INTRODUCTION

Matlab est un logiciel de calcul numérique commercialisé par la société MathWorks1. Il a été initialement développé à la fin des années 70 par Cleve Moler. Matlab signifie **Mat**rix **lab**oratory. Le logiciel de base peut être complété par de multiples *toolboxes*, c'est-à-dire des boîtes à outils. Celles-ci sont des bibliothèques de fonctions dédiées à des domaines particuliers. Nous pouvons citer par exemple : l'Automatique, le traitement du signal, l'analyse statistique, l'optimisation…

INTERFACE PRINCIPALE

Le logiciel propose un véritable environnement de travail composé de multiples fenêtres.
Nous pouvons distinguer quatre blocs :

- ***Command window*** (console d'exécution) : à l'invite de commande « >> », l'utilisateur peut entrer les instructions à exécuter. Il s'agit de la fenêtre principale de l'interface.
- ***Current directory*** (répertoire courant) : permet de naviguer et de visualiser le contenu du répertoire courant de l'utilisateur.
- ***Workspace*** (espace de travail) : permet de visualiser les variables définies, leur type, la taille occupée en mémoire…
- ***Command history*** : historique des commandes que l'utilisateur a exécutées.

A la validation de l'instruction, l'interface affiche le résultat de cette dernière. Afin d'alléger l'affichage, un point-virgule « ; » en fin de commande empêche le renvoie du résultat dans la fenêtre (évidemment l'instruction est toujours exécutée). Par exemple :

```
>> 3*5;
>>
```

Le calcul a été effectué mais le résultat n'est pas affiché.

LES VARIABLES

Aspects élémentaires

Matlab gère de façon automatique les nombres entiers, réels, complexes, les chaînes de caractères… Ainsi, la déclaration des variables est implicite, le symbole d'affectation est le signe « = »

```
>> x = 4
x =
4
>> y = 2
y =
2
>> x + y
ans =
6
>> x - y
ans =
2
>> x * y
ans =
8
>> x / y
ans =
2
>> x^y
ans =
16
```

Si l'utilisateur n'affecte pas explicitement le résultat d'une opération à une variable, Matlab l'affecte automatiquement à la variable « ans ». Concernant le nom des variables, l'interpréteur fait la distinction entre les minuscules et les majuscules.
La command « `clc` » permet d'efface l'écran (command window), la command « `clear` » permet de supprimer une variable du workspace (« `clear all` » les supprime toutes). Toutes les commandes tapées dans la ***Command window*** peuvent être retrouvées et éditées grâce aux touches ↑ de direction. Appuyez sur la touche pour remonter dans les commandes précédentes, ↓ pour redescendre.

Calcul arithmétique

\+ : sommation
\- : soustraction
/ : division
* : multiplication
^ *:puissance*

Constantes prédéfinies

Il existe des symboles auxquels sont associés des valeurs prédéfinies. En voici quelques uns :

Symbole	Signification	Valeurs
Pi	Nombre π	3.141592…
I ou j	Nombre complexe	$\sqrt{-1}$
realmax	Plus grand nombre flottant codable	1.7977e+308
realmin	Plus petit nombre flottant codable	2.2251e-308
inf	Nombre infinie	∞

Attention, ces valeurs peuvent être écrasées si le symbole est redéfini.

```
>> pi=1
pi =
1
>> clear pi
>> pi
ans =
3.1416
```

Il est donc possible de définir et manipuler explicitement des nombres complexes.

```
>> a = 2 + i*3;
>> b = 1 - i;
>> a + b
ans =
3 + 2i
>> a-b
ans =
   1+ 4i
>> a*b
ans =
   5+ 1i
>> a/b
ans =
   -0,5+ 2,5i
>> a^b
ans =
   9.2043 - 2.8440i
```

Dans MATLAB, les fonctions `'abs'` et `'angle'` permettent l'obtention directe du module et de l'argument d'un nombre complexe.

```
>>r=abs(a)
r =
3.6056
>>theta=angle(a)
theta =
0.9828
```

L'angle theta est en radians.

