

*Télécommunications*

*1ème Année*



# **Formation Matlab**



# *Formation Matlab*



## TABLE DES MATIERES

<b>PARTIE N° 1</b>	<b>MATLAB : COMMANDES DE BASE ET INTRODUCTION AUX CALCULS</b>	<b>5</b>
<b>MATRICIELS</b>	.....	<b>5</b>
1.1	DESCRIPTION DE LA FENETRE DE COMMANDE DE MATLAB.....	5
1.2	EXPLORATION DES BIBLIOTHEQUES .....	7
1.2.1	<i>Matlab et ses bibliothèques</i> .....	7
1.2.2	<i>Commande help</i> .....	8
1.2.3	<i>Commande lookfor</i> .....	8
1.2.4	<i>Localisation de la fonction utilisée</i> .....	9
1.2.5	<i>Autres commandes</i> .....	9
1.3	COMMANDES DE GESTION DE L'ESPACE DE TRAVAIL .....	9
1.3.1	<i>Introduction</i> .....	9
1.3.2	<i>Matlab comme simple calculateur</i> .....	10
1.3.3	<i>Nombre complexe</i> .....	11
1.3.4	<i>Mathématiques, géométrie</i> .....	11
1.3.5	<i>Ecriture directe d'une matrice ou d'un vecteur</i> .....	11
1.3.6	<i>Gestion des variables dans l'espace de travail : commandes whos, clear, format</i> .....	12
1.4	OPERATIONS DIRECTES SUR LES MATRICES .....	13
1.4.1	<i>Opérations arithmétiques :</i> .....	13
1.4.2	<i>Opérations relationnelles et logiques</i> .....	13
1.5	MANIPULATIONS SUR LES ELEMENTS MATRICIELS .....	13
1.5.1	<i>Manipulation des indices</i> .....	13
1.5.2	<i>Valeurs particulières</i> .....	14
1.6	MATRICES CREUSES (HELP SPARFUN) .....	14
1.7	MATRICE ET CHAINE DE CARACTERES.....	14
1.8	BILAN ET EXERCICE D'APPLICATION .....	15
1.8.1	<i>Ce qu'il faut (au minimum) retenir !</i> .....	15
1.8.2	<i>Exercice d'application 1</i> .....	16
<b>PARTIE N° 2</b>	<b>SUR CERTAINES FONCTIONS DEIEES AU CALCUL.....</b>	<b>17</b>
2.1	INITIALISATION ET MANIPULATION .....	17
2.2	ALGEBRE LINEAIRE .....	18
2.2.1	<i>Fonctions élémentaires</i> .....	18
2.2.2	<i>Fonctions de décomposition</i> .....	18
2.3	OPERATIONS SUR LES DONNEES .....	19
2.3.1	<i>Génération aléatoire</i> .....	19
2.3.2	<i>Analyse des données</i> .....	19
2.4	OPERATIONS SUR LES POLYNOMES .....	19
<b>PARTIE N° 3</b>	<b>ELEMENTS DE PROGRAMMATION .....</b>	<b>21</b>
3.1	INTRODUCTION .....	21
3.2	SCRIPT.....	21
3.2.1	<i>Définition d'un script</i> .....	21
3.2.2	<i>Construction d'un script</i> .....	21
3.3	FONCTIONS.....	22
3.3.1	<i>Introduction</i> .....	22
3.3.2	<i>Définition</i> .....	22
3.3.3	<i>Exemple de fonction</i> .....	23
3.4	STRUCTURES DE PROGRAMMATION USUELLES.....	23
3.4.1	<i>Structure de type test</i> .....	24
3.5	BILAN ET EXERCICES D'APPLICATION : CREATION D'UNE FONCTION.....	26
<b>PARTIE N° 4</b>	<b>GRAPHISME .....</b>	<b>27</b>
4.1	PREAMBULE .....	27
4.2	NOTION DE FIGURE.....	27
4.3	AFFICHAGE DES COURBES 2D .....	28
4.3.1	<i>Commande plot</i> .....	28
4.3.2	<i>Commande fplot</i> .....	29

4.3.3	<i>Commande subplot</i> .....	29
4.3.4	<i>Commande axis</i> .....	30
4.4	AFFICHAGE DES COURBES 3D .....	30
4.4.1	<i>plot3 : arguments vectoriels</i> .....	30
4.4.2	<i>Fonctions à arguments matriciels</i> .....	31
4.4.3	<i>meshgrid</i> .....	31
4.4.4	<i>plot</i> .....	32
4.4.5	<i>contour et contour3</i> .....	33
4.4.6	<i>mesh et surf</i> .....	33
4.4.7	<i>image</i> .....	34
4.4.8	<i>colormap et colorbar</i> .....	34
4.4.9	<i>view</i> .....	35
4.5	GRAPHES SPECIALISES .....	36
4.5.1	<i>Barres et surfaces</i> .....	36
4.5.2	<i>Camemberts</i> .....	36
4.6	EXERCICE : GRAPHISME ET FONCTIONS .....	37
4.7	AUTRE EXERCICE .....	38
<b>PARTIE N° 5 FICHIERS DE DONNEES</b> .....		<b>39</b>
5.1	FICHIERS MATLAB .....	39
5.1.1	<i>Commande save</i> .....	39
5.1.2	<i>Commande load</i> .....	39
5.2	FICHIERS BINAIRES ET ASCII: LE FORMAT STANDARD .....	40
5.2.1	<i>Commande fopen</i> .....	40
5.2.2	<i>Commande fread</i> .....	41
5.2.3	<i>Commande fwrite</i> .....	42
5.2.4	<i>Commande fclose</i> .....	42
5.3	EXERCICE .....	42
<b>PARTIE N° 6 APPLICATIONS</b> .....		<b>43</b>
6.1	RAPPELS SUR LE DEVELOPPEMENT EN SERIE DE FOURIER ET TRANSFORMEE DE FOURIER .....	43
6.2	EXERCICE .....	45
6.2.1	<i>Décomposition en série de Fourier, représentation spectrale et somme de série</i> .....	45

## ***INTRODUCTION***

---

L'objectif de cette formation est de se familiariser au logiciel *Matlab*, applicatif dédié au calcul scientifique le plus utilisé dans les laboratoires universitaires et industriels tel que le CEA, Thomson, le CNES, etc.

A l'origine, la toute première version de *Matlab* fut mise en œuvre aux Universités du Nouveau Mexique et de Stanford, à la fin des années 70. Elle se destinait essentiellement à l'analyse numérique et à l'algèbre linéaire. Depuis, *Matlab* est devenu un outil que l'on peut utiliser pour mener des simulations dans de très nombreux domaines (traitement du signal, traitement d'image, automatique, statistique, aéronautique, etc.) et pour développer des applications avec une interface graphique.

Cette formation traite plus particulièrement la déclaration et la gestion des données (matrice, chaîne de caractères, etc.), la création d'un programme et la gestion d'un espace de développement, la création de fonctions et gestion des bibliothèques et des fichiers de données.

Ce logiciel sera en outre utilisé tout au long de votre formation à l'école et dans l'industrie, notamment sur des thèmes liés aux probabilités, processus aléatoires, filtrage et plus généralement le traitement du signal et des images ainsi que l'automatique.





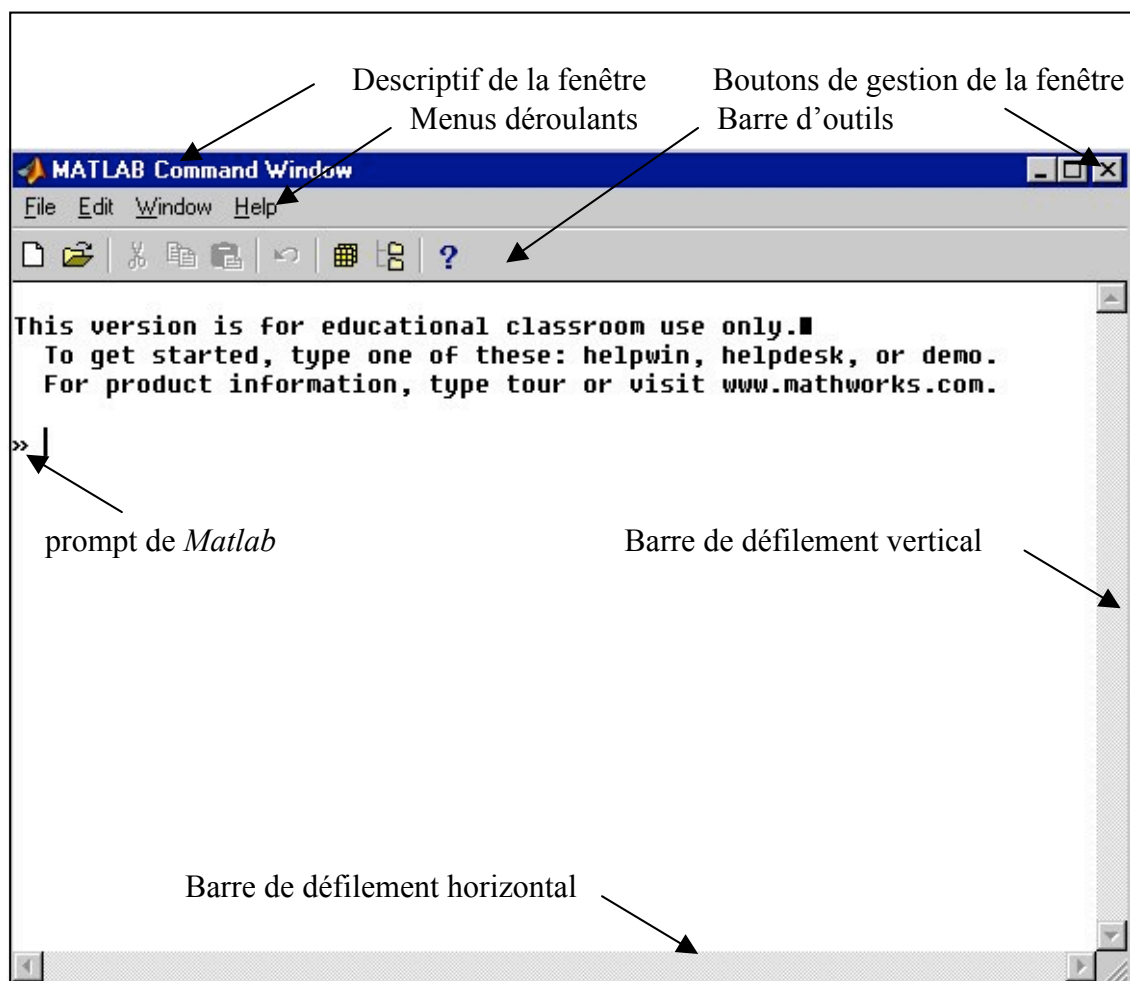
## ***PARTIE N° 1***

# ***MATLAB : COMMANDES DE BASE ET INTRODUCTION AUX CALCULS MATRICIELS***

---

### ***1.1 Description de la fenêtre de commande de Matlab***

Lancer *Matlab*, en utilisant le menu démarrer. Selon les versions du logiciel, on obtient l'un des deux fenêtres de commande suivantes. Le chargé de cette formation montrera éventuellement la mise en œuvre d'un raccourci. Avec la version précédente, on a :



A partir des versions 6 et 7, la fenêtre de commande ("*prompt*") est complétée par des fenêtres utilitaires permettant de visualiser les fichiers du répertoire courant ou les variables définies dans le *workspace* courant, ce qui correspond aux commandes *ls* et *whos*.

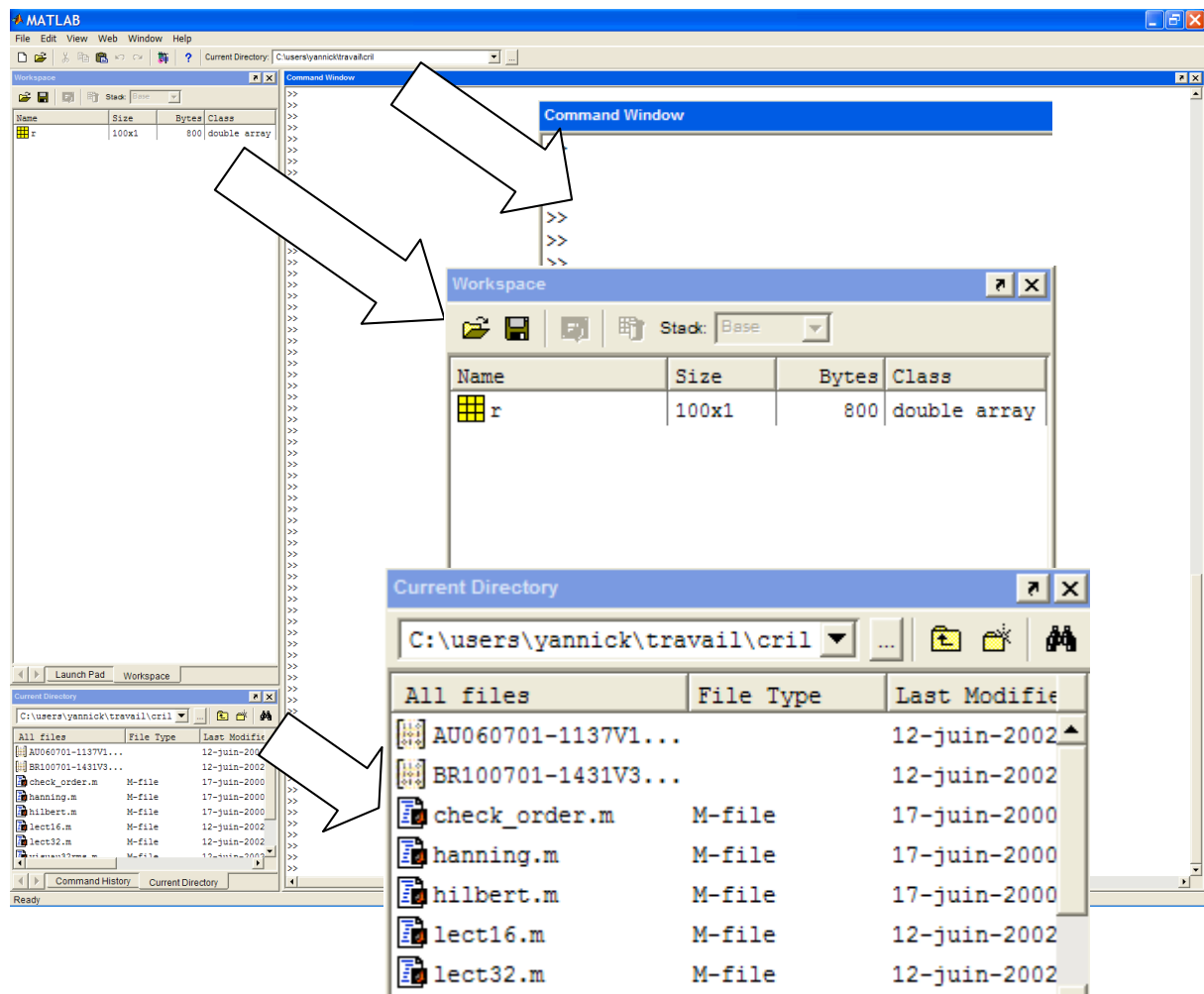


Figure n°1 : fenêtres Matlab

Il existe quatre menus déroulants :



- *File* : le menu fichier permet notamment d'accéder à des fichiers déjà existants ou de créer de nouvelles procédures et des nouveaux programmes. Il permet aussi de quitter *Matlab*. Les derniers programmes ouverts sont en outre indiqués.
- *Edit* : le menu édition permet d'accéder aux commandes classiques : copier, couper, coller, sélectionner tout, etc.
- *Window* : le menu Fenêtre donne accès aux différentes fenêtres de *Matlab* ouvertes.
- *Web* : spécifique à Matlab 6.1.
- *Help* : c'est le menu d'aide.

