

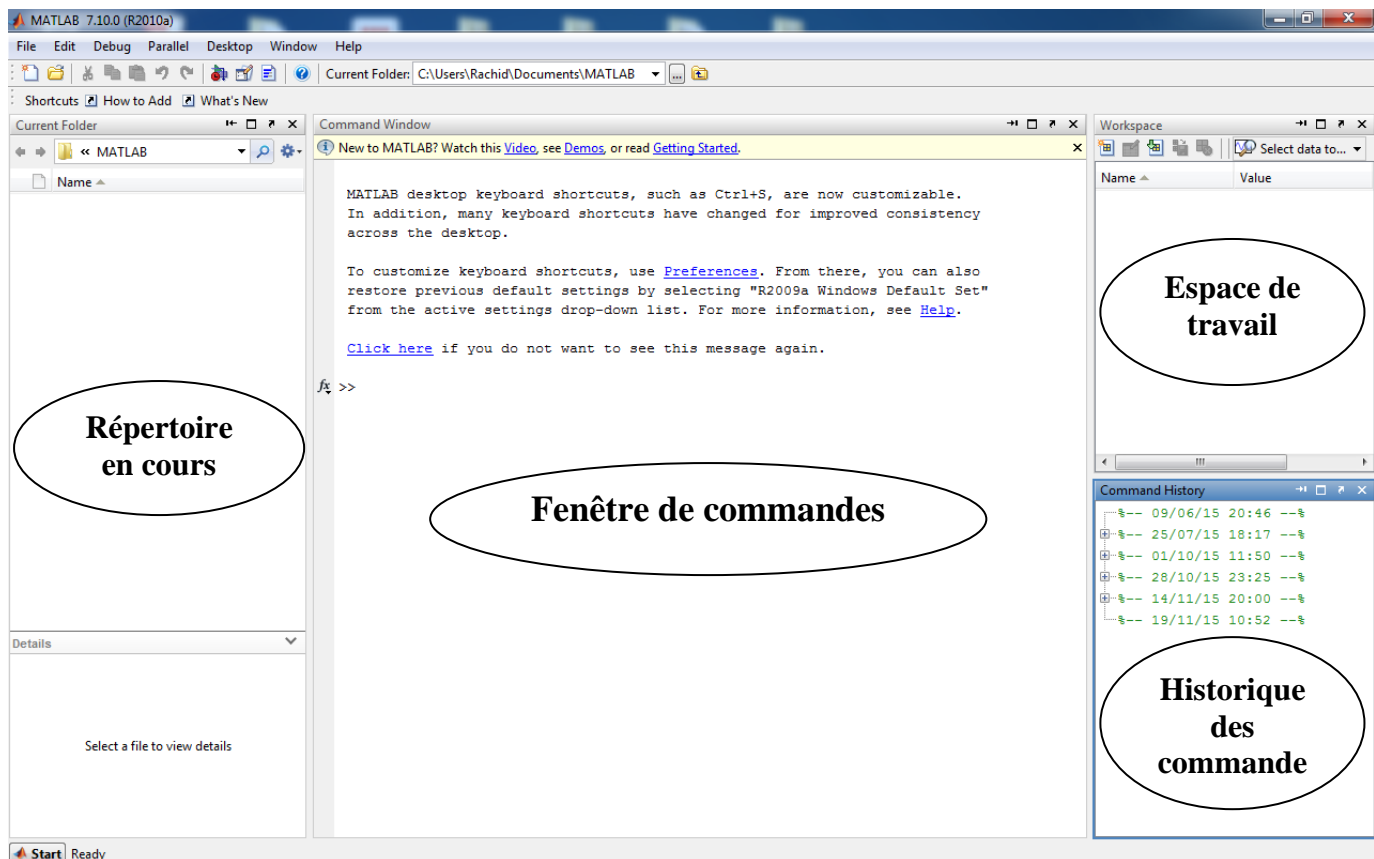
## Introduction à Matlab

### I. Présentation de l'interface du logiciel

L'interface de Matlab est divisée en quatre parties principales soit :

- La fenêtre de commandes (*Command Window*);
- L'espace de travail (*Workspace*);
- Le répertoire en cours (*Current Directory*);
- L'historique des commandes (*Command History*).

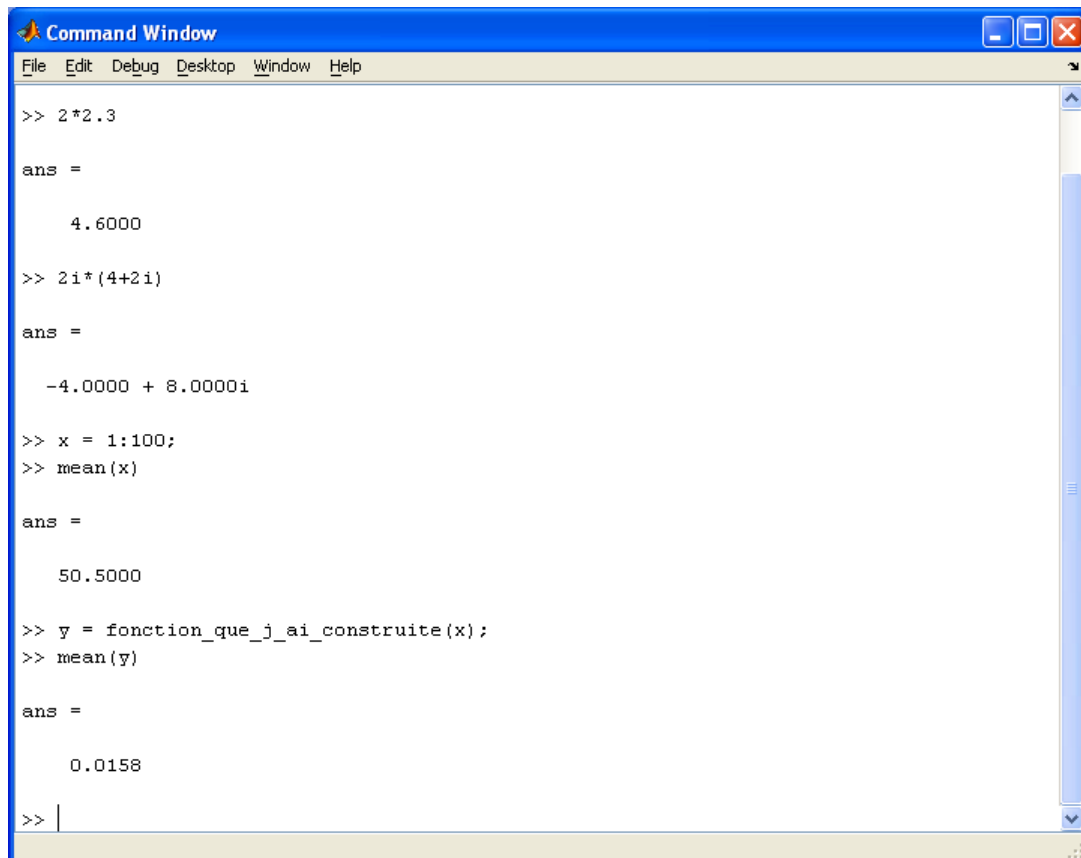
Ces parties sont représentées sur l'image ci-dessous.



Nous allons maintenant voir l'utilité ainsi que les principales fonctionnalités de chacune de ces fenêtres d'affichage.

#### Fenêtre de commandes

C'est la fenêtre dans laquelle on dicte les opérations à effectuer un peu comme l'ancien DOS où l'on exécutait des programmes ou des opérations à partir de lignes commandes. En utilisant la syntaxe appropriée, on peut se servir de la fenêtre de commandes pour effectuer des opérations de calcul ou pour appeler des fonctions provenant des bibliothèques de Matlab ou des fonctions que l'on a nous même construites comme on peut voir sur l'image ci-dessous. On peut également demander de l'aide en utilisant la commande *help* soit seul ou suivi du nom de la fonction dont on veut de l'information (exemple : *help mean*) et on peut accéder à certaines fonctionnalités de Matlab telles que Simulink et GUIDE. En bref, la fenêtre de commandes c'est ce qui fait le lien entre l'utilisateur et le programme.



```
Command Window
File Edit Debug Desktop Window Help

>> 2*2.3

ans =

    4.6000

>> 2i*(4+2i)

ans =

   -4.0000 + 8.0000i

>> x = 1:100;
>> mean(x)

ans =

    50.5000

>> y = fonction_que_j_ai_construite(x);
>> mean(y)

ans =

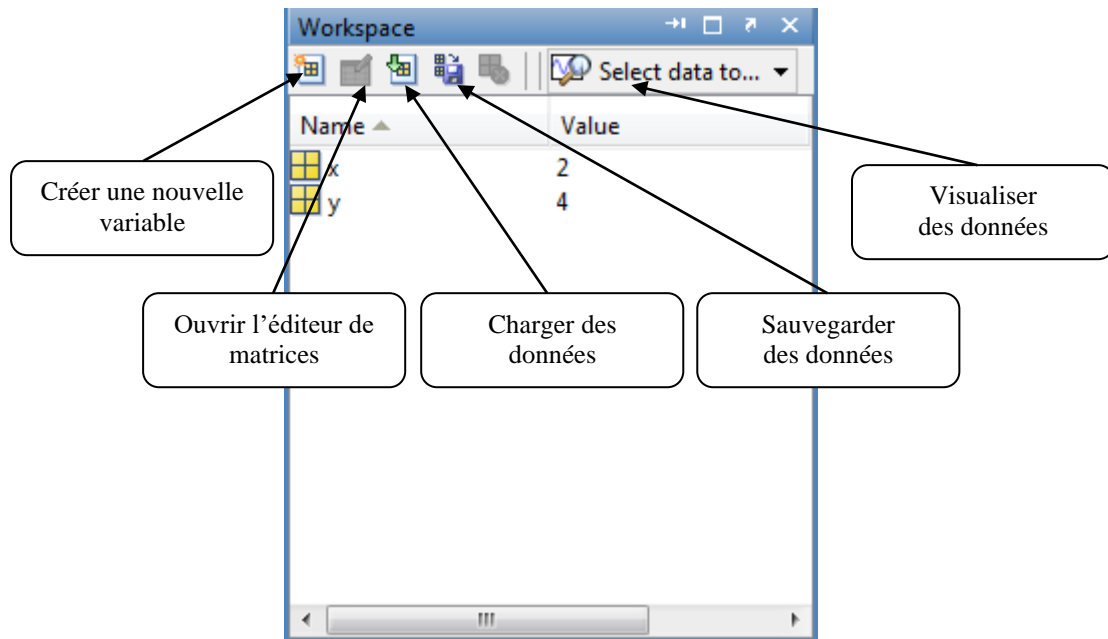
    0.0158

>> |
```

