients d'un polynôme à l'aide des moindres carrés
conv	Produit de deux polynômes
deconv	Division de deux polynômes

Fonctions pour les nombres complexes

Fonctions	Description
conj(x)	Calcule le complexe conjugué de x. Si x=a+b*i alors conj(x)=a-b*i
real(x)	Retourne la partie réelle de x
imag(x)	Retourne la partie imaginaire de x
isreal(x)	Retourne vrai(1) si aucun élément du vecteur "x" n'a une partie imaginaire.
abs(x)	Retourne la magnitude de x |x|= a b2 2+
angle(x)	Retourne l'angle du nombre complexe "x" calculé par l'expression atan2(imag(x),real(x))

Fonctions pour Séries et Transformées de Fourier

Fonctions	Description	Dans Matlab
fft	"Fast Fourier Transform", (similaire à la série de Fourier en temps discret, aussi utilisée pour évaluer la transformée de Fourier en temps discret)	X = fft(x)
ifft	"Inverse Fast Fourier Transform"	x = ifft(X);

fftshift	Déplace les composantes à fréquence nulle au milieu du spectre. S'il s'agit d'un vecteur, la fonction interchange la moitié droite avec la moitié gauche de X. Pour les matrices, la fonction interchange le 1er et le 3 ème quadrants avec le 2 ème et 4 ème.	fftshift(X)

Fonctions d'optimisation

Fonctions d'optimisation	Description
fsolve	Résolution d'un système d'équations non-linéaires
fzero	Trouve les zéros d'une fonction à une variable
fmin	Minimisation d'une fonction à une variable
fmins	Minimisation d'une fonction à plusieurs variables

Analyse de système à temps discret

Commandes	Description	Matlab
conv	Convolution de deux signaux en temps discret x[n]*h[n]	y=conv(x,h)
dstep	"Step response", réponse à l'échelon pour un système en temps discret, pour N points	dstep(num,den,N)
dimpulse	"Impulse Response", réponse impulsionnelle d'un système en temps discret, pour N points	dimpulse(num,den,N)
filter	Filtre digital décrit par les vecteurs A (den) et B (num). Filtre les données dans le vecteur X .	Y=filter(B,A,X)
freqz	Retourne la réponse en fréquence aux fréquences "w"	[H,omega]=freqz(num,den,w)

Annexe 5 GRAPHISME

Affichage des courbes 2D

Fonctions élémentaires	Définition
plot	Affichage linéaire
loglog	Echelle log-log
semilogx	Echelle semilog sur x
semilogy	Echelle semilog sur y
line	Définition d'une ligne

Outils	Définition
polar	Affi. en coordonnées polaires
bar	Affi. en mode escalier
hist	Affi. d'un histogramme
fplot	Affi. d'une fonction donnée

Outils	Définition
title	Création d'un titre
xlabel	Commentaire sur x
ylabel	Commentaire sur y
grid	Création d'une grille
text	Commentaire sur graphe
axis	Gestion des axes (zoom)
subplot	Multi-graphe sur même figure
hold	Mode surimpression

Couleur		Marqueurs		Styles de lignes	
y	jaune	.	point	-	Ligne solide
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)		

Affichage des courbes 3D

Visualisation 2D des données 3D	Définition
contour	Courbes de niveaux
image	Visualisation par niveau de gris
imagesc	Idem mais non normalisé
mesh	Déformation d'une grille 3D.
meshc	Combinaison de mesh et de contour.
meshz	Combinaison de mesh plus projection sur le plan z=0.
surf	Idem que mesh, mais avec surface colorée.
plot3	Visualisation de type plot, mais avec 3 vecteurs

Outils	définition
colormap	Changement de palette de couleurs
view	Orientation de la vue 3D
meshgrid	Définition des matrices représentant la grille

Graphes spécialisés

Fonctions	Définition
bar	Barres verticales
barh	Barres horizontales
bar3	Barres verticales 3d
bar3h	Barres horizontales 3d
area	Surfaces

Image

Fonctions	Définition
image	Affichage d'une image
imagesc	Affichage d'une image avec son intensité
imread	Lecture d'un fichier image à un format donné
imwrite	Ecriture d'un fichier image à un format donné
iminfo	lecture des informations d'un fichier image