Le soulignement des lettres *F*, *E*, *W* et *H* des menus *File*, *Edit*, *Window*, *Help* indique le raccourci clavier pour ouvrir le menu déroulant sans l'aide de la souris. On effectue alors la commande *alt+lettre soulignée*.

Comme dans les logiciels *Word* et *Excel*, on peut aussi exploiter la barre d'outils. Cette dernière comprend plusieurs « boutons » qui permettent notamment de créer une nouvelle

fenêtre d'édition pour l'écriture de programmes, d'ouvrir un fichier déjà existant, de couper, copier ou coller une partie d'un programme, d'annuler la dernière action, etc.

Il est à noter qu'une information « bulle » apparaît lorsque la souris est positionnée suffisamment longtemps sur l'un des boutons ; elle renseigne alors sur son rôle.

On constate aussi que la fermeture  et la mise en icône  de la fenêtre de commande peut se faire par un simple clic sur les boutons de gestion de fenêtre, situés en haut à droite. La fenêtre peut aussi couvrir la totalité de l'écran, puis reprendre sa taille initiale, à l'aide du dernier bouton.

Le déplacement de la fenêtre s'effectue en maintenant le bouton d'action de la souris (en général celui de gauche) sur la barre descriptive de la fenêtre.

Enfin, la fenêtre dispose de barres de défilements vertical et horizontal. Comme pour *Word*, *Excel*, *Powerpoint*, etc., le « dimensionnement » de la fenêtre est possible. On se place alors à la limite de la fenêtre et en maintenant le bouton d'action enfoncé de la souris, on définit une nouvelle taille de la fenêtre.

En outre, le prompt de *Matlab* est défini par le sigle `>>`.

Application : On propose, en guise de préambule, de manipuler la démonstration pour connaître les possibilités qu'offre le logiciel *Matlab*. Pour accéder à la démonstration, il suffit de taper **demo** dans la fenêtre de commande, ce qui fait apparaître des fenêtres interactives donnant un panorama des différents outils proposés par le logiciel. On peut en général y lire les principales lignes de programmes.

## 1.2 Exploration des bibliothèques

### 1.2.1 *Matlab* et ses bibliothèques

*Matlab* est un logiciel qui dispose d'une collection de fonctions/programmes, que l'on désigne aussi par des fichiers à extension **.m**. Ces **.m** sont classés par thèmes et constituent alors des boîtes à outils (ou *toolboxes* en anglais). On dispose ainsi des *toolboxes* 'general', 'identification', 'signal', etc. *Matlab* ayant été créé à partir d'une bibliothèque dédiée au calcul matriciel, de nombreuses bibliothèques sont relatives à l'algèbre linéaire ('elmat', 'specmat', 'matfun').

Ces bibliothèques sont mises en place au moment de l'installation de *Matlab*. Il est à noter qu'elles sont payantes. Mais elles sont très utiles car elles regroupent des traitements de bases (par exemple, le calcul du déterminant d'une matrice, l'obtention des valeurs propres ou singulières et des vecteurs associés, etc.). En général, chaque programme débute par un commentaire de quelques lignes décrivant l'objectif du programme, les entrées, etc.

Comme les bibliothèques doivent être accessibles de n'importe quel point où l'on se place dans l'arborescence du disque, une variable *PATH* doit être configurée pour définir les chemins d'accès. L'ensemble des liens vers des bibliothèques existantes est disponible en tapant dans la fenêtre de commandes : *path*.

On peut sinon créer des bibliothèques personnelles contenant ses propres fonctions ou d'autres trouvées sur le net. Pour cela, il faut définir des répertoires où elles seront placées. De la même manière que précédemment, il faut configurer une variable *PATH*.

Pour cela, on suit la procédure suivante :

- création d'un répertoire associé à la bibliothèque *nom\_lib* ;
- création du fichier *startup.m* qui va contenir la mise à jour du path général de *Matlab* :

***path('/home/Matlab/nom\_lib',PATH);***

Application : Créer un répertoire de travail à partir du « poste de travail » de votre ordinateur puis votre bibliothèque personnelle nommée *libpers*. Attention : ne pas oublier de remplacer le chemin */home/Matlab/nom\_lib* par celui adapté à votre cas !

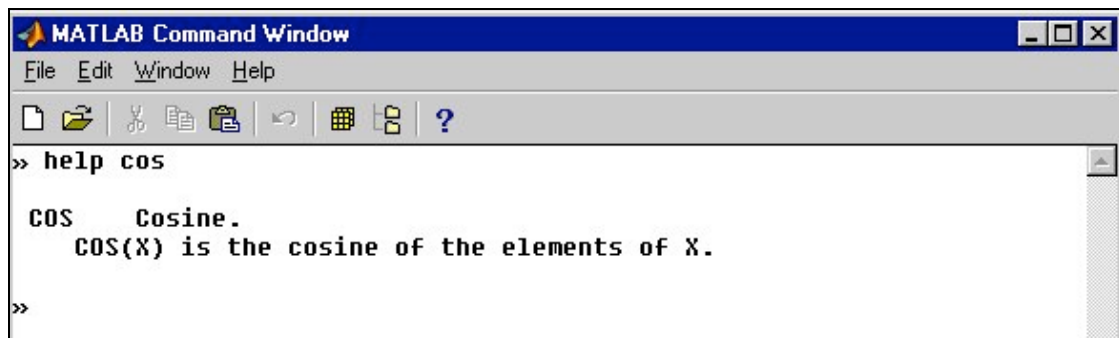
Pour explorer les bibliothèques, deux commandes *help* et *lookfor* associés à un mot clé peuvent être exploitées.

### 1.2.2 Commande *help*

Un champ de description est associé à chaque fonction ; grâce à la commande *help*, on obtient à l'écran un texte qui permet de savoir à quoi sert la fonction, mais aussi comment l'employer. Cette commande peut s'utiliser à deux niveaux :

- la commande *help* sans argument donne le listing des bibliothèques où sont regroupées les fonctions par type ;
- la commande *help "Topic"* fournit les renseignements sur les bibliothèques ou sur les fonctions selon que *Topic* équivaut à une bibliothèque ou à une fonction.

Exemple : si l'on tape *help cos*, on obtient une description de la fonction *cos*.



*Figure n°2 : utilisation de la commande help*

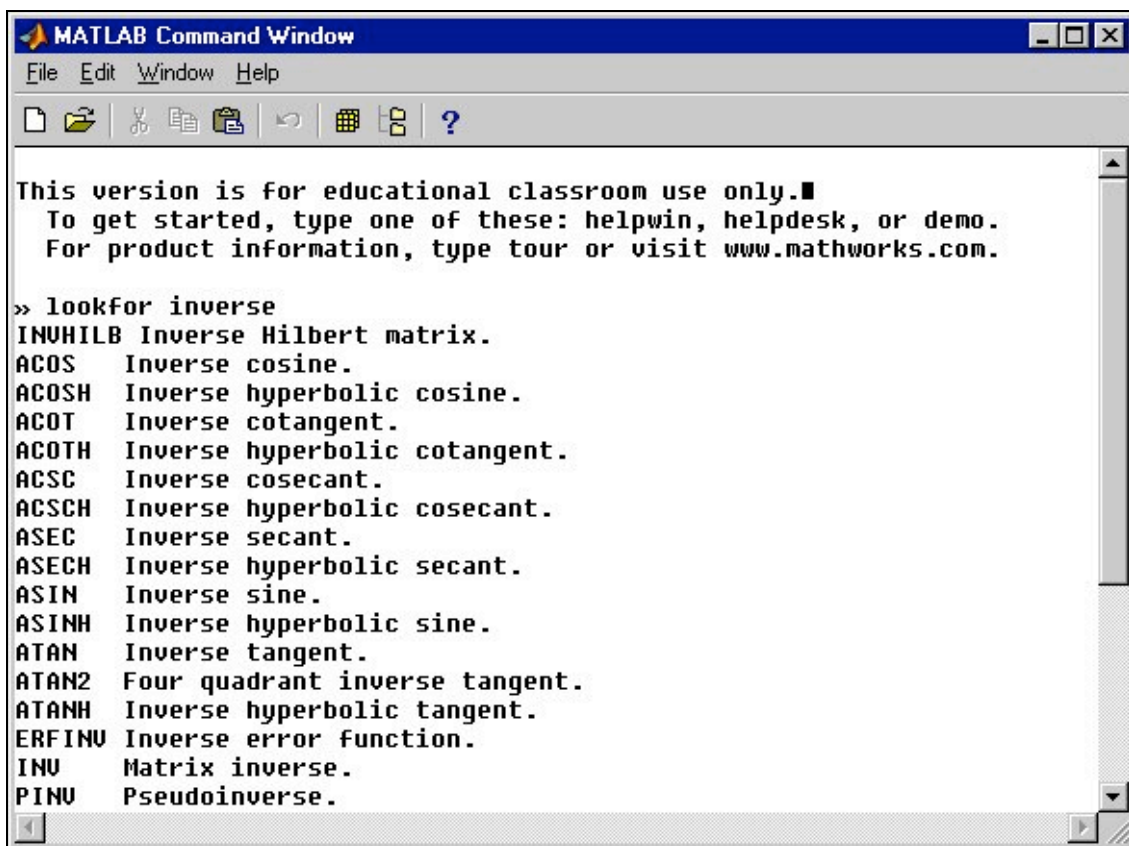
Application : lire les descriptions des fonctions *cos*, *det*, *eig*, *butter* et *plot*.

### 1.2.3 Commande *lookfor*

En utilisant la commande *lookfor*, on effectue une recherche par thème qui renvoie le nom de chaque fonction associée à ce thème et ses différents arguments.

Exemple : si l'on tape *lookfor inverse*, on obtient un listing des fonctions ayant le mot *inverse* dans leur texte de description (en anglais en général).

Application : rechercher la commande qui permet de réaliser l'affichage d'un titre pour un graphe.



*Figure n°3 : utilisation de la commande lookfor*

#### 1.2.4 Localisation de la fonction utilisée

Après avoir vu les commandes permettant de connaître les fonctionnalités des outils proposées par *Matlab*, il peut être utile de localiser la fonction étudiée sur le disque. Pour cela, on utilise la commande : ***which nom\_fonction***

Application : à quelles bibliothèques appartiennent les fonctions *cos*, *det*, *eig*, *butter*?

#### 1.2.5 Autres commandes

fonctions	Définition
<b>pwd</b>	indique le répertoire de travail
<b>cd</b>	permet le déplacement dans l'arborescence du disque
<b>ls</b>	description du contenu du répertoire

### 1.3 Commandes de gestion de l'espace de travail

#### 1.3.1 Introduction

Comme un calculateur, *Matlab* peut effectuer des opérations simples : une addition, une multiplication, une soustraction, une division, une mise à la puissance d'un nombre. L'objet de cette partie est tout d'abord d'effectuer ces opérations élémentaires puis d'introduire la notion de variables et de calculs mathématiques.

### 1.3.2 *Matlab* comme simple calculateur

Prenons un exemple simple : Yannick et Eric, « deux grands sportifs », décident de jouer au tennis pendant deux heures dans un club des Landes. Ils n'ont malheureusement pas de matériel et décident de louer les raquettes et les balles. La location d'une raquette est de 15 F pour une heure. La boîte de balle est louée 10 F pour la totalité de la partie et la location du court à l'heure est de 60 F.

Dans la fenêtre de commande, on peut immédiatement résoudre ce problème en tapant

```
>>4*15+10+2*60
```

En tapant sur entrée, le résultat s'affiche.

Il est à noter que la présence d'espace ou non entre les opérateurs n'influe pas sur le résultat, noté *ans* par défaut par *Matlab*. L'ordre des opérations se fait de gauche à droite et suit l'ordre habituel des règles de priorités. La multiplication (\*) et la division (/ ou \ ;  $100/2=2\backslash 100$ ) sont prioritaires devant l'addition (+) et la soustraction (-). La mise à la puissance (^) a l'ordre de priorité le plus élevé. Les parenthèses s'emploient elles aussi de façon classique.

Le problème que l'on vient de traiter peut se résoudre d'une autre manière en stockant les données dans des variables. Ainsi, on peut noter *raq* la variable relative au prix de la location d'une raquette, *nbre\_raq* le nombre de raquette, *balle* le prix de la location de la boîte de balles, *court* la location du court à l'heure et *heure* le nombre d'heures passées sur le terrain. On entre alors les valeurs variables une après l'autre comme suit :



```
>>raq=15    puis taper entrée !
```

Application : entrer toutes les variables et calculer le prix total que l'on stocke dans la variable *prix*.