## Espace de travail

On trouve dans cette fenêtre toutes les variables qui ont été créées et qui sont utilisables dans la fenêtre de commandes pour des opérations de calcul et pour faire de la visualisation. Ces variables sont regroupées sous forme de classes qui indiquent la nature de la variable. Il existe différents types de variables qui peuvent être définies. Dans ce cours, vous aurez à utiliser seulement des variables de type *double* (i.e. une variable qui a une précision double par rapport à une variable de type *float* mais qui occupe deux fois plus de mémoire soit 64 bits). D'autre part, on retrouve également la valeur de la variable sous la colonne *value* si sa matrice contient seulement un élément, sinon on retrouve inscrite dans cette colonne la taille de la matrice représentative de la variable.

La fenêtre d'affichage de l'espace de travail contient une barre d'outils vous permettant d'effectuer plusieurs opérations sur vos données très rapidement. Les principales opérations sont montrées sur l'image ci-dessous.

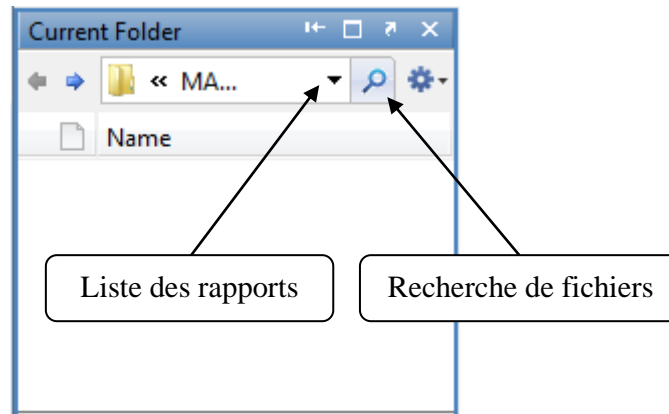


Dans un premier temps, on peut créer de nouvelles variables que l'on peut définir par la suite à l'aide de la fenêtre de commandes. On peut également ouvrir l'éditeur de matrices qui nous permet de visualiser les données sous forme de tableau. Des données provenant de Microsoft Excel peuvent être facilement importées dans l'éditeur de matrices en utilisant les fonctionnalités de Windows copier et coller.

On peut aussi effacer ou ajouter manuellement des données dans l'éditeur. D'autre part, on peut charger ou sauvegarder des données à partir de la barre d'outils de la fenêtre d'affichage de l'espace de travail. Ces données sont compilées dans un fichier ayant une extension \*.mat et peuvent être placées dans le répertoire de votre choix. Par contre, pour charger ces données, vous devez absolument spécifier le répertoire dans lequel elles se trouvent. Le répertoire en cours peut être spécifié dans la barre d'outils principale de Matlab à l'endroit marqué *Current Directory*. Finalement, les données peuvent être visualisées graphiquement en appuyant sur le bouton graphique Select data to... ▼.

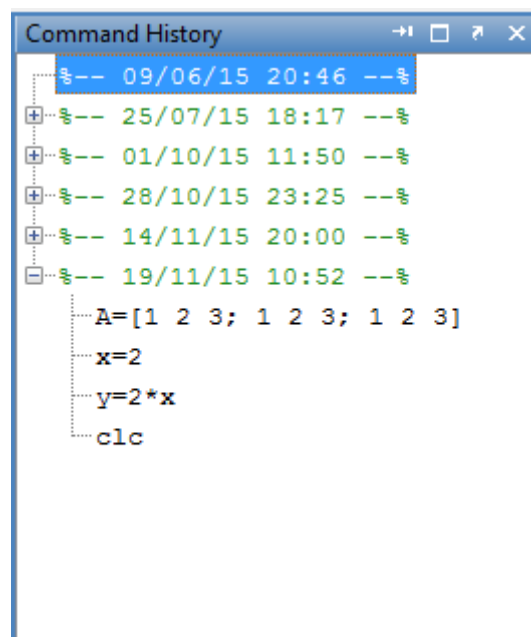
### Répertoire en cours

Le répertoire en cours permet de visualiser des fichiers se trouvant dans le même répertoire. Tous les fichiers peu importe leur extension sont affichées. On peut ouvrir un fichier en cliquant directement dessus. Les principaux fichiers qui peuvent être lus par Matlab sont les fichiers de fonctions ou de programmes (\*.m), les fichiers de données (\*.mat) et les fichiers des graphiques (\*.fig). De plus, dans la barre d'outils de la fenêtre d'affichage du répertoire en cours, on peut créer des rapports qui nous documentent sur les fichiers se trouvant dans le répertoire. Parmi les plus importants, il y a le *M-Lint Code Check Report* qui va répertorier les erreurs et les avertissements dans le code d'un programme ou d'une fonction. Il y a également le *File Comparison Report* qui repère les différences au niveau du code entre deux documents.



### Historique des commandes

Cette fenêtre contient les commandes utilisées précédemment dans la fenêtre de commandes. L'historique des commandes peut compiler plusieurs centaines de lignes de commandes. À partir de cette fenêtre, on peut copier et coller des lignes de commandes pour les réutiliser dans la fenêtre de commandes. D'autre part, à partir de la fenêtre de commandes, on peut retrouver des lignes de commandes utilisées précédemment en appuyant sur la flèche du haut se trouvant sur votre clavier. L'image ci-dessous montre cette fenêtre d'affichage.



## II. L'espace de travail

Comme tout langage de programmation matlab permet de définir des données variables. Les variables sont définies au fur et à mesure que l'on donne leurs noms (identificateur) et leurs valeurs numériques ou leurs expressions mathématiques. matlab ne nécessite pas de déclaration de type ou de dimension pour une variable. Les variables sont stockées dans l'espace de travail (ou workspace) et peuvent être utilisées dans les calculs subséquents. Pour obtenir la liste des variables actives de l'espace de travail on dispose des commandes `who` et `whos`. La commande `who` affiche le nom des variables actives. La commande `whos` donne plus d'informations : le nom, la taille du tableau (nombre de lignes et de colonnes) associé, l'espace mémoire utilisé (en Bytes) et la classe des données (principalement double array s'il

s'agit d'un tableau de valeurs réelles ou complexes et char s'il s'agit d'un tableau de caractères). La commande `clear` permet de nettoyer l'espace de travail : toutes les variables sont détruites. Il est possible de ne détruire qu'une partie des variables en tapant `clear nom-var` où `nom-var` est le nom de la (ou des) variable(s) à détruire.

Le nombre de fonctions de matlab étant énorme, vous devrez utiliser l'aide quasiment en permanence. La commande ***help*** vous donne un aperçu des commandes disponibles :

```
>> help
```

HELP topics:

matlab\general	- General purpose commands.
matlab\ops	- Operators and special characters.
matlab\lang	- Programming language constructs.
matlab\elmat	- Elementary matrices and matrix manipulation.
matlab\randfun	- Random matrices and random streams.
matlab\elfun	- Elementary math functions.
matlab\specfun	- Specialized math functions.
matlab\matfun	- Matrix functions - numerical linear algebra.
matlab\datafun	- Data analysis and Fourier transforms.
matlab\polyfun	- Interpolation and polynomials.
matlab\funfun	- Function functions and ODE solvers.
matlab\sparsfun	- Sparse matrices.
matlab\scribe	- Annotation and Plot Editing.
matlab\graph2d	- Two dimensional graphs.
matlab\graph3d	- Three dimensional graphs.

...

Pour obtenir les informations concernant une section particulières, entrez ***help section*** :

```
>> help elfun
```

Elementary math functions.

Trigonometric.

sin	- Sine.
sind	- Sine of argument in degrees.
sinh	- Hyperbolic sine.
asin	- Inverse sine.
asind	- Inverse sine, result in degrees.
asinh	- Inverse hyperbolic sine.
cos	- Cosine.
cosd	- Cosine of argument in degrees.
cosh	- Hyperbolic cosine.
acos	- Inverse cosine.

...

Pour avoir de l'aide directement sur une commande, entrez ***help commande*** :

```
>> help sin
```

SIN Sine of argument in radians.

SIN(X) is the sine of the elements of X.

See also asin, sind.