Quelques fonctions mathématiques

Fonctions trigonométriques :

```
sin     cos     tan
asin    acos    atan
sinh    cosh    tanh
```

Fonctions puissances, exponentielle et logarithmes :

```
power (ou ^)    sqrt    exp    log    log10
```

Fonctions réels vers entiers (arrondi, troncature…) :

`Fix(x)` : arrondi par défaut du réel x
`Floor(x)` : arrondi au voisinage de - ∞ du réel x
`Round(x)` : arrondi entier de x
`ceil(x)` : arrondi au voisinage de + ∞ du réel x

Fonctions autres :

`format` : affiche les résultats avec 4 chiffres après la virgule.
`format long` : affiche les résultats avec 16 chiffres après la virgule.
`sign(var:` **(signe) retourne 1 si** `var>0`, **-1 si** `var<0` **et 0 si** `var=0`.
`abs(var)`: **module de** `var`.
`real(var)`: **partie réelle de** `var`.
`imag(var)`: **partie imaginaire de** `var`.
`gcd(var1,var2)`: **plus grand diviseur commun des entiers** `var1` et `var2`.
`lcm(var1,var2)`: **plus petit multiple commun des entiers** `var1` et `var2`.
`gamma(var)`: **factorielle de** `var`.

LES VECTEURS

Avec Matlab, on travaille essentiellement avec un type d'objet : les matrices (D'où son nom : MATrix LABoratory). Une variable scalaire est une matrice de dimension 1 x 1 et un vecteur est une matrice de dimension 1 x n ou n x 1. Il est capital d'être à l'aise avec ces notions pour comprendre au mieux la philosophie de Matlab et l'exploiter efficacement.

Définition d'un vecteur

Un vecteur n'est rien d'autre qu'un tableau de valeurs. Il existe plusieurs façons de créer un vecteur et la plus simple d'entre elles est de l'écrire explicitement.

```
>> v = [1 2 3 4]
v =
1 2 3 4
```

L'ensemble des composantes est donné entre crochets et les valeurs sont séparées par un espace (ou une virgule « , »). Nous avons ici défini un vecteur ligne. Un vecteur colonne est créé en utilisant un point-virgule « ; » comme délimiteur.

```
>> v = [1 ; 2 ; 3 ; 4]
v =
1
2
3
4
```

Bien que simple, cette méthode n'est pas pratique pour définir des vecteurs de taille importante. Une seconde méthode utilise l'opérateur deux-points « : ». Il permet de discrétiser un intervalle avec un pas constant.

```
>> v = 0:0.2:1
v =
0 0.2 0.4 0.6 0.8 1
```

Cette instruction crée un vecteur contenant des valeurs allant de 0 à 1 avec un pas de 0.2. La syntaxe est la suivante : `vecteur = valeur_initial:incrément:valeur_finale`. Par défaut, le pas est égal à 1.

```
>> v = 0:5
v =
0 1 2 3 4 5
```

Enfin, des fonctions prédéfinies permettent de générer des vecteurs automatiquement.

```
>> v = linspace(0,10,1000);
>> v = logspace(-1,2,1000);
```

La première fonction crée un vecteur de 1000 points avec des valeurs allant de 0 à 10 également espacées. La seconde crée un vecteur de 1000 points sur un intervalle de 10^{-1} à 10^{2} avec un espacement logarithmique.

On peut accéder aux différents éléments d'un tableau en spécifiant un (ou des) indice(s) entre parenthèses.

```
>> v = [6 4 -1 3 7 0.3];
>> v(3)
ans =
-1
>> v(2:4)
ans =
4 -1 3
```

v(3) retourne le 3ième élément du vecteur *v*. L'argument 2:4 permet de sélectionner un bloc d'éléments (ici du second au quatrième).

Quelques fonctions utiles

Nous présentons dans ce paragraphe un ensemble de fonctions usuelles liées à l'utilisation des tableaux.

`length(v)`	renvoie la taille du tableau.
`max(v)`	renvoie la valeur maximale du tableau.
`min(v)`	renvoie la valeur minimale du tableau.
`mean(v)`	renvoie la valeur moyenne des éléments du tableau.
`sum(v)`	calcul la somme des éléments du tableau.
`prod(v)`	calcul le produit des éléments du tableau.
`sort(v)`	range les éléments du tableau dans l'ordre croissant.