Le nom des variables peut contenir jusqu'à 31 caractères. Ils doivent nécessairement commencer par une lettre et peuvent être suivis par des chiffres ou le caractère `_`. Les caractères de ponctuation ne sont pas acceptés. *Matlab* fait la différence entre minuscule et majuscule.

Remarque 1 : on peut valider les opérations en appuyant à chaque fois sur entrée ou les écrire sur une même ligne mais en les séparant par une **virgule**.

Remarque 2 : si une commande est suivie d'un **point virgule**, le résultat n'apparaît pas sur la fenêtre de commande mais la commande a été effectuée.

Remarque 3 : rappel des dernières commandes tapées dans la fenêtre de commandes : au lieu de réécrire une commande déjà effectuée, on peut exploiter les touches du clavier  et  pour la rappeler. Il faut cependant que le curseur soit positionné après le prompt. Après un emploi important de la fenêtre de commande, on peut accélérer la recherche de la commande, en écrivant après le prompt (>>), la première lettre de la commande souhaitée.

Remarque 4 : pour effectuer un commentaire, on commence par **%**.

Remarque 5 : on peut enfin exploiter les **...** pour signifier à *Matlab* que le reste de la commande apparaît à la ligne suivante.

Remarque 6 : On peut enfin interrompre *Matlab* à n'importe quel moment en appuyant sur **Ctrl+C**.

### 1.3.3 Nombre complexe

Pour définir la partie imaginaire d'un nombre complexe, **on multiplie par *i* ou par *j***. Pour extraire la partie imaginaire et la partie réelle d'un nombre complexe, on emploie respectivement les fonctions ***imag*** et ***real***. Le calcul du module se fait avec la fonction ***abs***. Enfin, si l'on souhaite passer à une représentation en coordonnées polaires du nombre complexe, la détermination de l'angle s'effectue à l'aide de la fonction ***angle***.

Application : entrer la variable *comp* égale à 1-2i. Retrouver sa partie réelle, sa partie imaginaire et sa représentation en coordonnées polaires.

### 1.3.4 Mathématiques, géométrie

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

L'une des bibliothèques les plus employées est celle qui contient les fonctions mathématiques géométriques les plus usuelles. Le tableau donné précédemment regroupe les principales fonctions prédéfinies relatives à ce thème.

### 1.3.5 Ecriture directe d'une matrice ou d'un vecteur

Pour définir une matrice sous *Matlab*, on suit les conventions suivantes :

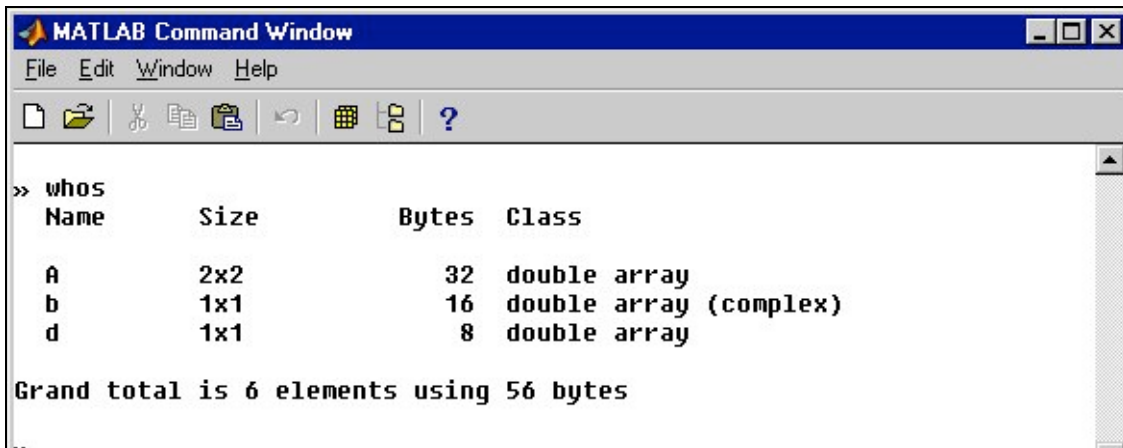
- séparer les éléments d'une même ligne par un blanc ;
- utiliser le symbole (;) pour indiquer la fin d'une ligne ;
- définir le début et la fin de la matrice par des crochets [ ] ;

Application : Entrer la matrice  $A = \begin{bmatrix} -3 & 3 \\ 4 & 5 \end{bmatrix}$  et le scalaire  $d=3$ .

### 1.3.6 Gestion des variables dans l'espace de travail : commandes *whos*, *clear*, *format*

La commande *whos* donne la liste des variables avec leurs caractéristiques définies dans l'espace mémoire. Les commandes *clear* et *format* permettent respectivement de détruire une variable de l'espace de travail et donne la possibilité de modifier le format d'affichage des variables qui sont restituées à l'écran.

Application : Détruire toutes les variables sauf la matrice *A* et le scalaire *d*. Puis, définir une variable scalaire complexe *b* en entrant `>>b = 1 + i*10` ; et taper la commande *whos*. Le résultat est le suivant :



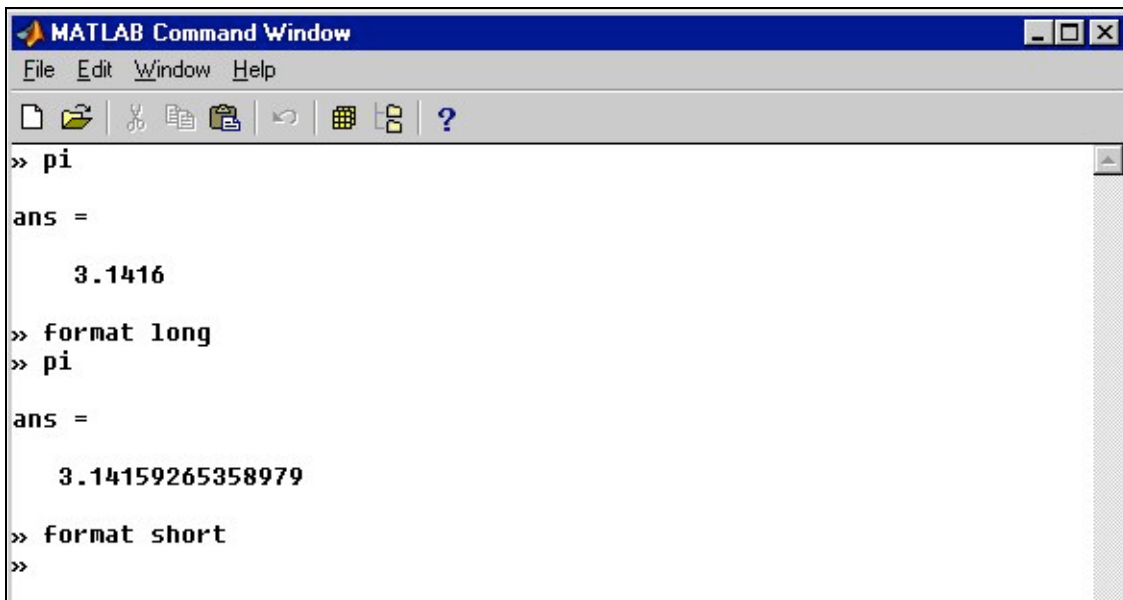
```
MATLAB Command Window
File Edit Window Help
[Icons]
>> whos
Name      Size      Bytes  Class
-----
A         2x2       32    double array
b         1x1       16    double array (complex)
d         1x1        8    double array

Grand total is 6 elements using 56 bytes
```

*Figure n°4 : utilisation de la commande whos*

Si l'on souhaite utiliser la valeur  $\pi$ , on tape *pi* (en minuscules).

Application 2 : modifier le format d'affichage de la valeur  $\pi$ .



```
MATLAB Command Window
File Edit Window Help
[Icons]
>> pi
ans =
    3.1416

>> format long
>> pi
ans =
    3.14159265358979

>> format short
>>
```

*Figure n°5 : utilisation de la commande format*



## 1.4 Opérations directes sur les matrices

### 1.4.1 Opérations arithmétiques :

Les opérations sur les matrices se font en écriture directe à l'aide des opérateurs \*, /, \, +, etc.

Application : une fois toutes les variables déjà existantes détruites, utiliser la fenêtre de

commandes pour définir :  $A = \begin{bmatrix} -3 & 3 \\ 4 & 5 \end{bmatrix}$   $x = \begin{bmatrix} 5 \\ 2 \end{bmatrix}$   $y = \begin{bmatrix} 4 \\ 1 \end{bmatrix}$ .

Calculer:

$w = A * x$  (vecteur)

$sd = w' * y$  (scalaire) . ' permet de calculer la **transposée** de la matrice.

$h = w - y$  (vecteur)

$z = w .* y$  (vecteur) . .\* permet d'effectuer une **multiplication terme à terme**.

### 1.4.2 Opérations relationnelles et logiques

fonctions	Définition
<>	différent de
<= >=	inférieur (supérieur) ou égal
==	Egal
~=	Différent
&	et logique
	Ou
~	complément logique (not)
xor	ou exclusif

## 1.5 Manipulations sur les éléments matriciels

### 1.5.1 Manipulation des indices

- Lecture d'un élément :

$A(i, j)$  permet de voir l'élément à la  $i^{\text{ème}}$  ligne et la  $j^{\text{ème}}$  colonne de  $A$  ;

- Modification d'un élément :

$A(i, j) = d$

- Lecture d'une ligne ou d'une colonne :

$A(i, :)$  renvoie la  $i^{\text{ème}}$  ligne de  $A$ .

$A(:, j)$  renvoie la  $j^{\text{ème}}$  colonne de  $A$

- Suppression de la  $i^{\text{ème}}$  ligne :

$A(i, :) = []$

Attention: après la suppression, les numéros de lignes seront décalés.

- Concaténation de deux matrices:

$B = [A, C]$  ou  $B = [A \ C]$  : concaténation horizontale: la matrice  $C$  est ajoutée à droite de la matrice  $A$

$B = [A; L]$  : concaténation verticale, la matrice  $L$  est ajoutée au bas de la matrice  $A$

Si l'on désire insérer une colonne (resp. une ligne) dans une matrice, cela revient en fait à concaténer deux matrices dont l'une est un vecteur colonne (resp. ligne). On ne peut pas insérer simplement une colonne ou une ligne à l'intérieur d'une matrice.

Application : ajouter à la matrice  $A$  une ligne dont les éléments sont  $[10 \ -2]$ . Puis, modifier la valeur de l'élément  $(3,2)$ , en lui donnant une nouvelle valeur égale à  $-205$ .

### 1.5.2 Valeurs particulières

Deux valeurs informatiques non définies sont manipulées par *Matlab* : ***NaN*** (Not a number) représente l'élément non défini et ***Inf*** est associé à l'infini.

Application : en posant  $x = [8 \ 0 \ 0]$  et  $y = [4 \ 0 \ 5]$  calculer  $d = y./x$

Remarque : les fonctions ***isnan*** et ***isinf*** permettent de savoir si dans une matrice les éléments ***NaN*** et ***inf*** sont présents.

## 1.6 Matrices creuses (*help sparsfun*)

Les matrices creuses (sparse matrix) sont des matrices caractérisées par un nombre très important d'éléments nuls. D'un point de vue informatique, un traitement particulier de ces matrices permet un **gain en place mémoire** et en coût calculatoire.

fonctions	Définition
<b>sparse</b>	création d'une matrice creuse
<b>full</b>	conversion d'une matrice creuse
<b>issparse</b>	retourne 1 si la matrice est creuse
<b>speye</b>	matrice identité creuse

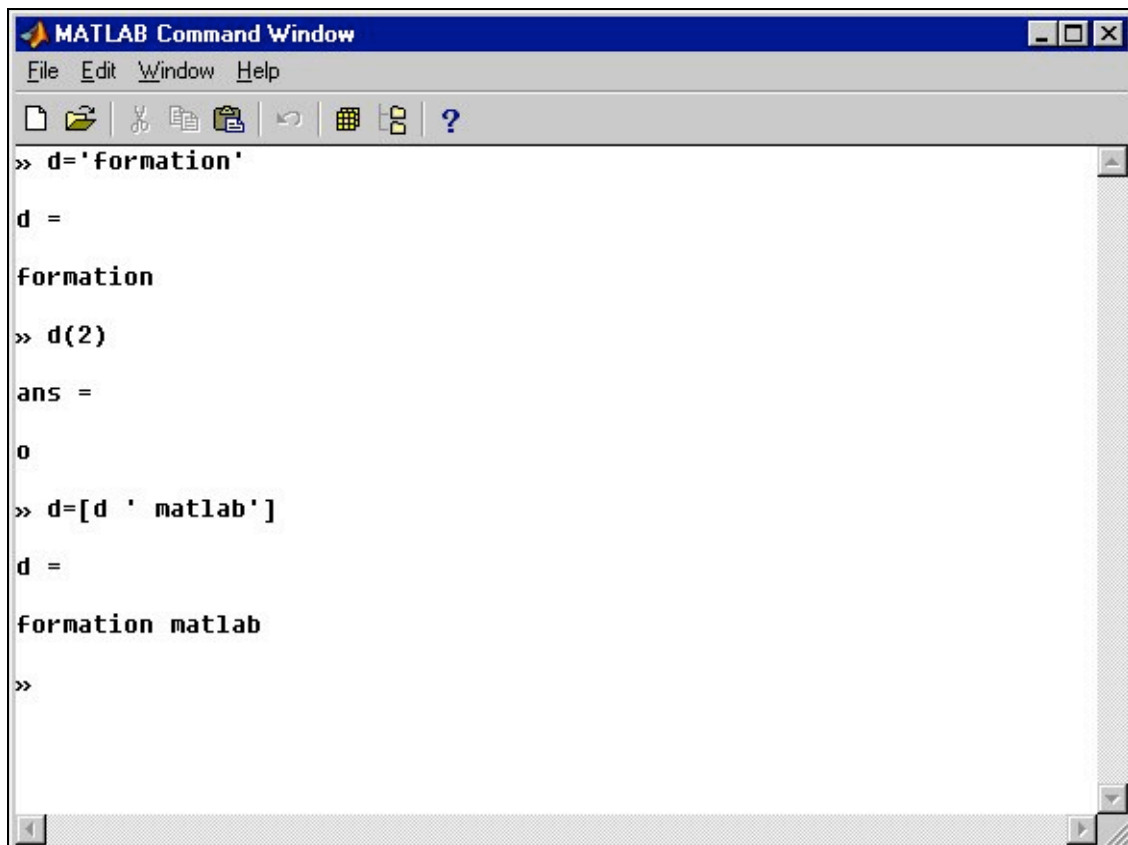
Application : Créer une matrice aléatoire de taille  $6 \times 6$ , notée  $A$  avec la fonction ***rand***. Générer la matrice  $B$  ne contenant que les éléments supérieurs à  $0.92$  et des zéros ailleurs ( $A > 0.92$ ).\* $A$ . Cette matrice est alors transformée en matrice creuse en utilisant les fonctions adaptées. Vous vérifierez le résultat avec la commande ***whos***.