Overloaded methods:  
codistributed/sin

Reference page in Help browser  
doc sin

### III. Les bases de MATLAB

#### 1. Nombres et variables

Comme mentionné précédemment, puisque Matlab est un logiciel de calcul numérique, chaque variable doit être définie pour être utilisée. Nous définissons une variable dans la fenêtre de commandes en lui donnant un nom suivi du symbole = et en lui assignant une valeur comme l'exemple ci-dessous :

$$x = 2.25$$

Si nous ne souhaitons pas voir le résultat apparaître dans la fenêtre de commandes, nous devons ajouter un point-virgule (;) à la fin de l'expression. Le point-virgule peut se placer à la fin de chaque expression représentée sous forme d'équation. Lorsque nous définissons une variable comme l'exemple ci-haut, Matlab crée une matrice de grandeur conforme à ce que nous lui avons soumis (dans l'exemple précédent cela correspond à une matrice 1 x 1). Une autre particularité du logiciel est que tout nombre soumis au logiciel est considéré comme étant un nombre complexe même si nous n'avons pas spécifié de partie imaginaire à notre nombre. Le nombre imaginaire  $\sqrt{-1}$  peut être représenté dans Matlab soit par la lettre i ou j. C'est au choix. Il n'y a aucun besoin de définir i ou j dans ces cas là puisque Matlab reconnaît ces variables comme étant complexes. Il en est de même pour certains nombres réels comme pi.

Nous avons vu comment définir une variable pour une valeur scalaire. Maintenant nous allons voir comment définir des matrices et des vecteurs. Pour définir une matrice ou un vecteur, il faut utiliser des crochets ([]). Il faut premièrement entrer les éléments d'une même rangée en les séparant par des espaces marquant le changement de colonne. Ensuite pour changer de rangée, il faut utiliser le point-virgule. L'exemple ci-dessous vous montre comment entrer une matrice identité 3x3.



```
Command Window
>> I=[1 0 0;0 1 0;0 0 1]

I =

     1     0     0
     0     1     0
     0     0     1

>>
```

Pour les vecteurs, la procédure est identique à celle pour les matrices exceptées que nous utilisons juste une rangée ou une colonne. Pour visualiser, créer, modifier ou supprimer un ou

plusieurs éléments d'une matrice ou d'un vecteur déjà créé, il faut utiliser les indexes de la variable. Un élément d'une matrice se définit comme suit :

`Variable(# rangée, # colonne)`

Par exemple, dans le cas de la matrice identité créée précédemment les éléments `I(1,1)` et `I(3,2)` correspondraient à 1 et 0 respectivement. Pour identifier tous les éléments d'une rangée ou d'une colonne, nous devons remplacer l'index correspondant par les deux-points (`:`) (dans l'exemple précédent `I(:,1)` correspondrait à la première colonne soit `[1;0;0]`).

## 2. Opérateurs arithmétiques

Nous avons vu comment définir des quantités scalaires et des matrices. Nous allons apprendre comment effectuer des opérations arithmétiques entre ces variables.

La liste suivante énumère les principaux opérateurs à utiliser

<i>Opérateurs</i>	<i>Fonctions</i>
<code>+</code>	Addition
<code>-</code>	Soustraction
<code>*</code>	Multiplication matricielle
<code>.*</code>	Multiplication des matrices élément par élément
<code>/</code>	Division droite des matrices
<code>./</code>	Division des matrices élément par élément
<code>^</code>	Mise en puissance
<code>.^</code>	Mise en puissance élément par élément
<code>'</code>	Transposée

Lorsque l'on effectue des opérations matricielles, il faut s'assurer que le format des matrices utilisées correspond à l'opération effectuée. L'image ci-dessous montre la différence entre l'opérateur (`^`) et (`.^`).

```
x =
     1     3     2
     0     3     7
     4     1     2

>> x^2

ans =
     9    14    27
    28    16    35
    12    17    19

>> x.^2

ans =
     1     9     4
     0     9    49
    16     1     4
```

### 3. Opérateurs relationnels et logiques

Les opérateurs relationnels et logiques sont utilisés dans les directives de contrôle notamment dans les boucles *while* et dans les directives *if*. On peut également s'en servir pour de simples comparaisons. Comme par exemple, si nous voulons savoir quels éléments d'une matrice sont plus grands que ceux d'une autre matrice, nous pouvons utiliser l'expression suivante :

```
x =
```

```
2  4  0
6  5  3
1  2  4
```

```
>> y=5*eye(3)
```

```
y =
```

```
5  0  0
0  5  0
0  0  5
```

```
>> y>x
```

```
ans =
```

```
1  0  0
0  0  0
0  0  1
```

La réponse nous est retournée sous forme de matrice. Les éléments qui valent 1 veulent dire que la comparaison est vraie pour cet index et ils valent 0 si la comparaison est fausse. Dans l'exemple ci-dessus la fonction *eye* est une fonction de Matlab qui définit une matrice identité.

La liste des opérateurs relationnels et logiques est présentée ci-dessous.

<i>Opérateurs</i>	<i>Fonctions</i>
<	Plus petit
>	Plus grand
<=	Plus petit ou égal
>=	Plus grand ou égal
==	Égalité (ne pas confondre avec = qui assigne)
~=	Pas égal
&	ET
	OU
~	Négation



## 4. Structures de contrôle

Maintenant que nous avons vu les principaux opérateurs arithmétiques relationnels et logiques, nous pouvons apprendre comment fonctionnent les boucles et les directives de contrôle. Nous allons voir dans cette section les boucles *for* et *while* ainsi que la directive *if* puisqu'elles sont largement utilisées en programmation.

La boucle *for* sert à effectuer les mêmes opérations pour plusieurs itérations. Nous pouvons nous en servir pour construire des matrices, des vecteurs des graphiques et plusieurs autres choses. Voici la syntaxe à employer lorsque nous voulons utiliser une boucle *for* :

```
for index = départ:incrément:fin
    expressions
end

% Exemple

x = []; % matrice vide
n = 10; % nombre d'itération
for i = 1:1:n % i = départ : incrément : fin
    x(i) = i^2 + 2; % expression
end
```

À chaque itération, les opérations sont effectuées jusqu'à la commande *end*. Rendu à ce point là, les opérations recommencent au début de la boucle avec une itération de plus. Ce processus recommence jusqu'à ce que *index* (*i* dans l'exemple) soit rendu à sa valeur finale. Dans l'exemple ci-dessus, la boucle *for* a servi à créer un vecteur *x* de dimension *n*.

La boucle *while* a comme objectif d'effectuer des opérations tant et aussi longtemps qu'une expression logique est vraie. Lorsque l'expression devient fausse, la boucle se ferme et le décodage de la fonction continue. La syntaxe à employer est la suivante :

```
while expression logique
    expressions
end

% Exemple

x = []; % matrice vide
n = 0; % variable à incrémenter
while n < 10 % expression logique
    x(n+1) = 2^n-10; % expression
    n = n+1; % itération
end
```

D'autre part, la directive de contrôle *if* sert à déterminer les opérations à effectuer dépendamment du résultat d'une condition spécifiée par l'utilisateur. Voici la syntaxe à employer dans le cas d'une directive *if* :

```

if expression logique
    expressions
elseif expression logique % si nécessaire
    expressions
else
    expressions
end

% Exemple

x = []; % matrice vide
i = 3.14; % variable à incrémenter
if i < pi % expression logique
    x = sin(rand(100,1)); % expression
else
    x = cos(rand(100,1)); % expression
end

```

L'expression logique mentionnée dans la syntaxe de la directive if est la condition qui doit être validée pour effectuer la série d'opération spécifiée sans quoi ce sera une autre série qui sera effectuée.

#### IV. Graphisme

Tout tracé avec Matlab, s'effectue dans une fenêtre graphique que l'on crée par la commande figure ou quand on exécute une commande de dessin (plot ...). On peut créer autant de fenêtres graphiques que l'on veut celles-ci étant numérotées de 1 à N au fur et à mesure de leur création. La fenêtre graphique par défaut et la dernière qui a été créée par figure ou la dernière activée par une commande de dessin ou sélectionnée avec la souris.

Figure % crée une fenêtre graphique qui devient la figure par défaut,

figure(n) % crée une fenêtre graphique numéro n qui devient la fenêtre active.

Fonctions

plot

t = 0:0.1:5;

x = 2\*sin(2\*pi\*t);

plot(t,x); % dessin de x en fonction de t, interpolation linéaire entre les points.

plot(t,x,'-') % idem avec style - - -

plot(t,x,'b-') % idem style --- couleur bleue

plot(t,x,'o') % idem pas d'interpolation, chaque point marqué par o

Un plot provoque l'effacement du dessin précédent (par défaut) on peut superposer des dessins en mettant le commutateur hold à on

hold on % désactivation par hold off

title('Titre de la figure');

xlabel('commentaire sur l'axe x');

ylabel('idem axe y');

axis([xmin,xmax,ymin,ymax]); % définit l'échelle des axes

legend('tracé 1','tracé 2',...); % chaque tracé est associé à une légende

grid % affiche une grille

`text(x,y,'texte')` % place texte à la position x y dans la fenêtre  
`gtext('texte')` % place texte à la position définie avec la souris

Une fenêtre graphique peut être subdivisée en plusieurs tracés,  
`subplot(n,p,q)` % subdivision en n\*q dessin et sélectionne à qième

## 1. Graphique 2D

Une courbe 2D est pour tout logiciel de tracé de courbes représenté par une série d'abscisses et une série d'ordonnées. Ensuite, le logiciel trace généralement des droites entre ces points. MATLAB n'échappe pas à la règle. La fonction s'appelle `plot`.