Toutes les fonctions mathématiques vues dans la partie du variable sont applicables aux variables de type vecteur.

```
>> v = [0 pi/4 pi/2 pi 2*pi]
v =
0 0.7854 1.5708 3.1416 6.2832
>> cos(v)
ans =
1.0000 0.7071 0.0000 -1.0000 1.0000
```

LES MATRICES

Définition d'une matrice

La définition d'une matrice est délimitée pas des crochets « [] ». Les différents éléments d'une ligne sont séparés par un espace (ou des virgules) et les différentes lignes sont séparées par des points-virgules « ; ». Ainsi pour définir une variable matricielle

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

on écrira :

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

ou

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

L'accès à un élément d'une matrice s'opère en spécifiant des indices entre parenthèses à la suite de son nom. L'élément situé la `i`$^{\text{ième}}$ ligne et la `j`$^{\text{ième}}$ colonne est obtenu par la commande `M(i,j)`. Par exemple, la valeur M_{23}est récupérée en tapant

```
>> M(2,3)
ans =
6
```

On peut également modifier directement un des éléments en lui affectant une nouvelle valeur.

```
>> M(2,3)=13;
>> M
M =
1 2 3
4 5 13
7 8 9
```

Matrices particulières

Quelques matrices particulières, et très utilisées, sont définies plus aisément au travers de fonctions. Ces fonctions prennent en argument les dimensions de la matrice que l'on souhaite construire. Le premier désigne le nombre de lignes et le second le nombre de colonnes.

```
>> Z = zeros(2,3)
Z =
0 0 0
0 0 0
```

Une matrice pleine de 1 :

```
>> U = ones(4,3)
U =
1 1 1
1 1 1
1 1 1
>> I = eye(3)
I =
1 0 0
0 1 0
0 0 1
```

Une matrice diagonale :

```
>> D = diag([2,4,0,7])
D =
2 0 0 0
0 4 0 0
0 0 0 0
0 0 0 7
```

Contrairement aux précédentes, cette dernière fonction prend en argument un vecteur. La taille de la matrice diagonale est donc déterminée par la taille du vecteur.

Il est souvent utile d'extraire des blocs d'un tableau existant. Pour cela on utilise l'opérateur « : ». Pour cela, il faut spécifier pour chaque indice la valeur de début et la valeur de fin. La syntaxe générale est donc la suivante (pour un tableau à deux dimensions) : `tableau(début:fin, début:fin)`.

Ainsi pour extraire le bloc $\begin{bmatrix} 2 & 3 \\ 5 & 6 \end{bmatrix}$ de la matrice M, on tapera :

```
>> M(1:2,2:3)
ans =
2 3
5 6
```

Le caractère « : » seul, signifie toute la longueur est extraite. De cette façon, on peut isoler une ligne, ou une colonne, complète. Exemples :

```
>> M(1:2,:)
ans =
1 2 3
4 5 6
>> M(1,:)
ans =
1 2 3
>> M(:,2)
ans =
2
5
8
```

Quelques fonctions utiles

Nous présentons dans ce paragraphe un ensemble de fonctions usuelles liées à l'utilisation des matrices :

« `size(M)` » renvoie les dimensions de la matrice.
« `max(M)` » renvoie un vecteur-ligne contenant les valeurs maximales associées à chaque colonne.
« `min(M)` » renvoie un vecteur-ligne contenant les valeurs minimales associées à chaque colonne.
« `rank(M)` » renvoie le rang de la matrice.
« `det(M)` » renvoie le déterminant de la matrice.
« `diag(M)` » extrait la diagonale de la matrice.
«`trace(M)` » renvoie la trace de la matrice (somme d'éléments de la diagonale de la matrice)
«`triu(M)` » extrait la matrice-triangle supérieure de M. `tril` donne la matrice-triangle inférieure.
« `eig(M)` » renvoie un vecteur contenant les valeurs propres de la matrice.

Opérations sur les matrices

Un des atouts remarquables de Matlab est la possibilité d'effectuer les opérations arithmétiques traditionnelles de façon naturelle sans avoir à les programmer. Les opérateurs standards sont donc directement applicables aux matrices. Si la commande entrée ne respecte pas les règles de calcul matriciel (compatibilité des opérandes), le logiciel renverra une erreur.

```
+       addition
-       soustractn
*       produit
/       division à droite
\       division à gauche
^       puissance
'       transposition
inv()   inversion
```

```
>> A = [2 1;6 9];
>> B = [1 0;-4 3];
>> A + B
ans =
3 1
2 12
>> A * B
ans =
-2 3
-30 27
>> A'
ans =
2 6
1 9
>> inv(B)
ans =
1.0000 0
1.3333 0.3333
>> A / B
ans =
3.3333 0.3333
18.0000 3.000
>> A^2
ans =
10 11
66 87
```

Si l'on souhaite effectuer une opération, non pas matricielle, mais éléments par éléments, l'opérateur doit être précédé d'un point « . » : .* ./ .^ .\