Remarque : on verra dans la partie 2 que la fonction ***rand*** permet la génération de processus aléatoire

## 1.7 Matrice et chaîne de caractères

Dans *Matlab*, les chaînes de caractères sont des matrices. Elles sont définies par des guillemets simples : 'entre guillemets simples'.

Remarque: Il ne s'agit pas des guillemets inversés : `pas de guillemets inversés`, ni de guillemets doubles : "pas de guillemets doubles".



```

MATLAB Command Window
File Edit Window Help
[Icons]
>> d='formation'
d =
formation
>> d(2)
ans =
o
>> d=[d ' matlab']
d =
formation matlab
>>

```

*Figure n°6 : exemple sur les chaînes de caractères*

Les fonctions associées se trouvent dans la toolbox « strfun ». (*help strfun*). On y trouve les fonctions de conversion, de comparaison, de recherche, etc.

## **1.8 Bilan et exercice d'application**

### **1.8.1 Ce qu'il faut (au minimum) retenir !**

*Matlab* est un logiciel qui dispose d'une collection de bibliothèques contenant des fonctions/programmes extension **.m**. Comme un calculateur, *Matlab* peut effectuer des opérations simples et peut stocker des variables de nature différentes (matrice, vecteur, scalaire, réel, complexe, etc.).

Parmi les commandes à connaître : *path*, *help*, *lookfor*, *which*, *pwd*, *cd*, *ls*, *whos*, *clear*, *format*, etc.

On peut interrompre *Matlab* à n'importe quel moment en appuyant sur Ctrl+C.

### 1.8.2 Exercice d'application 1

- Créer la matrice suivante  $A = \begin{bmatrix} 16 & 5 & 9 & 4 & -1 \\ 3 & 10 & 6 & 15 & 2 \\ 2 & 11 & 7 & 14 & 17 \\ 13 & 8 & 12 & 1 & 4 \end{bmatrix}$
- Créez une matrice  $B = \{b_{ij}\}$  de la même taille que  $A = \{a_{ij}\}$ , et pour laquelle  $b_{ij}=1$  si  $6 \leq a_{ij} \leq 11$ , et  $b_{ij}=0$  sinon.
- Rechercher les éléments de la matrice  $A = \{a_{ij}\}$  qui obéissent à la relation  $6 \leq a_{ij} \leq 11$ . On utilisera pour cela la fonction **find**, qui permettra de déterminer les éléments de la matrice B égaux à 1.
- Calculer la somme de la diagonale commençant par l'élément  $a_{1,2}$ . On utilisera pour cela les fonctions **diag** et **sum**.
- Calculer le déterminant de la matrice  $A * A^T * a_{1,2}$ . On utilisera pour cela la fonction **det**.
- Générer à l'aide de la fonction **rand** une matrice D dont les éléments sont aléatoires de taille (4×4). Créer une matrice binaire à l'aide des opérateurs relationnels correspondant aux éléments de D qui sont supérieurs à 0.2 et inférieurs à 0.8. Créer alors la matrice G qui ne contient que les éléments de D donnés par les inégalités et des zéros ailleurs.

## PARTIE N° 2

## SUR CERTAINES FONCTIONS DEDIEES AU CALCUL

---

### 2.1 Initialisation et manipulation

Deux fonctions permettent de générer des matrices initialisées à zéro ou à une valeur quelconque.

fonctions	Définition
<b>zeros</b>	remplissage de zéros
<b>ones</b>	remplissage de 1
<b>eye</b>	matrice identité

Application : Créer un vecteur ligne contenant successivement douze 0, puis huit 1 et enfin douze 0.

Application 2 : Créer les matrices suivantes à l'aide des fonctions *zeros*, *ones* et *eye*, et en les concaténant horizontalement ou verticalement de façon appropriée.

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -0.2 & 0.1 & 0.2 & -0.05 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}.$$

Différentes fonctions permettent de modifier l'organisation des éléments d'une matrice à partir de rotations ou de symétries.

fonctions	Définition
<b>flipud</b>	symétrie horizontale (up-down)
<b>fliplr</b>	symétrie verticale (left-right)
<b>triu</b>	partie triangulaire supérieure
<b>tril</b>	partie triangulaire inférieure
<b>sort</b>	arrangement par ordre croissant
<b>rot90</b>	rotation de 90°

Application : A partir d'une matrice de taille 4\*4, opérer successivement sur cette même matrice une rotation de 90° puis une symétrie verticale.

## 2.2 Algèbre linéaire

De nombreuses fonctions relatives à l'algèbre existent dans *Matlab*. Cet outil a été développé en effet dans le contexte de l'algèbre dans ses versions préliminaires. Notre objectif n'est pas de fournir un descriptif exhaustif de toutes les fonctions. On se limite à certain nombre afin de proposer un point d'entrée à l'utilisateur.

### 2.2.1 Fonctions élémentaires

Les fonctions élémentaires regroupent les fonctions qui restituent une grandeur relative à la matrice étudiée.

fonctions	définition
<b>trace</b>	trace de la matrice
<b>det</b>	déterminant
<b>norm</b>	normes 1, 2 etc. de la matrice
<b>cond</b>	conditionnement
<b>rank</b>	rang

Application : Calculer le rang de la matrice  $\begin{bmatrix} 1 & 2 & -1 & 2 \\ 2 & 4 & -2 & 4 \end{bmatrix}$ .

### 2.2.2 Fonctions de décomposition

De nombreuses techniques de décomposition existent dans l'algèbre linéaire. Ces décompositions se retrouvent à travers les fonctions suivantes :

fonctions	définition
<b>eig</b>	décomposition en éléments propres
<b>svd</b>	décomposition en éléments singuliers

Application : Calculer les valeurs singulières associée à la décomposition SVD de la matrice  $A$  donnée par la relation  $A = UDV^T$

$$A = \begin{bmatrix} 2 & 0 & -1 & 2 \\ 3 & 1 & 6 & 5 \\ 4 & 2 & 1 & 8 \end{bmatrix}$$

Vérifier que l'on a les valeurs propres de la matrice  $AA^T$  qui correspondent aux valeurs singulières élevées au carré.

## 2.3 Opérations sur les données

### 2.3.1 Génération aléatoire

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

fonctions	définition
<b>rand</b>	remplissage de valeurs aléatoires : distribution uniforme
<b>randn</b>	remplissage de valeurs aléatoires : distribution gaussienne

### 2.3.2 Analyse des données

Certaines de ces fonctions sont relatives à la caractérisation d'un processus aléatoire. Elles permettent toutes de caractériser le jeu de données qui est en entrée.

fonctions	définition
<b>median</b>	valeur médiane
<b>max</b>	maximum
<b>min</b>	minimum
<b>std</b>	écart type (standard deviation) Cf. cours de probabilités, processus aléatoires
<b>mean</b>	moyenne
<b>corrcoef</b>	coefficient de corrélation Cf. cours de probabilités, processus aléatoires
<b>prod</b>	produit des éléments
<b>sum</b>	somme des éléments

Application : Générer à l'aide de la fonction **rand** 15 échantillons d'un processus aléatoire de distribution uniforme. Calculer sa moyenne et sa variance. En outre, extraire la valeur max et min du jeu de données en incluant l'information relative à leur position dans le vecteur.

## 2.4 Opérations sur les Polynômes

Les différentes fonctions présentées dans ce paragraphe sont relatives aux polynômes, ces polynômes sont gérés sous *Matlab* en suivant un ordre décroissant pour l'exposant et donc pour la déclaration de leurs coefficients.

fonctions	définition
<b>roots</b>	Calcul des racines connaissant les coefficients
<b>poly</b>	Calcul des coefficients connaissant les racines
<b>polyval</b>	Calcul de P(x)
<b>polyfit</b>	Identification des coefficients d'un polynôme à l'aide des moindres carrés
<b>conv</b>	Produit de deux polynômes
<b>deconv</b>	Division de deux polynômes

Application : Déterminer les coefficients du polynôme caractérisé par les racines  $A = [0,5 \quad 2 \quad 4,3]$ . Calculer la valeur prise par  $P(x)$  pour  $x = -2$ .

Application 2 Evaluer les coefficients du polynôme caractéristique associé à la matrice  $A = \begin{bmatrix} 2 & -1 \\ 3 & 7 \end{bmatrix}$ .



### **3.1 Introduction**

Après avoir vu un certain nombre de commandes utilisées directement sous le prompt de *Matlab*, on s'intéresse maintenant à la construction de programmes contenant plusieurs commandes. La programmation sous le logiciel *Matlab* conserve l'esprit de simplicité à savoir une programmation directe sans pointeur ou autre. Pour développer une application, deux éléments sont donc à considérer :

1. le programme principal appelé "script".
2. les éléments de type fonction ou sous-fonction.

### **3.2 Script**

#### **3.2.1 Définition d'un script**

Un programme *Matlab* est un fichier texte contenant un certain nombre de commandes *Matlab*. Sur le disque, il apparaît toujours de la façon suivante :

*nom\_script.m*

Ce programme s'écrit à l'aide d'un éditeur de texte quelconque. Par défaut, vous avez celui proposé par *Matlab* mais il peut être modifié à l'aide du menu **préférences**.

#### **3.2.2 Construction d'un script**

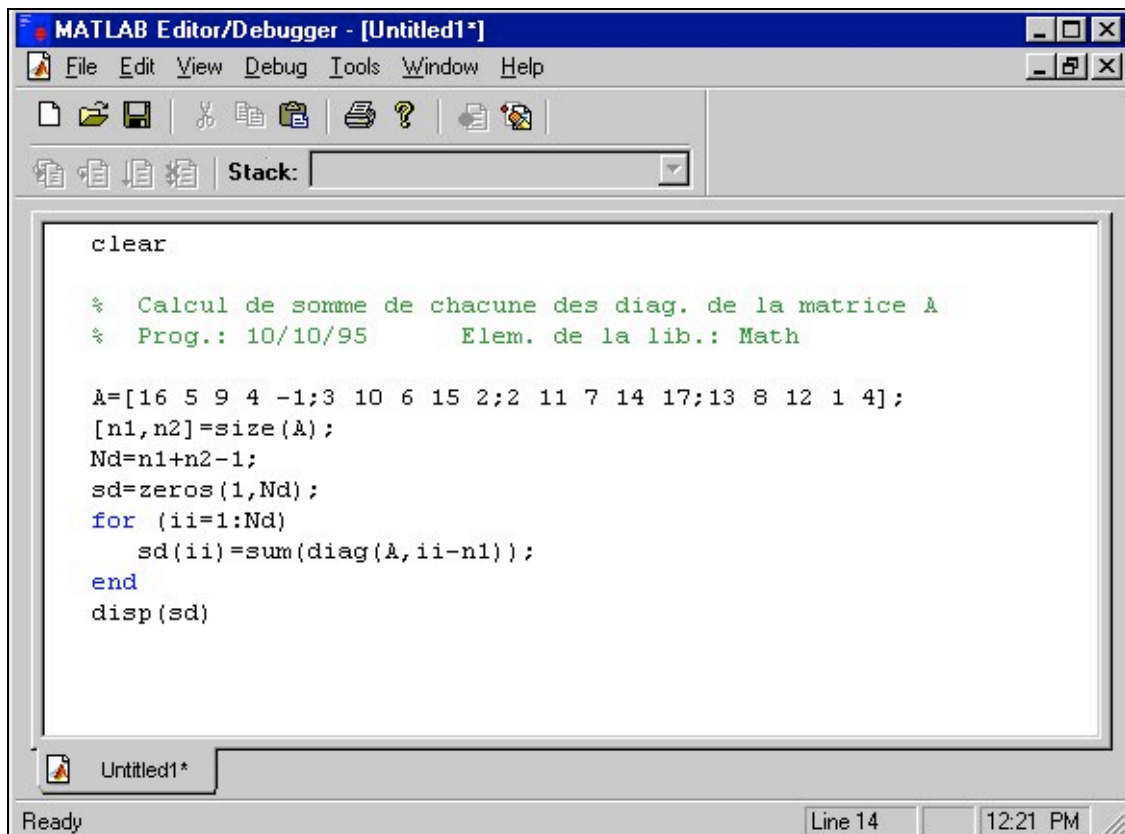
Leur construction est très simple. En fait, il s'agit d'une succession de commandes dans un fichier texte, celles-ci même que l'on entrait précédemment sous le prompt. Pour réaliser un bon script, les trois points suivants doivent être respectés :

1. Pour plus de sécurité, toujours débiter un script par la commande ***clear*** pour que les données contenues dans l'espace de travail n'interfèrent pas avec les variables utilisées dans le script.
2. Créer une zone de commentaires. En effet, pour réaliser une bonne gestion de ses programmes, il est utile de leur associer un commentaire détaillant leur fonctionnement. Pour chaque ligne de commentaire, le sigle **%** doit être tapé au début de chaque ligne. Il est à noter que ces commentaires sont accessibles avec la commande ***help***.

3. Comme *Matlab* est un langage pré-compilé, il faut être rigoureux dans sa programmation en vue d'une économie de temps de traitement.