### L'instruction `plot` (Tracer une courbe simple)

L'utilisation la plus simple de l'instruction `plot` est la suivante.

```
plot( vecteur d'abscisses, vecteur d'ordonnées )  
[ x1 x2 ... xn ] [ y1 y2 ... yn ]
```

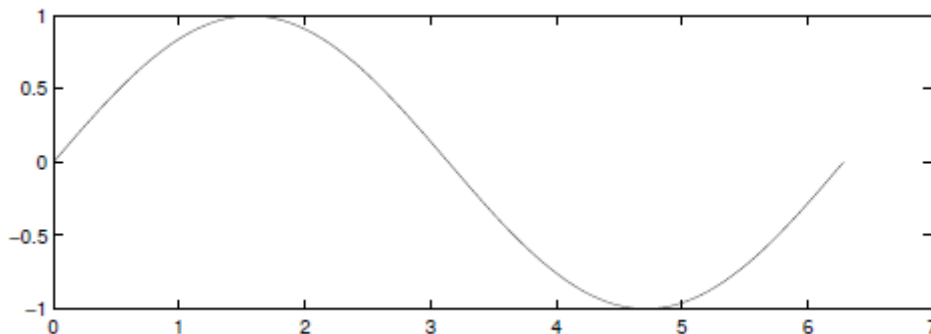
Les vecteurs peuvent être indifféremment ligne ou colonne, pourvu qu'ils soient tous deux de même type. En général ils sont lignes car la génération de listes de valeurs vue à la fin du chapitre précédent fournit par défaut des vecteurs lignes.

Par exemple, si on veut tracer  $\sin(x)$  sur l'intervalle  $[0, 2\pi]$ , on commence par définir une série (raisonnable, disons 100) de valeurs équidistantes sur cet intervalle :

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

puis, comme la fonction  $\sin$  peut s'appliquer terme à terme à un tableau:

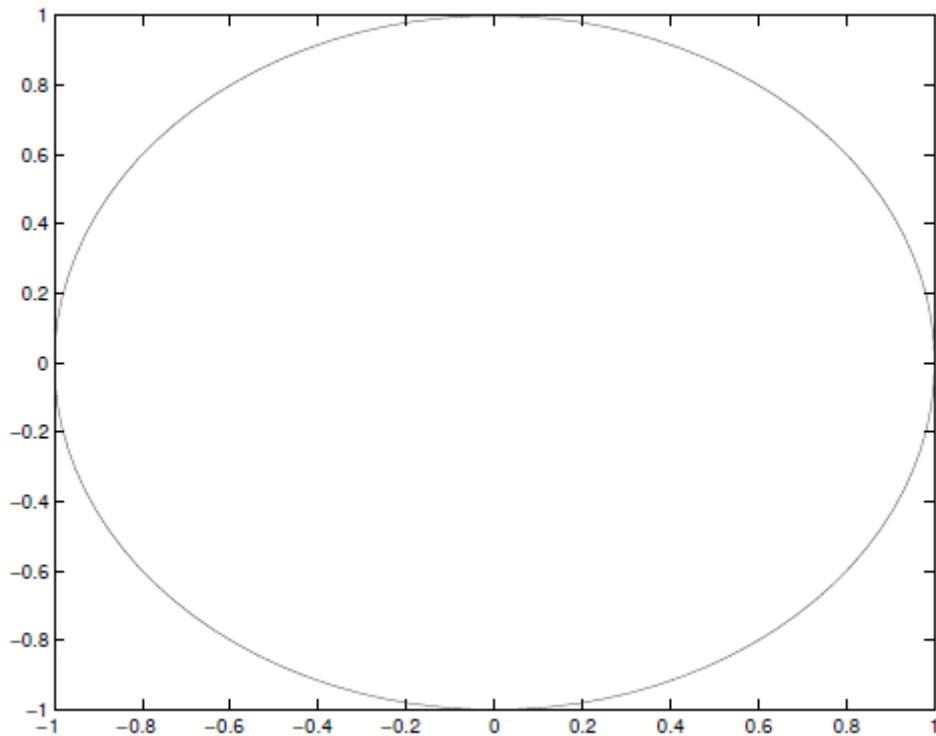
```
>> plot(x, sin(x))
```



On voit que les axes s'adaptent automatiquement aux valeurs extrémales des abscisses et ordonnées.

On remarquera que tout ce que demande `plot`, c'est un vecteur d'abscisses et un vecteur d'ordonnées. Les abscisses peuvent donc être une fonction de  $x$  plutôt que  $x$  lui-même. En d'autres termes, il est donc possible de tracer des courbes paramétrées :

```
>> plot(cos(x), sin(x))
```



### Remarque.

La commande `fplot` permet de tracer le graphe d'une fonction sur un intervalle donné.

La syntaxe est : `fplot('nomf', [xmin , xmax])` où `nomf` est le nom d'une fonction  
`[xmin , xmax]` est l'intervalle pour lequel est tracé le graphe de la fonction.

Il est possible de tracer plusieurs fonctions sur la même figure par cette commande.

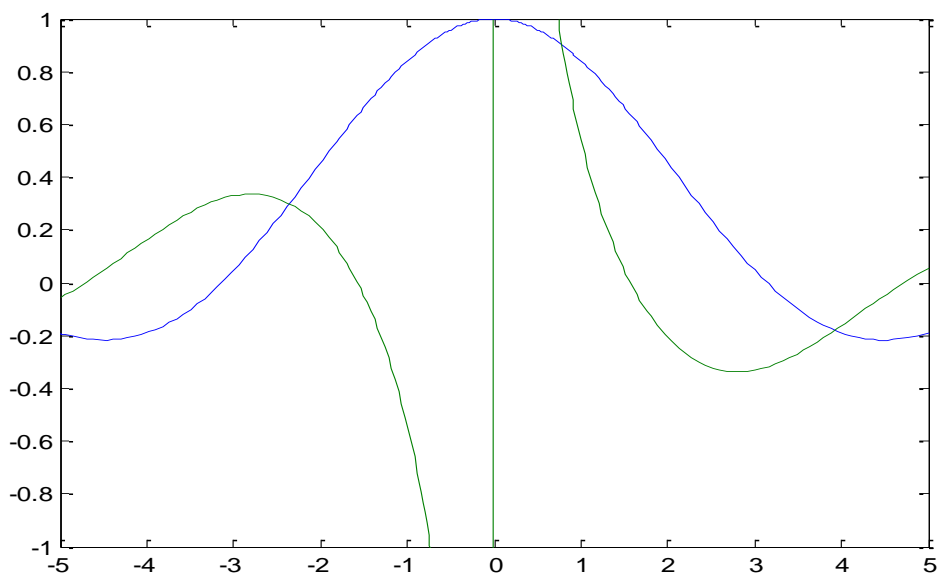
Il faut pour cela utiliser la commande `fplot` de la manière suivante :

`fplot('nom_f1,nom_f2,nom_f3',[x_min,x_max])`

Il est possible de gérer les bornes des valeurs en ordonnées en passant comme second argument de la commande `fplot` le tableau `[x_min , x_max , y_min , y_max ]`.

### Exemple.

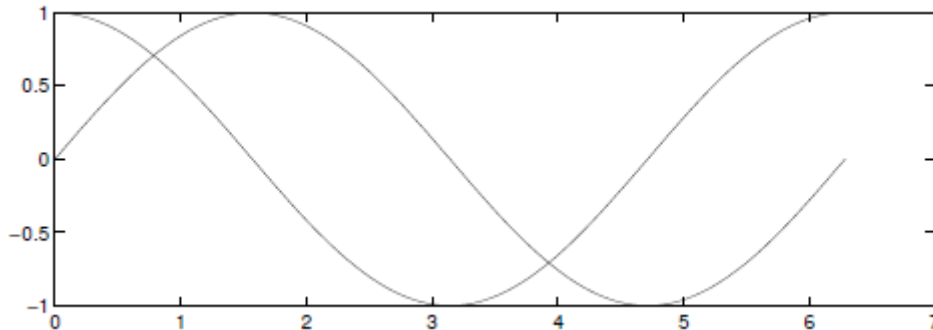
```
>> fplot('sin(x)/x, cos(x)/x',[-5,5,-1,1])
```



## 2. Superposer plusieurs courbes

Il suffit de spécifier autant de couples (abscisses, ordonnées) qu'il y a de courbes à tracer.  
Par exemple pour superposer sin et cos :

```
>> plot(x,cos(x),x,sin(x))
```



Les deux courbes étant en réalité dans des couleurs différentes. Cette méthode fonctionne même si les abscisses des deux courbes ne sont pas les mêmes.