Application :

calcul des sommes des éléments de chacune des diagonales d'une matrice.



```
clear

% Calcul de somme de chacune des diag. de la matrice A
% Prog.: 10/10/95      Elem. de la lib.: Math

A=[16 5 9 4 -1;3 10 6 15 2;2 11 7 14 17;13 8 12 1 4];
[n1,n2]=size(A);
Nd=n1+n2-1;
sd=zeros(1,Nd);
for (ii=1:Nd)
    sd(ii)=sum(diag(A,ii-n1));
end
disp(sd)
```

*Figure n°7 : exemple de script*

*Une fois le script écrit et enregistré, on peut l'exécuter en tapant sur la fenêtre de commande le nom du programme.*

### **3.3 Fonctions**

#### **3.3.1 Introduction**

Les fonctions ont un rôle très important dans *Matlab* puisque ce sont elles qui constituent un de ses principaux atouts. Ces fonctions sont utilisables à partir d'un script ou directement de l'espace de travail.

#### **3.3.2 Définition**

Une fonction est définie par :

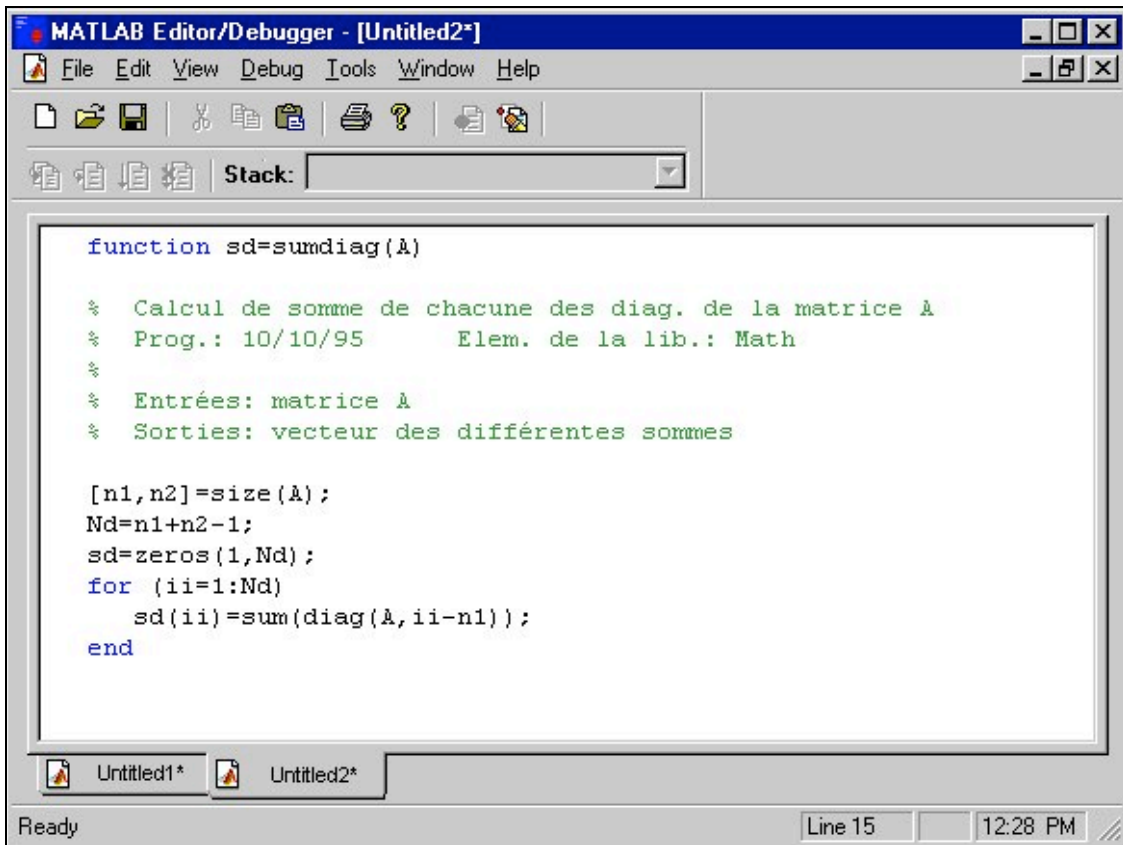
- son nom ;
- ses entrées ;
- ses sorties.

syntaxe générale :

*function [sortie\_1,sortie\_2, ...]=nom\_fonc(entrée\_1,entrée\_2, ...)*

### 3.3.3 Exemple de fonction

Pour illustrer le principe de construction d'une fonction, reprenons l'exemple du calcul des diagonales, mais en créant une fonction utilisable pour une matrice A quelconque. On exploite cette fonction sous le prompt de *Matlab*.



*Figure n°8 : exemple de fonction*

### 3.4 Structures de programmation usuelles

On rappelle dans ce paragraphe les principales structures de programmation que l'on rencontre en général. Sous *Matlab*, le nombre de ces structures est très réduit puisque l'on en a deux types :

*for* (*k = Val\_Init : Pas : Val\_fin*)

↑  
↓  
*end*

*while* (*condition*)

↑  
↓  
*end*

### 3.4.1 Structure de type test

```
if(condition)  
↑  
↓  
else  
↑  
↓  
end
```

```
if(condition1)  
↑  
↓  
elseif(condition2)  
↑  
↓  
else  
↑  
↓  
end
```

```
switch  
↑  
case  
↑  
↓  
otherwise  
↑  
↓  
end
```

Remarque :

Comme *Matlab* est un logiciel interprété, il faut utiliser avec parcimonie les boucles **for** et **while**. Il vaut mieux dès que possible utiliser la boucle interprétée " : ".

Pour remplir un vecteur A des valeurs de 11 à 20, au lieu de :

```
A=zeros(1,10);  
for ii=1:10  
    A(ii)=ii+10;  
end
```

il vaut mieux

```
A=11:20;
```

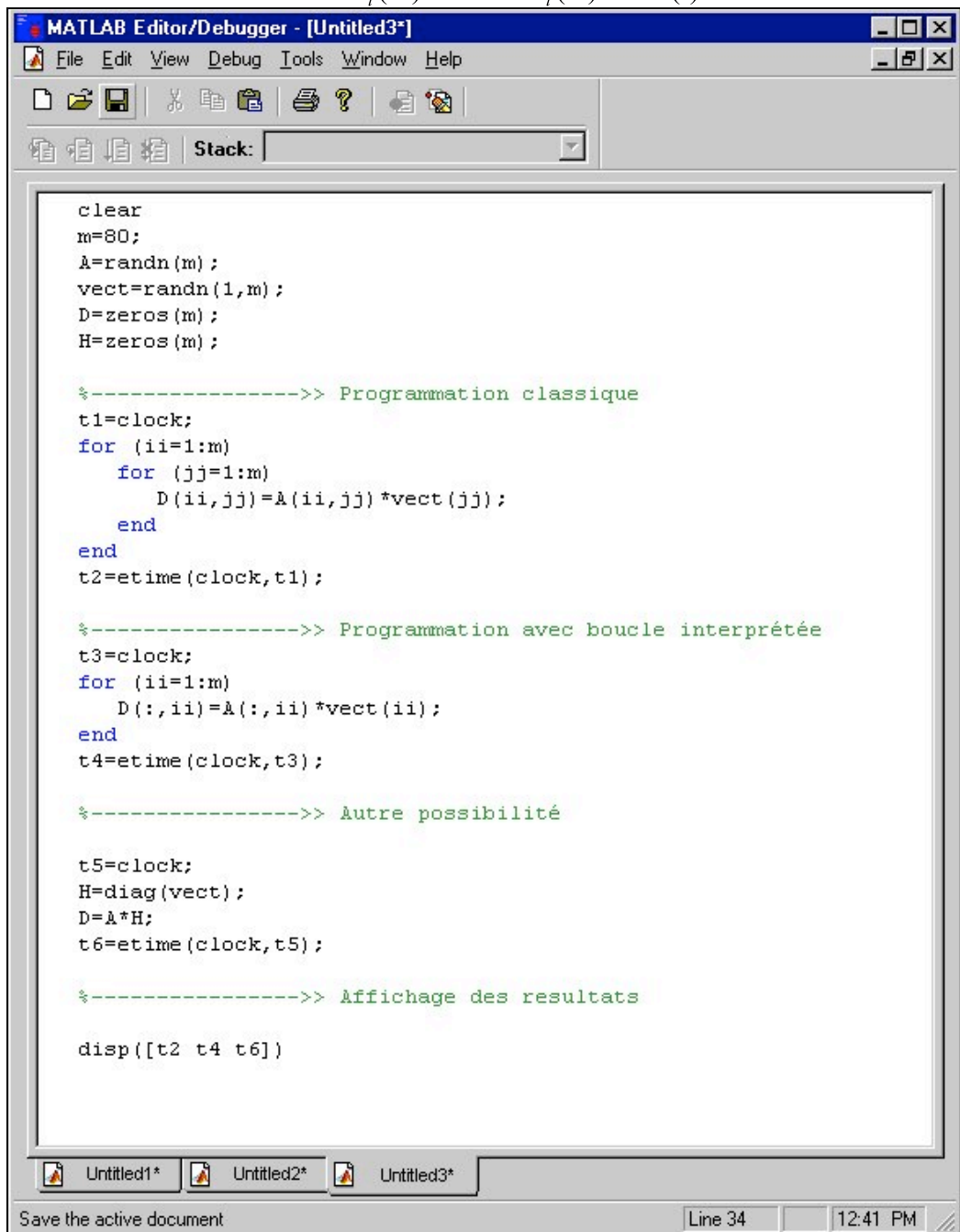
ou bien

```
A=[1:10]+10;
```

Application :

A partir d'une matrice  $A$  et d'un vecteur  $vect$ , on veut calculer la matrice  $D$  telle que :

$$colonne_i(D) = colonne_i(A) * vect(i)$$



```
clear
m=80;
A=randn(m);
vect=randn(1,m);
D=zeros(m);
H=zeros(m);

%----->> Programmation classique
t1=clock;
for (ii=1:m)
    for (jj=1:m)
        D(ii,jj)=A(ii,jj)*vect(jj);
    end
end
t2=etime(clock,t1);

%----->> Programmation avec boucle interprétée
t3=clock;
for (ii=1:m)
    D(:,ii)=A(:,ii)*vect(ii);
end
t4=etime(clock,t3);

%----->> Autre possibilité

t5=clock;
H=diag(vect);
D=A*H;
t6=etime(clock,t5);

%----->> Affichage des resultats

disp([t2 t4 t6])
```

*Figure n°9 : exemple d'utilisation des boucles et leurs inconvénients*

### ***3.5 Bilan et Exercices d'application : Création d'une fonction***

- 1) Générer un vecteur complexe défini par  $A=[0.22+0.5*i; 0.4-0.2*i]$
- 2) Extraire les parties réelles, imaginaires, modules et phases des éléments du vecteur  $A$ .
- 3) Créer une fonction ***farg\_A*** permettant de calculer la phase des éléments complexes d'une matrice  $A$ . Deux arguments en entrée pour la fonction sont imposés: le premier représente la matrice  $A$  et le second est une variable texte qui permet de sortir le résultat, soit en radian, soit en degré.     ***Strcmp***

### 4.1 Préambule

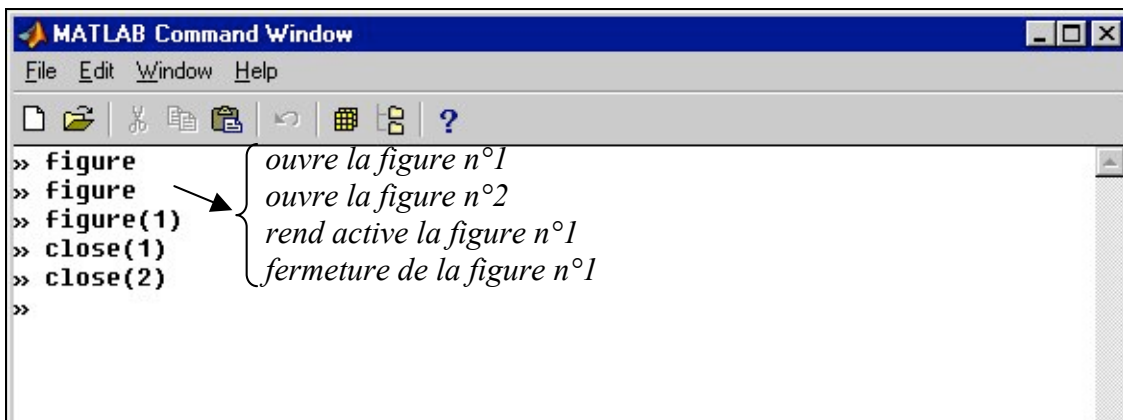
Une des priorités des « développeurs » de *Matlab* est aujourd'hui d'offrir aux utilisateurs un maximum d'outils graphiques. L'approche "orientée objet" a permis de donner une très grande efficacité et souplesse à ces outils. L'aspect des graphes que l'on manipule peut ainsi à tout moment être modifié par exemple par un changement de format du graphe, des couleurs ou du style de la représentation.

### 4.2 Notion de figure

Avant de détailler les différentes fonctions de visualisation, il faut noter que l'on peut générer simultanément plusieurs graphes à l'écran. Pour cela, on utilise la commande **figure** qui permet d'ouvrir autant de fenêtres graphiques que désiré. Chaque **figure** porte un numéro qui permet de la référencer et ainsi de savoir où l'on envoie les sorties graphiques. Les figures sont indexées automatiquement, la commande **close** permet de les fermer à partir de l'espace de travail. Il est à noter que toutes les fenêtres se ferment à l'aide de la commande **close all**.

Application :

Ouvrir deux figures et les refermer.



```

MATLAB Command Window
File Edit Window Help
[Icons]
>> figure      ouvre la figure n°1
>> figure      ouvre la figure n°2
>> figure(1)   rend active la figure n°1
>> close(1)    fermeture de la figure n°1
>> close(2)
>>

```

Figure n°10 : ouverture et fermeture de fenêtre pour figure

### 4.3 Affichage des courbes 2D

*Matlab* possède un grand nombre de fonctions permettant de produire des courbes 2D. Chacune accepte en entrée des formes vectorielles ou matricielles automatiquement mises à l'échelle suivant les axes définis par ces données.