Dans le cas plus fréquent où les abscisses sont les mêmes, il existe un autre moyen de superposer les courbes. On fournit toujours le vecteur des abscisses, commun aux deux courbes, et on fournit autant de vecteurs d'ordonnées qu'il y a de courbes. Tous ces vecteurs d'ordonnées sont regroupés dans un même tableau, chaque ligne du tableau représentant un vecteur d'ordonnées :

**plot** ( *vecteur d'abscisses*, *tableau d'ordonnées* )

$[ x_1 \ x_2 \ \dots \ x_n ]$	$\begin{bmatrix} y_1^1 & y_2^1 & \dots & y_n^1 \\ y_1^2 & y_2^2 & \dots & y_n^2 \\ \vdots & \vdots & \dots & \vdots \\ y_1^m & y_2^m & \dots & y_n^m \end{bmatrix}$	Première courbe
		Deuxième courbe
		m <sup>ème</sup> courbe

Par exemple, pour superposer sin et cos, on devra fournir à plot les arguments suivants :

```
plot ( [ x1 x2 ... xn ], [ cos(x1) cos(x2) ... cos(xn) ] )
                        [ sin(x1) sin(x2) ... sin(xn) ] )
```

Le deuxième tableau se construit très facilement avec le point-virgule

```
>> plot(x, [cos(x);sin(x)])
```

### Remarque :

Lorsque l'on écrit une expression arithmétique dans les arguments de plot, il faut utiliser systématiquement les opérateurs terme à terme

.\* ./ et .^ au lieu de \* / et ^

Essayez par exemple de tracer le graphe de la fonction  $x \sin(x)$  sur  $[0, 2\pi]$

```
>> x = 0:0.1:1;
```

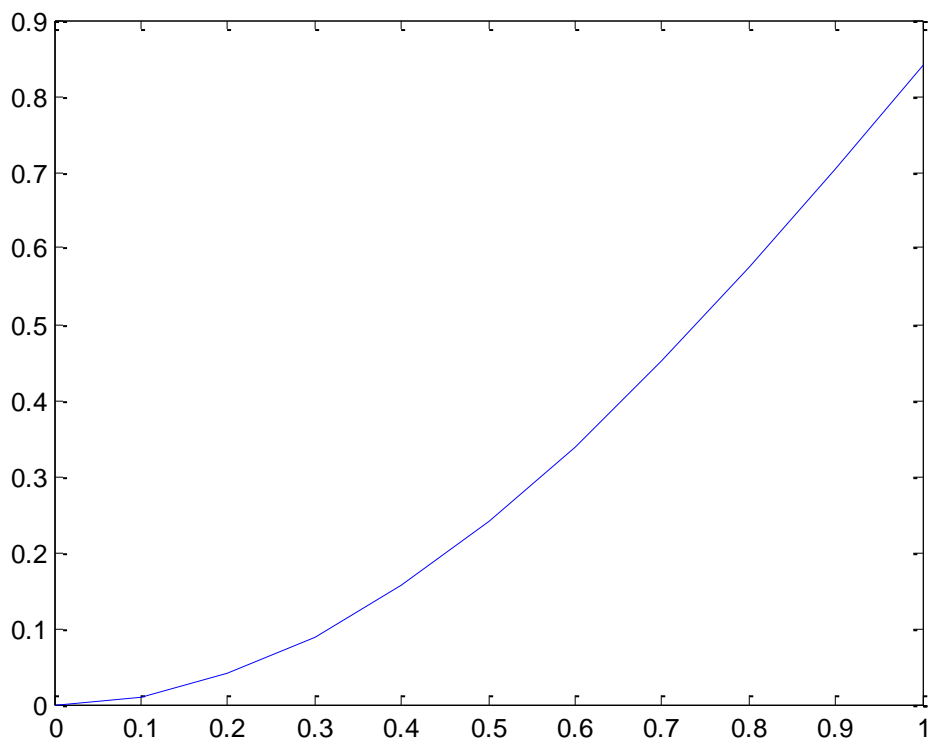
```
>> plot(x, x*sin(x))
```

```
??? Error using ==> * Inner matrix dimensions must agree.
```

il fallait utiliser la multiplication terme à terme .\* soit :

```
>> plot(x, x.*sin(x))
```

et ça marche !



### 3. Attributs de couleurs

Vous aurez remarqué que MATLAB attribue des couleurs par défaut aux courbes. Il est possible de modifier la couleur, le style du trait et celui des points, en spécifiant après chaque couple (abscisse, ordonnée) une chaîne de caractères (entre quotes)

Les tableaux ci-dessous montrent les principales options d’affichage que vous pouvez utiliser.

Style des traits

Symbole	Style
-	Continu
--	Tiré
:	Pointillé
-.	Tiré-pointillé

Marqueurs

Symbole	Marqueur
+	Plus
o	Cercle
*	Astérisque
.	Point
x	Croix
s	Square
d	Diamant
^	Triangle pointant vers le haut
v	Triangle pointant vers le bas

>	Triangle pointant à droite
<	Triangle pointant à gauche
p	Étoile à cinq branches
h	Étoile à six branches

---

Couleurs	
Symbole	Couleur
b	Blue
k	Black
r	Red
g	Green
c	Cyan
y	Yellow
m	Magenta
w	White

Lorsque l'on utilise seulement un style de points, MATLAB ne trace plus de droites entre les points successifs, mais seulement les points eux-même. Ceci peut être pratique par exemple pour présenter des résultats expérimentaux. Les codes peuvent être combinés entre eux. Par exemple

```
>> plot(x,sin(x),'',x,cos(x),'r-')
```

#### 4. Echelles logarithmiques

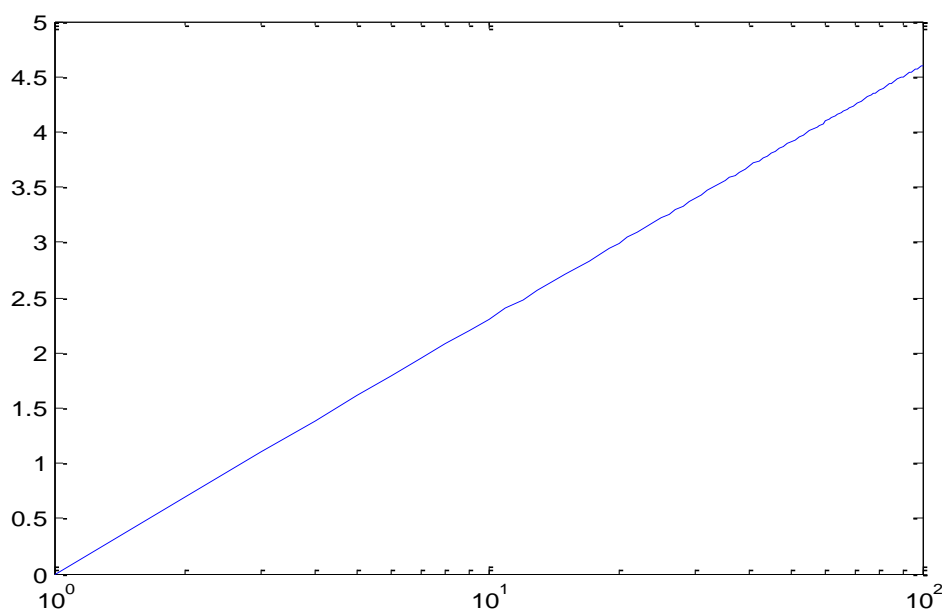
On peut tracer des échelles log en abscisse, en ordonnée ou bien les deux. Les fonctions correspondantes s'appellent respectivement semilogx, semilogy et loglog . Elles s'utilisent exactement de la même manière que plot.

Par exemple :

```
>> x=1:100;
```

```
>> semilogx(x,log(x))
```

donne la courbe suivante :



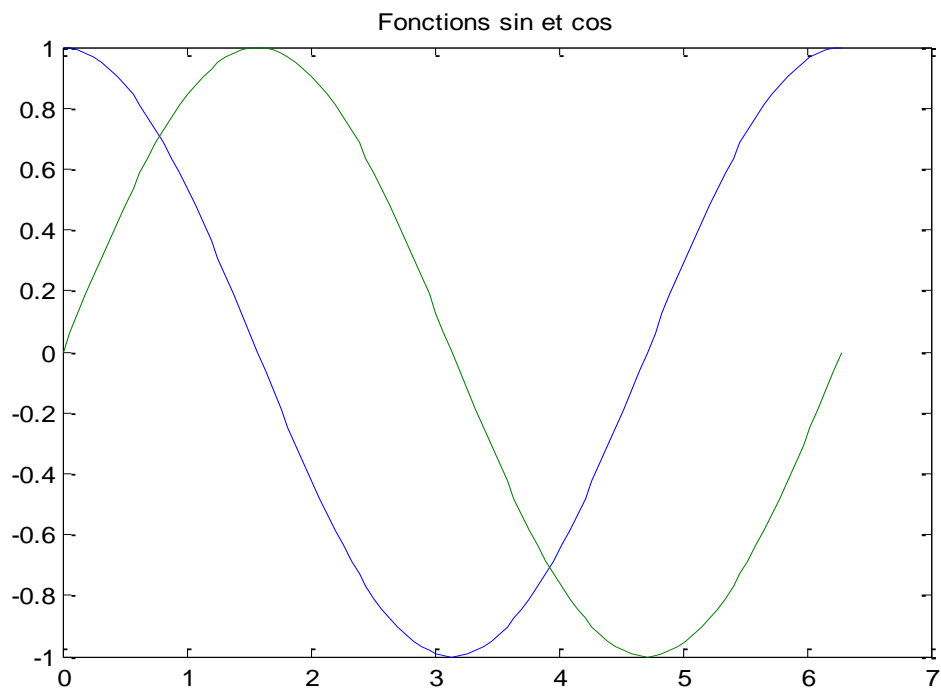
## 5. Décoration des graphiques

### a. Titre

C'est l'instruction `title` à laquelle il faut fournir une chaîne de caractères.