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

Outils	définition
<i>polar</i>	Affi. en coordonnées polaires
<i>bar</i>	Affi. en mode escalier
<i>hist</i>	Affi. d'un histogramme
<i>fplot</i>	Affi. d'une fonction donnée

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

#### 4.3.1 Commande *plot*

La commande *plot* génère l'affichage des éléments d'un vecteur ou des colonnes d'une matrice. Si  $y$  est un vecteur, *plot(y)* produit un affichage linéaire des éléments de  $y$  suivant l'index du vecteur. Si on spécifie deux vecteurs en entrée *plot(x,y)*, on obtient l'affichage des éléments de  $y$  suivant ceux de  $x$ . Les couleurs et les symboles associés aux données sont paramétrables (voir *help plot*).

Application :

1. Réaliser l'affichage de la fonction *Chirp* définie par  $\sin(t^2)$  sur l'intervalle:  $0 < t < 5$  avec un pas de 0.1 en utilisant la commande *plot*. Ajoutez une légende à l'aide de *title*, *xlabel* et *ylabel*.
2. Modifier le style de la courbe pour ne représenter que les points calculés, sans les lignes entre points consécutifs. Choisir ensuite votre propre mode de représentation.



### 4.3.2 Commande *fplot*

Cette commande permet d'afficher une fonction sans l'effet de l'échantillonnage. Pour cela, on doit en premier lieu créer une fonction *Matlab* représentant la fonction mathématique à étudier.

```
function x=chirp(t);  
% Cette fonction crée un signal "chirp"  
x=sin(t.^2);
```

Puis, sous le prompt *Matlab*, on entre:

```
» fplot('chirp',[0 5])
```

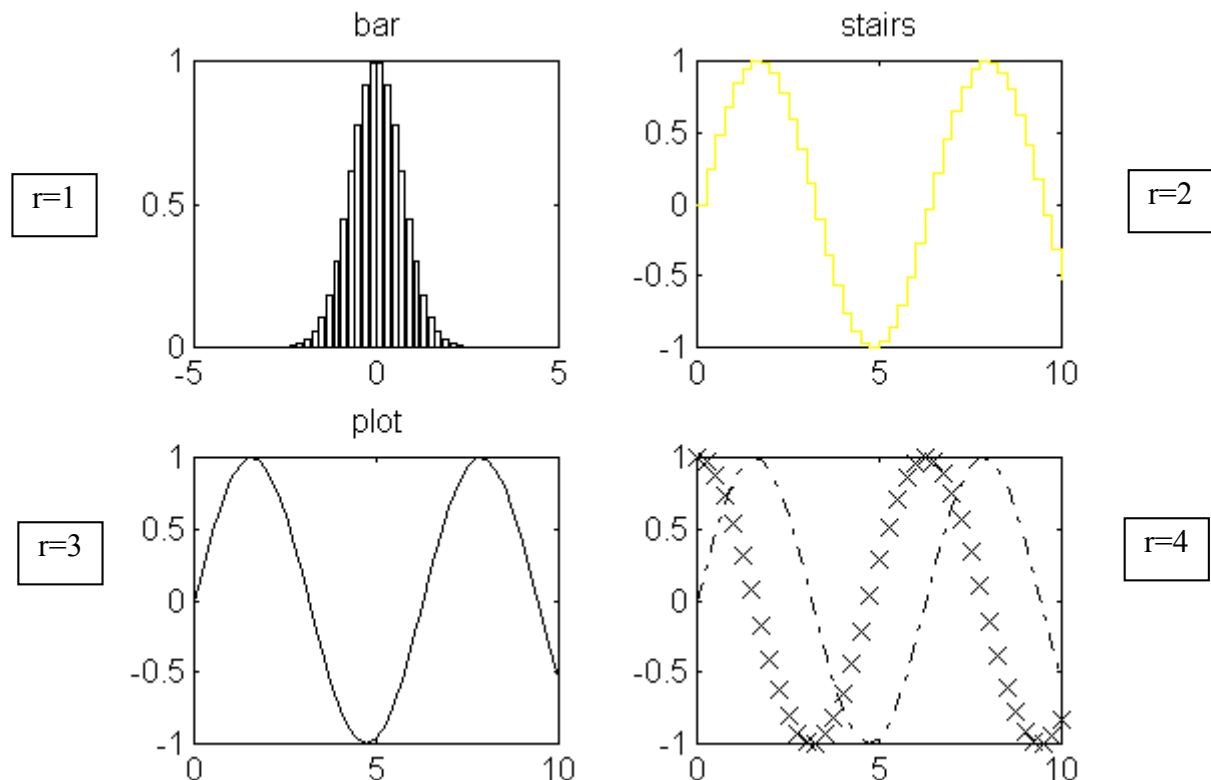
Cette fonction calcule plus finement la fonction sur les portions où la fonction présente des discontinuités graphiques.

### 4.3.3 Commande *subplot*

Cette commande est un outil qui permet de gérer le nombre de graphes que l'on crée sur une même figure. Elle partitionne la figure comme une matrice  $(n,p)$  dont les éléments sont des graphes.

*subplot(n,p,r)*

où:  $n$  est le nombre de lignes,  $p$  est le nombre de colonnes,  
 $r$  est le numéro de placement dans la matrice (colonne par colonne).



*Figure n°11 : utilisation du subplot*

#### 4.3.4 Commande *axis*

Lorsque l'on veut ne visualiser qu'une portion d'un vecteur de données, on utilise la commande *axis*. Elle peut être utilisée de la façon suivante :

2D *axis*([ xmin xmax ymin ymax])

3D *axis*([ xmin xmax ymin ymax zmin zmax])

Elle permet de gérer les échelles sur les différents axes. Elle peut en outre être utilisée comme un zoom statique. Il est à noter que cette commande permet aussi de gérer l'aspect général des axes. En effet, on peut l'utiliser avec du texte comme entrée, ainsi:

*axis('square')* l'espace graphique prend une forme carré.

*axis('equal')* indique que les marqueurs sur les axes seront les mêmes.

*axis('off')* supprime les axes à l'écran.

*axis('on')* restaure les axes à l'écran.

#### 4.4 Affichage des courbes 3D

Comme pour le cas 2D, on dispose d'un certain nombre de fonctions de visualisation des données définies dans un domaine tridimensionnel.

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

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

Certaines de ces fonctions attendent des arguments de type vectoriel ou matriciel.

##### 4.4.1 *plot3* : arguments vectoriels

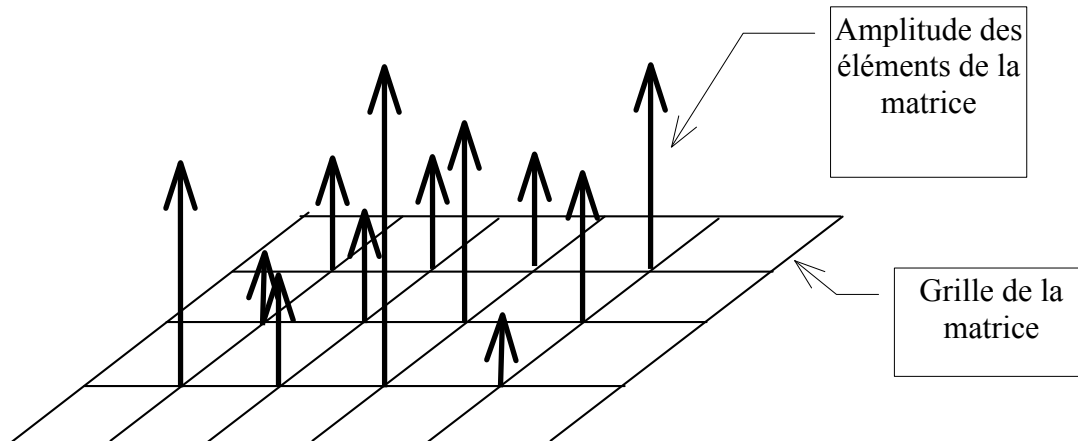
Cette fonction est identique à la commande *plot*, mais avec  $x$ ,  $y$  et  $z$  comme arguments.

Application : Définir une variable définissant le segment  $[0,10\pi]$  par pas de  $\pi/50$ . Entrer la commande *plot3(sin(t),cos(t),t)*

#### 4.4.2 Fonctions à arguments matriciels

Ces fonctions permettent de visualiser des données définies sur une grille représentée par une matrice. Différentes possibilités de visualisation existent :

- soit par niveau de gris (0-256) -->> **image**
- soit par niveau d'amplitude -->> **contour**
- soit par représentation de la grille -->> **mesh,surf**



*Figure n°12 : fonctions à argument matriciel*

#### 4.4.3 meshgrid

Toutes ces fonctions nécessitent des matrices comme entrées. Dans certains cas, on est obligé de générer soi-même la matrice à visualiser. Pour cela, *Matlab* possède une fonction qui permet d'obtenir les points de la grille représentant les  $x$  et  $y$ .

Application :

$$z(x,y) = 3(1-x)^2 e^{-x^2-(y+1)^2} - 10\left(\frac{x}{5} - x^3 - y^3\right)e^{-x^2-y^2} - \frac{1}{3}e^{-(x+1)^2-y^2}$$

On choisit pour cela la région:

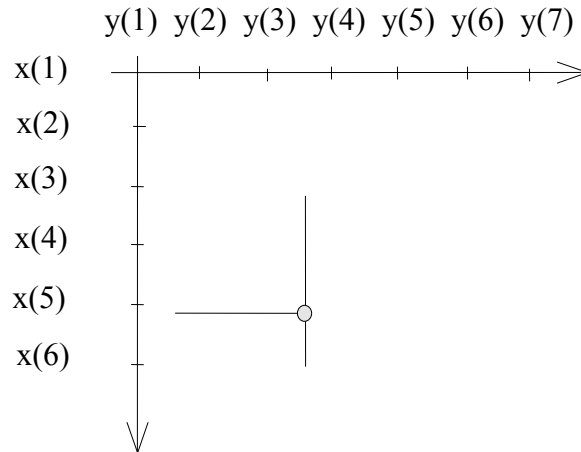
$$\begin{cases} -3 \leq x \leq 3 \\ -3 \leq y \leq 3 \end{cases}$$

Pour définir la grille de visualisation à partir des vecteurs définis plus haut, on utilise la commande **meshgrid** :

$$[X,Y]=\text{meshgrid}(-3:1/8:3);$$

La matrice  $Z$  contient tous les points calculés à partir de la grille générée par les matrices  $X$  et  $Y$ .

$$X = \begin{bmatrix} x(1) & x(1) & x(1) & x(1) & x(1) & x(1) & \dots \\ x(2) & x(2) & x(2) & x(2) & x(2) & x(2) & \dots \\ x(3) & x(3) & x(3) & x(3) & x(3) & x(3) & \dots \\ x(4) & x(4) & \boxed{x(4)} & x(4) & x(4) & x(4) & \dots \\ x(5) & x(5) & x(5) & x(5) & x(5) & x(5) & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \quad Y = \begin{bmatrix} y(1) & y(2) & y(3) & y(4) & y(5) & y(6) & \dots \\ y(1) & y(2) & y(3) & y(4) & y(5) & y(6) & \dots \\ y(1) & y(2) & y(3) & y(4) & y(5) & y(6) & \dots \\ y(1) & y(2) & \boxed{y(3)} & y(4) & y(5) & y(6) & \dots \\ y(1) & y(2) & y(3) & y(4) & y(5) & y(6) & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$



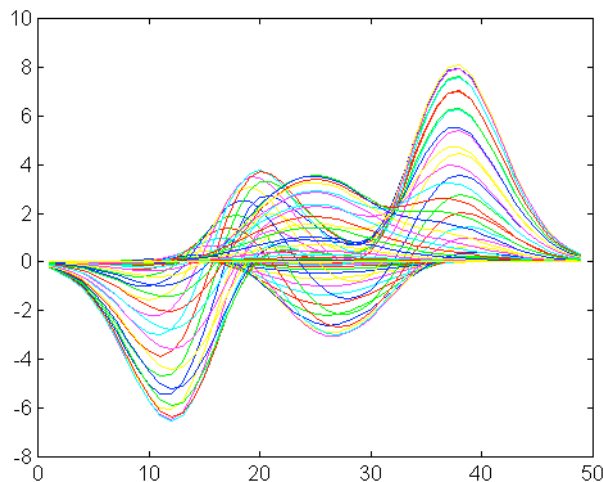
Il suffit alors d'utiliser l'opérateur "élément par élément" pour les opérations sur les matrices ainsi définies. Le début du programme s'écrit donc :

```
% Visualisation 3D
[x,y]=meshgrid(-3:1/8:3);
z = 3*(1-x).^2.*exp(-(x.^2) - (y+1).^2) ...
- 10*(x/5 - x.^3 - y.^5).*exp(-x.^2-y.^2) ...
- 1/3*exp(-(x+1).^2 - y.^2);
```

On étudie maintenant les différents outils pour visualiser la matrice Z.

#### 4.4.4 plot

Cette fonction, la plus simple, permet de tracer une ligne pour chaque colonne de la matrice Z.



#### 4.4.5 *contour* et *contour3*

Avec la fonction *contour*, on visualise les données contenues dans la matrice *Z* par courbes de niveau. *Matlab* trace les courbes joignant les mêmes niveaux d'amplitude de la matrice *Z* sur un plan 2D. *Contour3* réalise la même fonction, mais dans un plan 3D, le nombre de niveaux étant paramétrable.

*contour(x,y,z,M)*

où *x*, *y* sont les matrices ou vecteurs des axes, *z* la matrice des amplitudes et *M* le nombre de niveaux.