Le titre apparaît en haut de la fenêtre graphique :

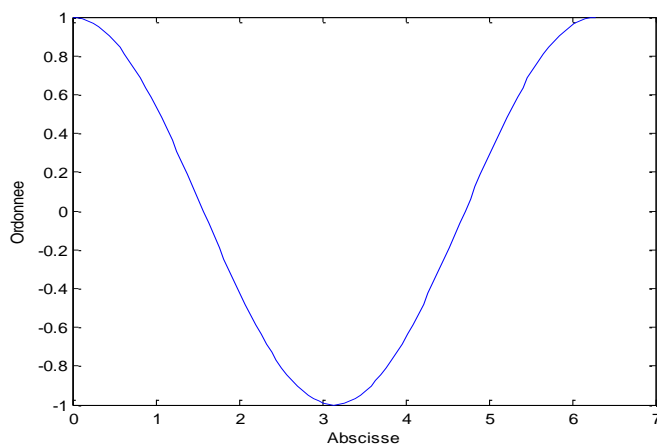
```
>> plot(x,cos(x),x,sin(x))  
>> title('Fonctions sin et cos')
```



### b. Labels

Il s'agit d'afficher quelque chose sous les abscisses et à côté de l'axe des ordonnées :

```
>> plot(x,cos(x))  
>> xlabel('Abscisse')  
>> ylabel('Ordonnée')
```





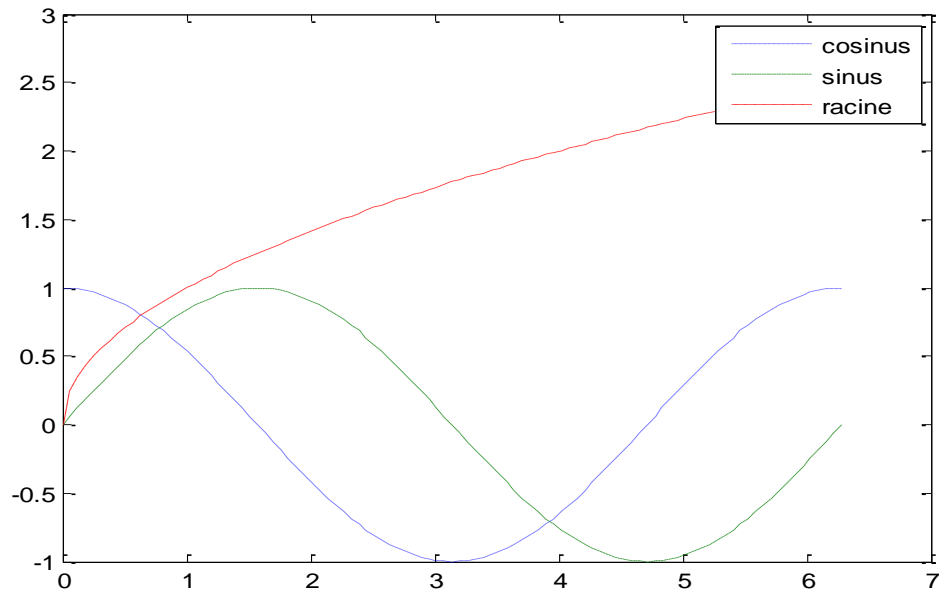
### c. Légendes

C'est l'instruction `legend`

Il faut lui communiquer autant de chaînes de caractères que de courbes tracées à l'écran. Un cadre est alors tracé au milieu du graphique, qui affiche en face du style de chaque courbe, le texte correspondant.

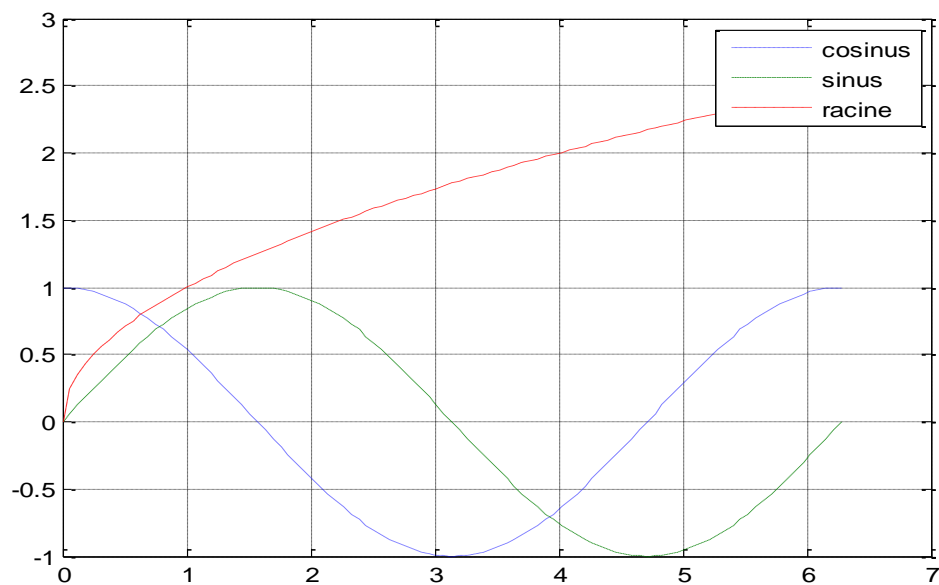
Par exemple :

```
>> plot(x,cos(x),'-',x,sin(x),'-',x,sqrt(x),'--')  
>> legend('cosinus','sinus','racine')
```



Tracer un quadrillage C'est l'instruction `grid`, qui est utilisé après une instruction `plot` affiche un quadrillage sur la courbe. Si on tape à nouveau `grid`, le quadrillage disparaît.

```
>> grid
```



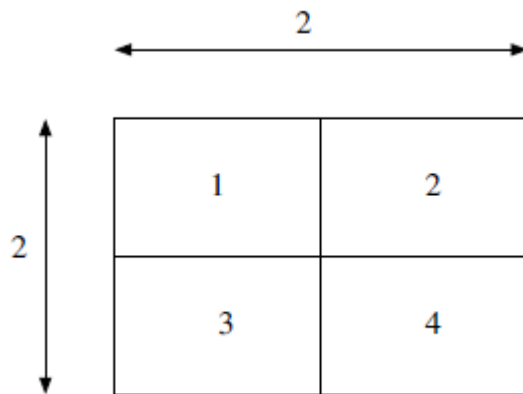
## 6. Afficher plusieurs graphiques (subplot)

Voilà une fonctionnalité très utile pour présenter sur une même page graphique un grand nombre de résultats.

L'idée générale est de découper la fenêtre graphique en pavés de même taille, et d'afficher un graphe dans chaque pavé. On utilise l'instruction subplot en lui spécifiant le nombre de pavés sur la hauteur, le nombre de pavés sur la largeur, et le numéro du pavé dans lequel on va tracer :

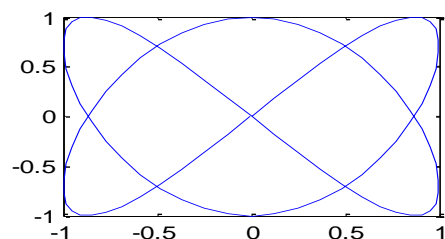
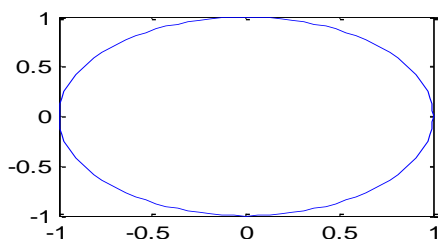
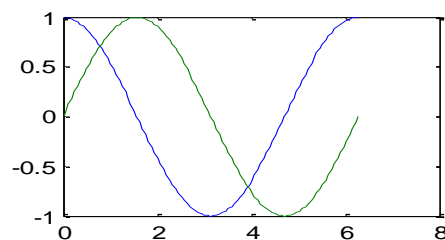
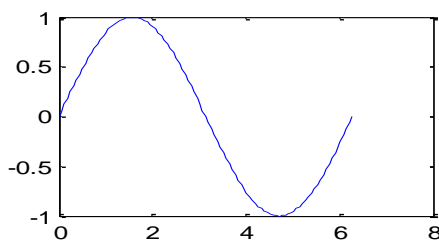
subplot (Nbre pavés sur hauteur, Nbre pavés sur largeur, Numéro pavé)

La virgule peut être omise. Les pavés sont numérotés dans le sens de la lecture d'un texte : de gauche à droite et de haut en bas :



Une fois que l'on a tapé une commande subplot, toutes les commandes graphiques suivantes seront exécutées dans le pavé spécifié. Ainsi, le graphique suivant est obtenu à partir de la suite d'instructions :

```
>> subplot(221)
>> plot(x,sin(x))
>> subplot(222)
>> plot(x,cos(x),x,sin(x),'-.')
>> subplot(223)
>> plot(cos(x),sin(x))
>> subplot(224)
>> plot(sin(2*x),sin(3*x))
```



## V. Graphes 3D

### 1. Lignes de niveau d'une fonction de deux variables

Pour tracer les lignes de niveau de la fonction  $g(x, y)$  pour  $x$  dans  $[x_{\min}, x_{\max}]$  et  $y$  dans  $[y_{\min}, y_{\max}]$  on procède ainsi :

- Création d'un maillage, de maille de longueur  $h$ , du domaine  $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$  par :