Exemple :

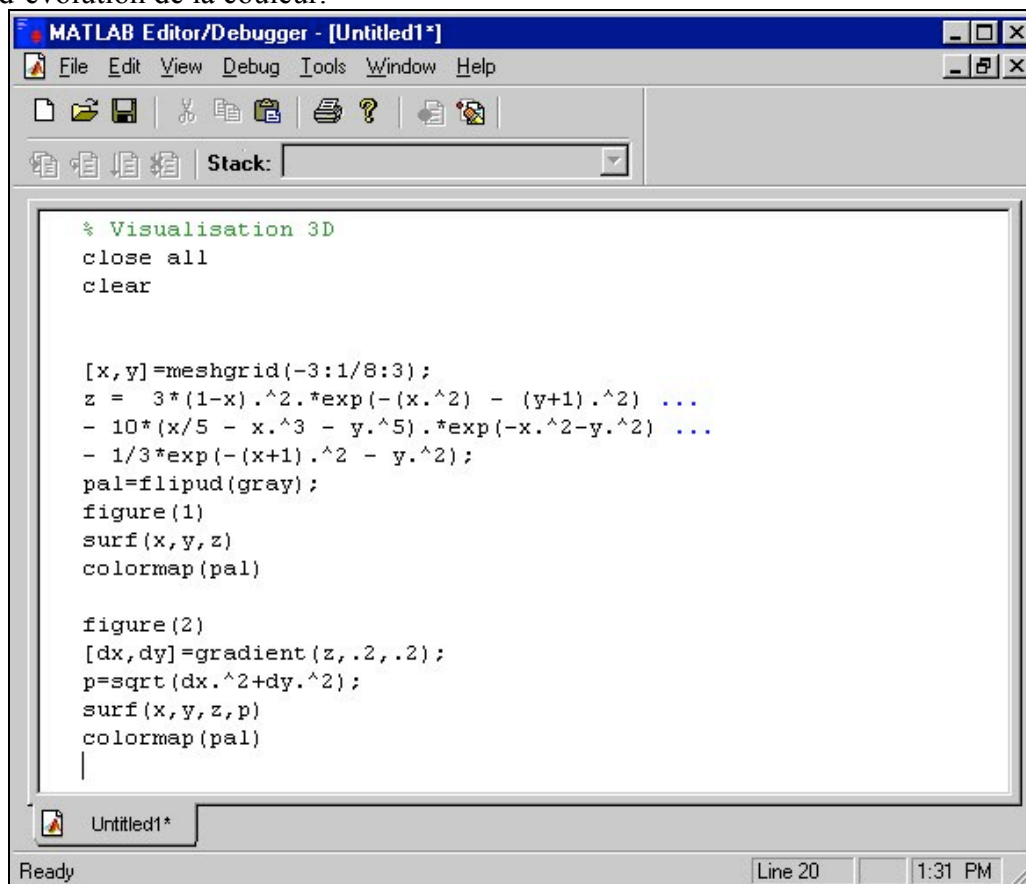
```
figure(1); contour(Z,20)  
figure(2); contour3(Z,20)
```

#### 4.4.6 *mesh* et *surf*

Les commandes *mesh* et *surf* affichent une surface en 3 dimensions en perspective. La commande *mesh* associe les couleurs aux éléments de la grille tandis que *surf* colore les facettes de la grille. (idem pour *surfc*, *meshc*, *meshz*)

*mesh(x,y,z,C) ; surf(x,y,z,C)*

où *x*, *y* sont les matrices ou vecteurs des axes, *z* la matrice des amplitudes et *C* représente le mode d'évolution de la couleur.



```
MATLAB Editor/Debugger - [Untitled1*]  
File Edit View Debug Tools Window Help  
Stack:  
% Visualisation 3D  
close all  
clear  
  
[x,y]=meshgrid(-3:1/8:3);  
z = 3*(1-x).^2.*exp(-(x.^2) - (y+1).^2) ...  
- 10*(x/5 - x.^3 - y.^5).*exp(-x.^2-y.^2) ...  
- 1/3*exp(-(x+1).^2 - y.^2);  
pal=flipud(gray);  
figure(1)  
surf(x,y,z)  
colormap(pal)  
  
figure(2)  
[dx,dy]=gradient(z,.2,.2);  
p=sqrt(dx.^2+dy.^2);  
surf(x,y,z,p)  
colormap(pal)  
|  
Ready Line 20 1:31 PM
```

*Figure n°13 : fonctions à argument matriciel*

#### 4.4.7 *image*

fonctions	définition
<i>image</i>	Affichage d'une image
<i>imagesc</i>	Affichage d'une image avec son intensité
<i>imread</i>	Lecture d'un fichier image à un format donné
<i>imwrite</i>	Ecriture d'un fichier image à un format donné
<i>iminfo</i>	lecture des informations d'un fichier image

La commande *image* restitue la matrice d'entrée sous forme d'une carte 2D avec des niveaux de gris correspondants. La matrice d'entrée doit donc être normalisée par rapport aux  $m$  niveaux de couleurs de la carte sélectionnée. En réalité, cette fonction est un cas particulier de la commande *pcolor* qui peut être totalement paramétrable dans la définition de ces axes, la gestion des couleurs, etc.

Si on désire utiliser une matrice non normalisée par rapport à la palette de couleurs, il faut travailler avec la commande *imagesc*.

#### 4.4.8 *colormap* et *colorbar*

La fonction *colormap* permet de sélectionner une palette de couleurs prédéfinie. Une palette de couleurs est un tableau de taille  $(m,3)$  dont les lignes spécifient l'intensité en code RGB (Red, Green, Blue) de la façon suivante:

$$map(k,:) = \begin{bmatrix} r(k) & g(k) & b(k) \end{bmatrix}$$

avec  $0 \leq r(k), g(k), b(k) \leq 1$

Exemple de codes :

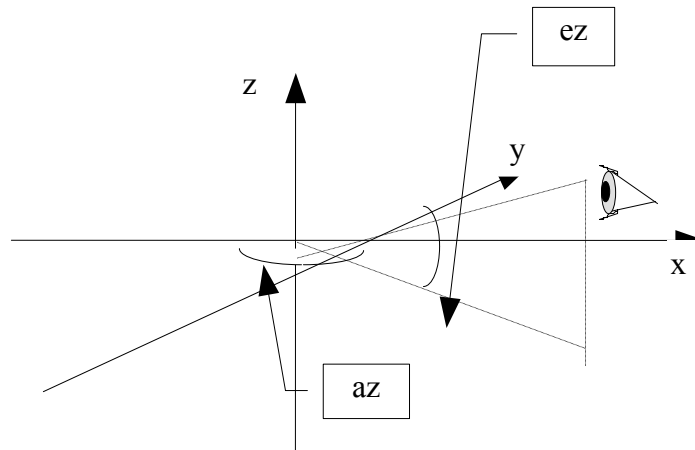
[0 0 0]	noir
[1 1 1]	blanc
[1 0 0]	rouge
[.5 1 .83]	bleu marine

Par défaut,  $m$  vaut 64. Il peut aller jusqu'à la valeur 256. *Matlab* propose plusieurs palettes de couleurs: *hsv*, *hot*, *cool*, *jt*, *pink*, *copper*, *gray*.

Pour associer les couleurs aux données, *Matlab* segmente l'axe des  $z$ , en utilisant  $[zmin, zmax]$ , en un nombre d'intervalles correspondant aux nombres de lignes de la palette de couleurs. Pour visualiser la palette avec les segments, on peut utiliser la commande *colorbar*.

#### 4.4.9 view

Lorsque l'on utilise une commande de visualisation 3D, la courbe apparaît sous un angle de vue donné. Grâce à la commande **view**, on peut faire varier les angles de point de vue. Le repère utilisé est le suivant:



**view(az,ae)**  
**view([xe ye ze])**

Par défaut, les arguments ont comme valeurs (0,90) pour des graphiques 2D et (-37.5,30) pour des graphiques 3D.

*Matlab* permet aussi de générer des graphiques de très haute qualité avec des effets d'éclairage, de texture etc. La commande **shading** offre la possibilité de faire disparaître la grille et **interp** permet de réaliser une interpolation afin de créer un effet de texture. La commande **light** permet de gérer la position d'éclairage de la figure. La commande **material** offre la possibilité de modifier le type de reflet de la courbe.

Exemple : un effet d'éclairage

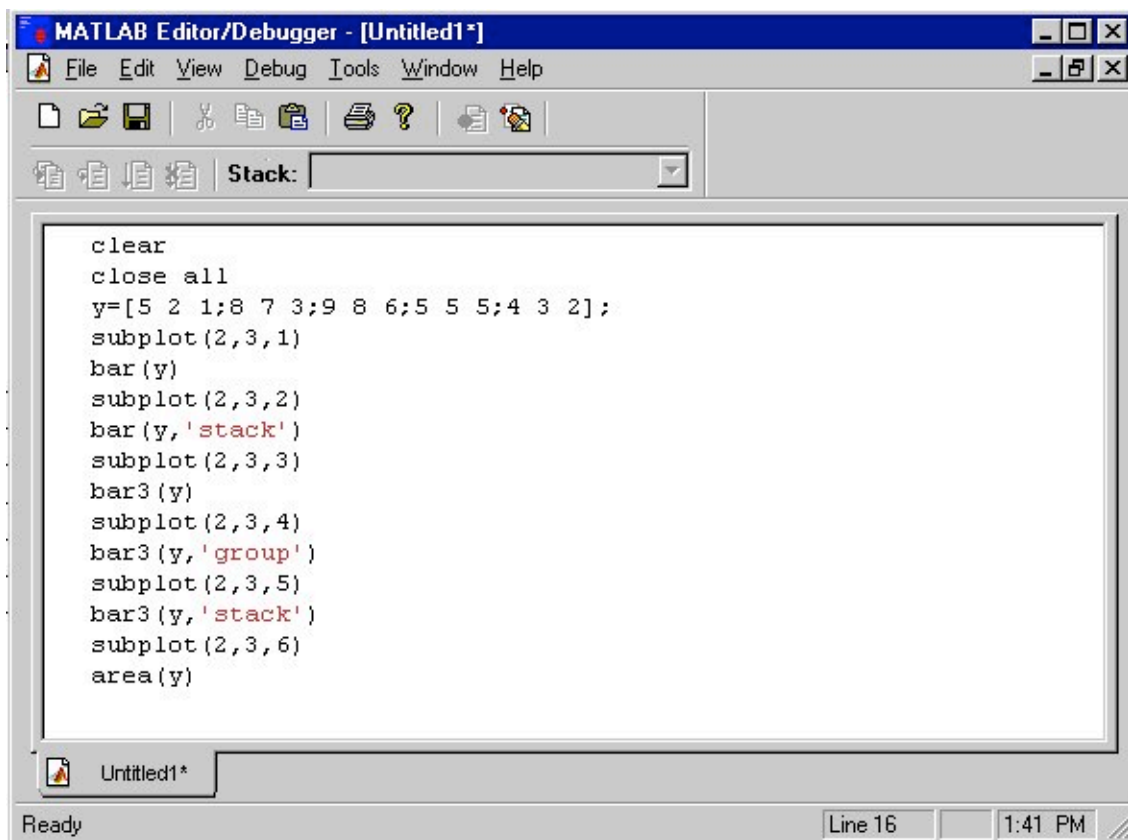
```
MATLAB Editor/Debugger - [Untitled1*]
File Edit View Debug Tools Window Help
close all
[x,y,z]=peaks(30);
figure(1)
surf(x,y,z)
shading interp
light('position',[0 -1 0])
figure(2)
surf(x,y,z)
shading interp
light('position',[-2 5 3])
Ready Line 10 1:39 PM
```

## 4.5 Graphes spécialisés

### 4.5.1 Barres et surfaces

fonctions	définition
<i>bar</i>	Barres verticales
<i>barh</i>	Barres horizontales
<i>bar3</i>	Barres verticales 3d
<i>bar3h</i>	Barres horizontales 3d
<i>area</i>	Surfaces

Exemple :



```
clear
close all
y=[5 2 1;8 7 3;9 8 6;5 5 5;4 3 2];
subplot(2,3,1)
bar(y)
subplot(2,3,2)
bar(y,'stack')
subplot(2,3,3)
bar3(y)
subplot(2,3,4)
bar3(y,'group')
subplot(2,3,5)
bar3(y,'stack')
subplot(2,3,6)
area(y)
```

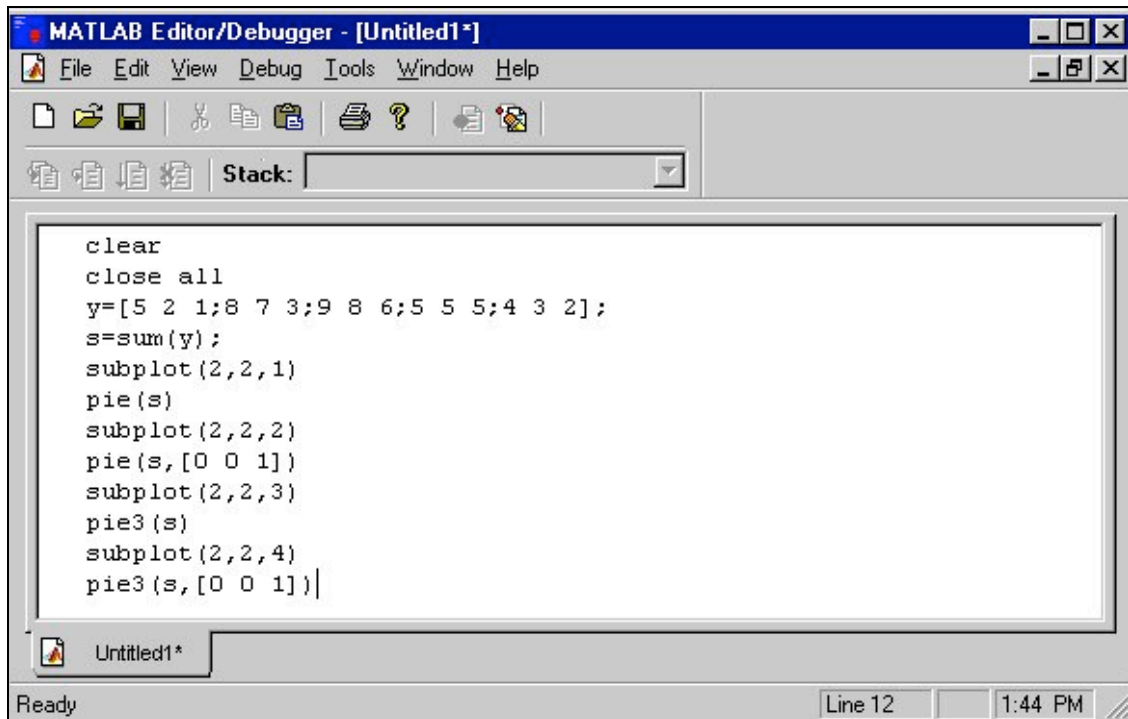
Figure n°14 : exemple d'utilisation de graphes spécialisés

### 4.5.2 Camemberts

fonctions	définition
<i>pie</i>	Camembert
<i>pie3</i>	Camembert 3d



Exemple :



*Figure n°15 : autre exemple d'utilisation de graphes spécialisés*

#### **4.6 Exercice : graphisme et fonctions**

On propose de réaliser l'étude du polynôme suivant :

$$p(x) = x^3 - 30x + 30$$

1) Evaluer le polynôme sur le segment  $[-8,8]$  pour un pas de calcul de 0.1.