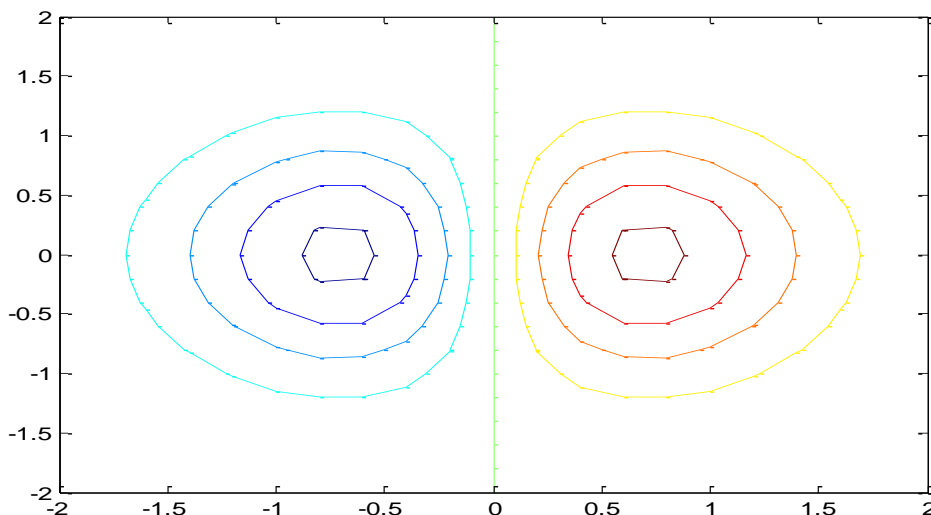
$[X,Y] = \text{meshgrid}(x\_min : h : x\_max, y\_min : h : y\_max).$

- Évaluation de la fonction aux nœuds de ce maillage.
- Affichage des lignes de niveau par la commande `contour(X,Y,Z)`.

#### Exemple.

Pour tracer les lignes de niveau de la fonction  $g(x, y) = x \cdot \exp(-(x^2 + y^2))$  sur le domaine  $[-2, 2] \times [-2, 2]$  en prenant un maillage de maille de longueur  $h = 0.2$ , on exécute :

```
>> [X,Y] = meshgrid(-2:.2:2, -2:.2:2);  
>> Z = X.*exp(-X.^2-Y.^2);  
>> contour(X,Y,Z)
```



On peut également écrire une fonction utilisateur `g.m` et utiliser l'instruction `contour(X,Y,g(X,Y))`.

### 2. Représentation de d'une surface d'équation $z=g(x,y)$

Pour tracer une surface d'équation  $z = g(x, y)$ , on procède de la manière suivante :

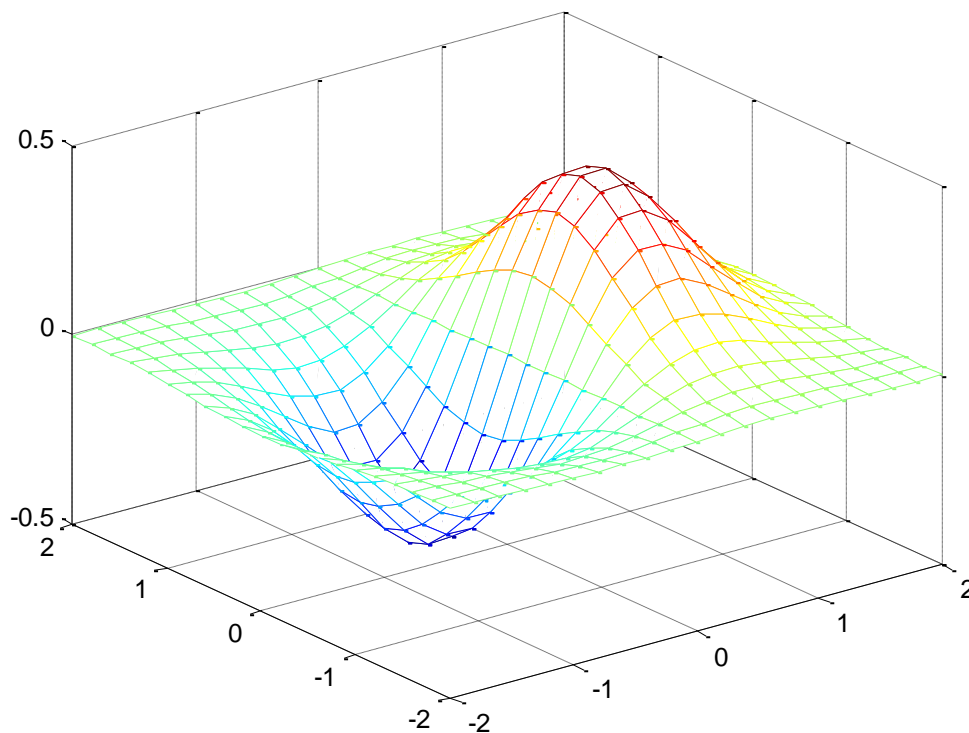
- Création d'un maillage, de maille de longueur  $h$ , du domaine  $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$  par :

$[X,Y] = \text{meshgrid}(x\_min : h : x\_max, y\_min : h : y\_max).$

- Évaluation de la fonction aux nœuds de ce maillage.
- Affichage de la surface par la commande `mesh(X,Y,Z)`.

#### Exemple

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



Par défaut les valeurs extrêmes en  $z$  sont déterminées automatiquement à partir des extremums de la fonction sur le domaine spécifié. Il est possible de modifier ces valeurs (et également les valeurs extrêmes en abscisses et ordonnées) par la commande `axis` dont la syntaxe est : `axis(x_min x_max y_min y_max z_min z_max)`.

### 3. Représenter une surface paramétrée

La commande `surf` permet de tracer une surface paramétrée d'équations,

$$\begin{aligned}x &= g1(u, v), \\y &= g2(u, v), \\z &= g3(u, v).\end{aligned}$$

La fonction  $G = (g1, g2, g3)$  peut être définie directement par une expression MATLAB ou être définie comme une fonction utilisateur.

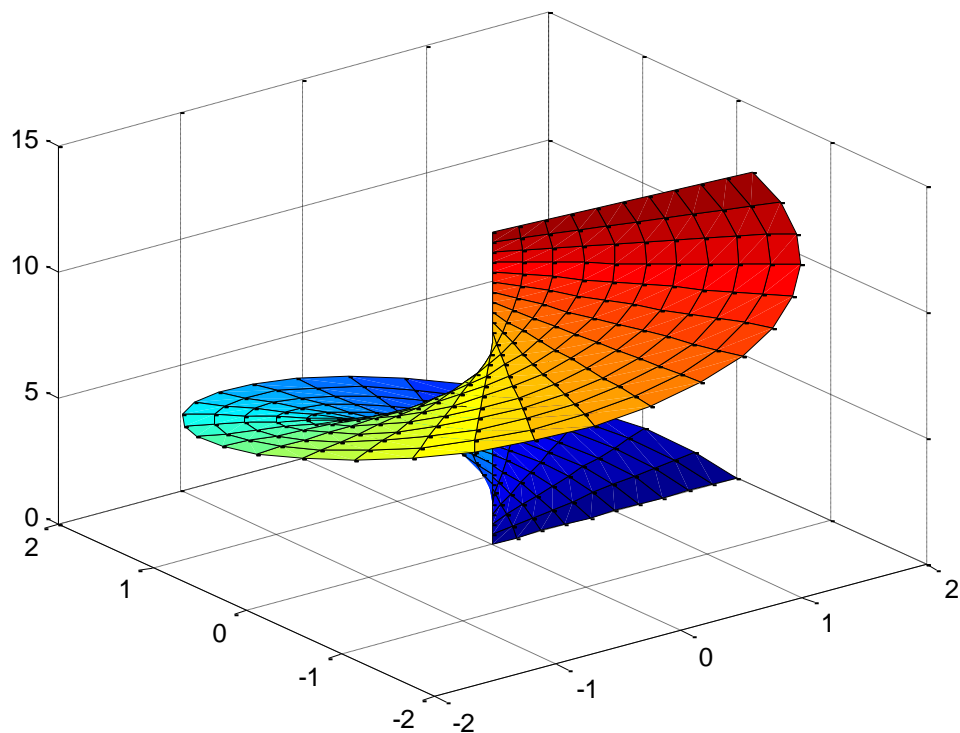
#### Exemple

Pour tracer la surface paramétrée d'équations

$$\begin{aligned}x &= v \cos u, \\y &= v \sin u, \\z &= 2u.\end{aligned}$$

sur le domaine  $[0, 2] \times [0, 2]$  avec un maillage de longueur  $h = 0.2$ , on exécute :

```
>> [U,V] = meshgrid(0:.2:2*pi, 0:.2:2);
>> X = V.*cos(U);
>> Y = V.*sin(U);
>> Z = 2*U;
>> surf(X,Y,Z)
```



## VI. Présentation de l'éditeur de texte

L'éditeur de texte permet à l'utilisateur d'écrire des script et de créer ses propres fonctions. Pour créer une nouvelle fonction ou écrire un script il faut ouvrir une nouvelle fenêtre appelée l'éditeur de texte. Pour se faire, nous devons cliquer sur l'onglet *file*, appuyer sur *New* et choisir l'option *M-file*. Une nouvelle fenêtre indépendante devrait maintenant apparaître à l'écran. C'est ici que nous devons écrire le code pour créer une nouvelle fonction.

Un aspect primordial de la programmation avec Matlab est la séquence de programmation. Lorsque nous appelons une fonction que nous avons créée avec l'aide de l'éditeur de texte à partir de la fenêtre de commandes, Matlab va effectuer les opérations dans l'ordre dans lequel nous avons écrit le code. Par exemple, il est très important que nous définissions nos variables avant de les utiliser sans quoi Matlab nous retournera un message d'erreur dans la fenêtre de commandes.

Lorsque nous écrivons une nouvelle fonction il faut utiliser la syntaxe suivante :

```

1  function [output1,output2,...]=nom_de_la_fonction(input1,input2,...)
2      code
3      ...
4
5  return
6

```

Où les inputs sont les variables d'entrée et les outputs les variables de sortie.

La commande *function* définit le format de la fonction qui sera appelée dans la fenêtre de commandes. Les noms des variables que vous utilisez dans votre code doivent être les mêmes que ceux dans l'entête de la fonction, mais peuvent être différents lorsque vous appelez la fonction dans la fenêtre de commandes. La commande *return* indique la fin du programme. L'ajout de cette commande est toutefois optionnel puisque la fonction n'a pas besoin nécessairement de cette commande pour se terminer.

Lorsque vous écrivez votre code, il est bien d'ajouter des commentaires pour vous rappeler comment fonctionne votre code. Pour ajouter des commentaires il vous faut ajouter le symbole % avant votre commentaire, ainsi le restant de la ligne sera dédié à des commentaires et ne sera pas traité par Matlab. Les commentaires apparaissent normalement en vert dans votre programme. Voici un exemple de code d'une fonction créée par l'utilisateur :

```
function [intt]=trapt(u,deltaz,deltat)

%   Computation of the time integral of a two-dimensional vector

%   -----
%   DESCRIPTION
%   This program computes numerically the time integral of a 2D vector
%   using the trapezoidal method
%
%   Inputs
%       u:      displacement matrix
%       deltaz: space increment
%       deltat:  time increment
%
%   Output
%       intt:    time integral
%   -----

zstep = size(u,2);
tstep = size(u,1);

for j = 1:zstep
    for m = 1:tstep-1
        s(m,1) = (u(m+1,j)+u(m,j));
    end
    intt(1,j)=deltat/2*sum(s,1);
end
return
```

Votre code peut inclure des fonctions de la librairie Matlab mais également des fonctions que vous avez créées. Il est toujours très important de s'assurer que la fonction soit correctement appelée. Si vous avez des doutes, consultez la rubrique d'aide qui vous dira comment la fonction doit être appelée et comment elle fonctionne.

### Remarque.

Il est impératif que la fonction ayant pour nom *func* soit enregistrée dans un fichier de nom *func.m* sans quoi cette fonction ne sera pas « visible » par MATLAB.

Il n'y a pas de mot-clé (par exemple *end*) pour indiquer la fin de la fonction. La fonction est supposée se terminer à la fin du fichier.

En pratique, il ne faut écrire qu'une seule fonction par fichier (qui doit porter le nom de cette fonction).

Il existe la notion de « sous-fonction ». Une sous-fonction est une fonction écrite dans le même fichier qu'une autre fonction (dite principale) et qui ne sera utilisable que par cette fonction principale (une sous-fonction ne peut pas être appelée par un autre sous-programme que la fonction principale).

On peut provoquer un retour au programme appelant grâce à la commande `return`.

### Affichage de résultat

La commande **disp** permet d'afficher un tableau de valeurs numériques ou de caractères.

L'autre façon d'afficher un tableau est de taper son nom.

```
>> A=magic(3);
```

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

```
8 1 6
```

```
3 5 7
```

```
4 9 2
```

```
>> A
```

```
A =
```

```
8 1 6
```

```
3 5 7
```

```
4 9 2
```

On utilise fréquemment la commande **disp** avec un tableau qui est une chaîne de caractères pour afficher un message.

Par exemple : `disp('Calcul du déterminant de la matrice A')`.

On utilise également la commande `disp` pour afficher un résultat.

Par exemple :

```
disp(['Le déterminant de la matrice A vaut ', num2str(det(A))]).
```

**Attention**, si la chaîne de caractères contient une apostrophe il est impératif de doubler l'apostrophe.

### Lecture de données

La commande `input` permet de demander à l'utilisateur d'un programme de fournir des données. La syntaxe est

```
var = input(' une phrase ').
```

Cette donnée peut être une valeur numérique ou une instruction MATLAB (qui est alors évaluée). Un retour chariot provoque la fin de la saisie.

Il est possible de provoquer des sauts de ligne pour aérer la présentation en utilisant le symbole `\n` de la manière suivante : `var = input('\n une phrase : \n')`

#### Exemple.

```
N=input('donner la valeur de l''entier N; \n N= ');
```

```
donner la valeur de l'entier N;
```

```
N= 10
```

```
>> V=input('donner la valeur du vecteur ...
```

```
V; \n V = ');
```

```
donner la valeur du vecteur V;
```

```
V = [1 2 3]
```

```
>> N
```

```
N =
```

```
10
```

```
>> V
```

V =  
1 2 3

Si l'on souhaite saisir une réponse de type chaîne de caractères on utilise la syntaxe :  
var = input(' une phrase ','s').

## Librairies de fonctions

Dans cette section sont présentées diverses fonctions qui pourront vous être utiles durant le cours. La liste de ces fonctions est très loin d'être exhaustive, c'est pourquoi il vous faudra utiliser la rubrique d'aide pour trouver des fonctions qui effectueront les opérations que vous désirez faire.

### Statistiques

max(x) : Donne la valeur maximum à l'intérieur du vecteur x  
mean(x) : Calcule la moyenne d'un vecteur x  
median(x) : Calcule la médiane d'un vecteur x  
min(x) : Donne la valeur minimum à l'intérieur du vecteur x  
std(x) : Calcule l'écart-type d'un vecteur x

### Matrices

Les fonctions matricielles les plus courantes sont:

det(A) : renvoie le déterminant de la matrice carrée A  
inv(A) : renvoie le l'inverse de la matrice carrée A  
eig(A) : renvoie les valeurs propres (eigenvalues) de la matrice carrée A  
[P,D]= eig(A) : renvoie une matrice diagonale D formée des valeurs propres de A et une matrice P dont les vecteurs colonnes sont les vecteurs propres correspondant.  
poly(A) : renvoie les coefficients du polynôme caractéristique associé à la matrice carrée A(attention à l'ordre des coefficients).  
rank(A) : renvoie le rang de la matrice carrée A,  
trace(A) : renvoie la trace de la matrice A.  
expm(A) : renvoie l'exponentielle matricielle de A  
det(A) : Calcule le déterminant de la matrice A  
eye(i) : Crée une matrice identité de taille i x i  
ones(m,n) : Crée une matrice m x n remplie de un  
zeros(m,n) : Crée une matrice m x n remplie de zero  
rand(i,j) : Crée une matrice de nombres aléatoires compris entre 0 et 1 de taille i x j  
size(A,i) : Donne la taille de la matrice A dans la ième dimension  
[m,n]=size(A) : donne le nombre de ligne et le nombre de colonne de la matrice A  
sum(A,i) : Calcule la somme des lignes d'une matrice A dans la ième dimension

transpose(A) : Donne la transposée de la matrice A  
on peu donner la transposée de A par A'

### Graphiques

bar(x,y) : Trace un diagramme à barres à partir des vecteurs x et y  
errorbar(x,sd) : Affiche l'écart-type (sd) d'un vecteur x en utilisant des barres  
hold : Rendre les échelles constantes  
hist(y) : Trace un histogramme à partir des vecteurs x et y



plot(x) : Trace le graphique du vecteur x  
subplot(m,n,p) : Crée l'espace pour le pième sous-graphique de l'espace graphique  
m x n

### Annotations graphiques

grid : Ajoute un grillage au graphique  
legend : Ajoute une légende au graphique  
title : Ajoute un titre au graphique  
xlabel : Étiquette l'axe des x  
ylabel : Étiquette l'axe des y

### Résolution des systèmes linéaires

La commande MATLAB (backslash) est la commande Générique pour résoudre un système linéaire . L'algorithme mis en œuvre dépend de la structure de la matrice A du système.

MATLAB utilise dans l'ordre les méthodes suivantes:

Si A est une matrice triangulaire, le système est résolu par simple substitution.

Si la matrice A est symétrique ou hermitienne, définie positive, la résolution est effectuée par la méthode de **Choleski**.

Si A est une matrice carrée mais n'entrant pas dans les

Deux cas précédents, une factorisation **LU** est réalisée en utilisant la méthode d'élimination de Gauss avec stratégie de pivot partiel.

Si A n'est pas une matrice carrée, la méthode QR est utilisée.

Chacune des méthodes précédentes peut être utilisée de

Manière spécifique grâce aux commandes chol, lu, qr.

Dans le cas des matrices stockées sous forme **sparse**, des algorithmes particuliers sont mis en œuvre. Il est également possible d'utiliser des méthodes itératives. Les commandes **cgs**, **bicg**, **bicgstab** mettent par exemple en œuvre des méthodes de type gradient conjugué.....

#### Exemple.

```
>>A=[12;34]; b=[11];
```

```
>>x=A\b
```

```
x=
```

```
-1
```

```
1
```

```
>>A*x
```

```
ans=
```

```
1
```

```
1
```

## VII. Conclusion

Matlab est un langage de programmation, un logiciel, un monde de fonctions et de techniques. Matlab est un outil très simple, très puissant, très complet et qui ne nécessite pas une formation spéciale pour le maîtriser, juste de ne pas hésiter à utiliser **help**, à naviguer dans **demo** et à consulter les fichiers de documentation pdf.