➔ ***polyval***

2) Visualiser la courbe ainsi définie (Utiliser la notion de boucle interprétée).

➔ ***plot,grid***

3) Calculer les racines et les afficher sur le même graphique que précédemment.

➔ ***roots, hold, zeros, length***

4) Créer une fonction ***mypoly.m*** représentant le calcul du polynôme pour un  $x$  donné.

5) Evaluer l'aire formée par la courbe entre les deux premières racines.

➔ ***quad***

6) Visualiser graphiquement l'aire calculée

➔ ***fill***

#### 4.7 Autre exercice

On se propose d'identifier le polynôme d'ordre  $n$  qui va modéliser la courbe suivante :

$$y(x) = \sin(x) + \sin(3.2x) \text{ pour } x \in [0, \pi]$$

Générer un jeu de données avec un pas= $\pi/256$ .

A l'aide de commande ***polyfit***, calculer les coefficients du polynôme qui modélise la fonction  $y(x)$  au sens des moindres carrés. Calculer à l'aide de ***polyval*** la courbe obtenue et visualiser la ainsi que  $y(x)$ .

Calculer la norme de l'erreur de modélisation pour toutes les valeurs de l'ordre comprises entre 1 et 10. Visualiser la courbe obtenue.

$$E(x) = y(x) - p(x) \text{ avec } p(x) = a_1x^n + a_2x^{n-1} + \dots + a_nx^1 + a_{n+1}$$

*Matlab* possède deux types de fichiers. Le premier type est constitué par les fichiers créés à un format propriétaire à *Matlab*. Ces fichiers sont reconnaissables par leur extension en *.mat*. Ce format spécifique permet de sauvegarder des variables au format numérique ou texte, des variables réelles ou complexes très simplement. Le second type regroupe les fichiers binaires et ASCII communs à tous les langages de programmation. Ces derniers permettent donc de communiquer avec l'extérieur et en particulier de rapatrier des données collectées par d'autres logiciels.

### **5.1 Fichiers Matlab**

Les fichiers *nom\_fic.mat* constituent donc le système de fichiers propres à *Matlab*. Leur manipulation est très facile, quel que soit le type ou la taille des matrices que l'on désire sauvegarder.

#### **5.1.1 Commande *save***

Cette commande permet de créer ce premier type de fichier. La syntaxe est la suivante :

*save nom\_fich var1 var2 ...*

Les variables sont sauvegardées comme des matrices et donc se retrouvent au même format lorsqu'on les recharge afin de réutiliser.

Remarque: on peut sauvegarder ces variables au format ASCII (8 ou 16 bits)

**save** PATH/nom\_fich var1 var2 ...-ascii

#### **5.1.2 Commande *load***

Cette commande permet la relecture des variables sauvegardées avec la commande *save*.

*load* PATH/nom\_fich

Remarque: La commande *load* offre aussi la lecture de fichier ASCII sous forme de matrice

*load* PATH/nom\_fich.txt

## 5.2 Fichiers binaires et ASCII: le format standard

Ce type de fichier permet d'importer et d'exporter des données vers l'extérieur dans le format de son choix. Les commandes nécessaires à leur manipulation sont les suivantes :

<i>fopen</i>	<i>fseek</i>	<i>sprintf</i>
<i>fread</i>	<i>ferror</i>	<i>fscanf</i>
<i>fwrite</i>	<i>ftell</i>	<i>sscanf</i>
<i>fclose</i>	<i>fprintf</i>	

### 5.2.1 Commande *fopen*

Cette commande ouvre et donne des informations sur le fichier que l'on désire manipuler. Il y a deux types de syntaxes :

```
fid = fopen('filename')  
fid = fopen('filename','permission')  
[fid,message] = fopen('filename','permission','architecture')  
[filename,permission,architecture] = fopen(fid)
```

Ces trois premières commandes correspondent toutes à l'ouverture d'un fichier dans le mode spécifié par l'argument *permission*.

- r* : Ouverture en lecture.
- r+*: Ouverture en lecture et en écriture.
- w*: Efface le contenu ou crée un nouveau fichier et le place en mode écriture.
- w+*: Efface le contenu ou crée un nouveau fichier et le place en mode écriture/lecture.
- a*: Crée un nouveau fichier ou ouvre un fichier existant pour l'écriture à la suite des données déjà présentes.
- a+*: idem mais en lecture/écriture.

Si la commande a réussi, la variable *fid* devient un entier > 2 et la variable *message* est vide.

Si la commande n'a pas réussi, la variable *fid* prend la valeur -1. Dans ce cas, un texte a été associé à la variable *message* qui peut vous aider à définir l'erreur de manipulation.

## 5.2.2 Commande *fread*

Elle permet la lecture d'un fichier de données binaires. Sa syntaxe est la suivante:

$$[A, count] = fread(fid, size, 'precision')$$

La commande *fread* permet donc la lecture d'un fichier binaire spécifié par *fid* et écrit les données dans une matrice *A*. *fread* réalise la lecture d'un nombre d'éléments spécifié par *size* suivant une précision donnée.

Si l'argument *size* est donné, les entrées possibles sont:

*n* : lit *n* éléments vers un vecteur colonne.

*inf* : lit toutes les données et crée un vecteur de même taille (par défaut).

*[m,n]* lit (*m\*n*) éléments et crée une matrice de taille *[m,n]* suivant l'ordre des colonnes. Si la taille du fichier est inférieure, le reste se trouve comblé par des zéros.

L'argument *precision* correspond à la précision numérique utilisée pour sauvegarder les données. Il correspond au nombre d'octets alloué à la variable pour coder sa valeur. Un standard de précision indépendant de la machine source est le *float32* et *float64*. Ce format permet une bonne portabilité.

*Matlab* Langage C et Fortran

'schar'	'signed char'	Signed character; 8 bits
'uchar'	'unsigned char'	Unsigned character; 8 bits
'int8'	'integer*1'	Integer; 8 bits
'int16'	'integer*2'	Integer; 16 bits
'int32'	'integer*4'	Integer; 32 bits
'int64'	'integer*8'	Integer; 64 bits
'uint8'	'integer*1'	Unsigned integer; 8 bits
'uint16'	'integer*2'	Unsigned integer; 16 bits
'uint32'	'integer*4'	Unsigned integer; 32 bits
'uint64'	'integer*8'	Unsigned integer; 64 bits
'float32'	'real*4'	Floating-point; 32 bits
'float64'	'real*8'	Floating-point; 64 bits
'double'	'real*8'	Floating-point; 64 bits

formats portables

'char'	'char*1'	Character; 8 bits
'short'	'short'	Integer; 16 bits
'int'	'int'	integer; 32 bits
'long'	'long'	Integer; 32 or 64 bits
'ushort'	'unsigned short'	Unsigned integer; 16 bits
'uint'	'unsigned int'	Unsigned integer; 32 bits
'ulong'	'unsigned long'	Unsigned integer; 32 or 64 bits
'float'	'float'	Floating-point; 32 bits

### 5.2.3 Commande *fwrite*

*fwrite* permet l'écriture de données contenues dans une matrice. Sa syntaxe est la suivante:

$$count = fwrite(fid,A,'precision')$$

Elle permet donc de passer des données contenues dans la matrice A (sauvegarde dans le sens des colonnes) vers un fichier défini par *fid* suivant une précision donnée. *Count* correspond au nombre de données effectivement translitées.

### 5.2.4 Commande *fclose*

Cette commande correspond à la fermeture d'un ou de plusieurs fichiers. Sa syntaxe est:

$$status = fclose(fid)$$
$$status = fclose('all')$$

## 5.3 Exercice

1) Définissez une matrice A:

1.1	1.2	1.3
2.1	2.2	2.3
3.1	3.2	3.3
4.1	4.2	4.3

Enregistrez cette matrice A dans un fichier 'test.mat' au format Matlab (*save*)

Effacez l'ensemble des variables (*clear all*)

Chargez le fichier (*load*)

Vérifiez que la matrice A a bien été lue à partir du fichier.

2) Même question en enregistrant la matrice dans un fichier 'test.txt' au format ASCII. Vérifiez à l'aide du bloc note que le fichier contient bien les valeurs de la matrice A de façon lisible.

3) Même question en enregistrant la matrice dans un fichier 'test.raw' au format binaire, en utilisant *fopen*, *fwrite*, *fread* et *fclose*. On utilisera par exemple une précision 'double'.

La fonction *fread* ne renvoie pas une matrice, mais un vecteur; pourquoi ? Quelle fonction matlab utiliser pour redonner sa "forme" 4x3 à la matrice ?

**6.1 Rappels sur le développement en série de Fourier et Transformée de Fourier**

Tout signal  $v(t)$  périodique et de période  $T$  peut se décomposer en série de Fourier. Ainsi, en prenant

$$a_k = \frac{2}{T} \int_{-T/2}^{T/2} v_s(t) \cos\left(\frac{2k\pi}{T}t\right) dt$$

$$b_k = \frac{2}{T} \int_{-T/2}^{T/2} v_s(t) \sin\left(\frac{2k\pi}{T}t\right) dt$$

la décomposition en séries de Fourier de  $v(t)$  est alors

$$v(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} a_k \cos\left(\frac{2k\pi}{T}t\right) + b_k \sin\left(\frac{2k\pi}{T}t\right).$$

Une autre écriture est souvent utilisée, surtout en traitement du signal : il s'agit du développement en série de Fourier en termes complexes.

$$v(t) = \sum_{k=-\infty}^{\infty} c_k \exp\left(jk \frac{2\pi}{T}t\right)$$

avec

$$c_k = \frac{1}{T} \int_{-T/2}^{T/2} v_s(t) \exp\left(-jk \frac{2\pi}{T}t\right) dt.$$

Les coefficients  $c_k$  et  $c_{-k}$  sont complexes conjugués et sont liés aux quantités  $a_k$  et  $b_k$  comme suit :

$$c_k = \frac{1}{2}(a_k - jb_k) \text{ et } c_{-k} = \frac{1}{2}(a_k + jb_k)$$

Il est à noter que  $|c_k| = |c_{-k}|$  et que  $c_k = |c_k| \exp(j\phi_k)$  avec  $\phi_k = -\text{Arc tan}\left(\frac{b_k}{a_k}\right)$ .

On peut aussi exprimer le signal  $v(t)$  à partir de  $\phi_k$ . On a

$$v(t) = c_0 + \sum_{k=1}^{\infty} 2|c_k| \cos\left(\frac{2k\pi}{T}t + \phi_k\right).$$

Remarque : Le signal périodique n'est pas à énergie finie sur l'intervalle  $]-\infty; +\infty[$ . Cela revient à dire que la quantité

$$\int_{-\infty}^{+\infty} v^2(t) dt$$

n'a pas une valeur finie.  $v(t)$  n'est donc pas de carré intégrable.

Remarque 2 : On constate en outre que d'après l'**égalité de Parseval**,

$$\sum_{k=1}^{\infty} |c_k|^2 = \frac{1}{T} \int_T |v(t)|^2 dt$$

Si  $v(t)$  est réel,  $\sum_{k=1}^{\infty} |c_k|^2 = \frac{1}{T} \int_T v^2(t) dt$ . La puissance moyenne totale du signal est donc égale à la somme des puissances moyennes des différents harmoniques et de la composante continue.

Remarque 3 : On rappelle que la valeur moyenne d'un signal périodique est donnée par

$$\mu = \frac{1}{T} \int_0^T v(t) dt.$$

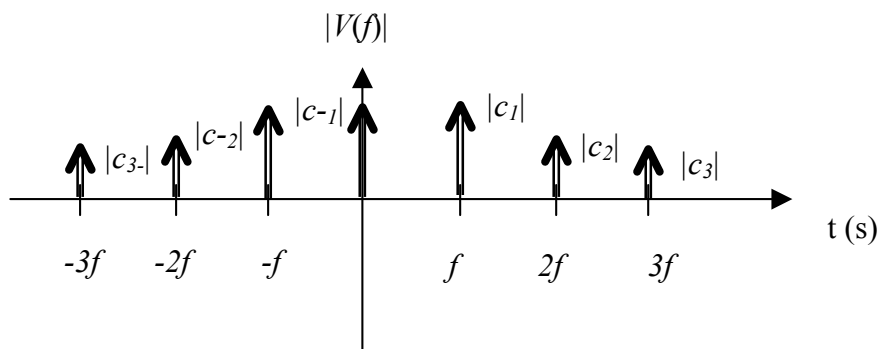


Figure n°16 : spectre d'amplitude du signal périodique

Le spectre du signal périodique  $v(t)$  a une représentation spectrale discrète.



## 6.2 Exercice

### 6.2.1 Décomposition en série de Fourier, représentation spectrale et somme de série

Soit  $y(t)$  la fonction de période  $2\pi$  définie sur  $[-\pi; \pi]$  par :

$$y(t) = \begin{cases} 0 & \text{si } -\pi \leq t \leq 0 \\ \sin(t) & \text{si } 0 \leq t \leq \pi \end{cases}$$

- Développer cette fonction en série de Fourier à partir des fonctions sinus et cosinus ;
- Représenter la somme du fondamental avec les 5, puis 10 puis 30 premiers harmoniques. On créera pour cela une fonction.
- Vérifier la formule  $\sum_{p=1}^{+\infty} \frac{1}{4p^2 - 1} = \frac{1}{2}$ . On utilisera *Matlab* pour visualiser l'évolution de la

fonction  $g(x) = \sum_{p=1}^x \frac{1}{4p^2 - 1}$ .