Appliquons ces opérateurs aux matrices de l'exemple précédent.

```
>> A .* B
ans =
2 0
-24 27
>> B ./ A
ans =
0.5000 0
-0.6667 0.3333
>> A.^2
ans =
4 1
36 81
```

`expm` `logm` `sqrtm` `mpower`

Par exemple, les fonctions « `exp` » et « `expm` » effectuent deux opérations tout à fait différentes :

Pour $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$

`exp(A)` $\rightarrow \begin{bmatrix} e^{a_{11}} & e^{a_{12}} \\ e^{a_{21}} & e^{a_{22}} \end{bmatrix}$ et `expm(A)` $\rightarrow e^{A}$

REPRESENTATIONS GRAPHIQUES

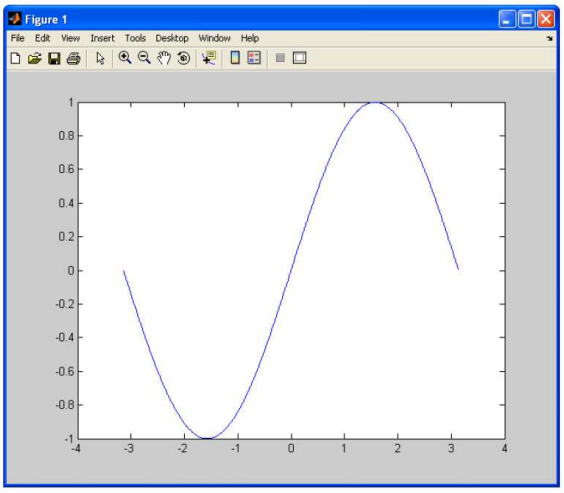
Les bibliothèques de Matlab proposent un très grand nombre de fonctions pour la manipulation d'objets graphiques.

Graphiques 2D

le tracé d'une courbe s'effectue à partir de la commande `plot()`.

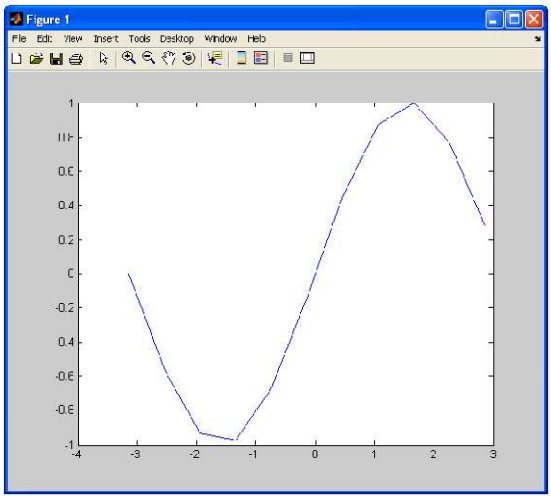
Traçons la fonction sinus dans l'intervalle [$-\pi$, π] avec un pas de 0.01.

```
>> x = -pi : .01 : pi;
>> y = sin(x);
>> plot(x,y)
```



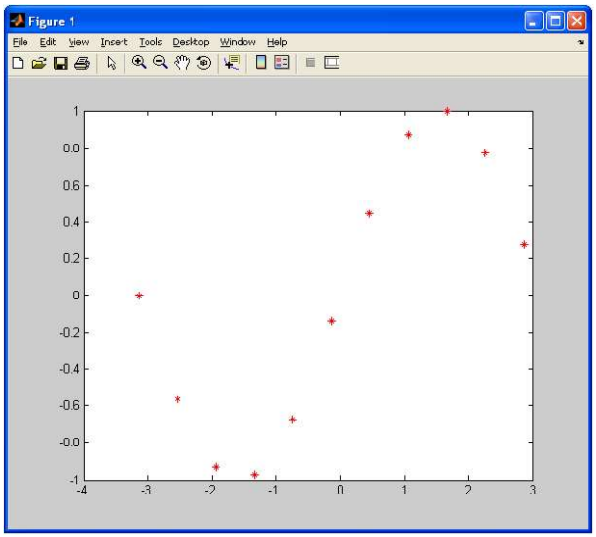
Le pas étant faible, la courbe semble parfaitement tracée. Bien évidemment, si l'on diminue le nombre de points (le pas est augmenté), la courbe apparaitra plus saccadée.

```
>> x = -pi : .6 : pi;
>> y = sin(x);
>> plot(x,y)
```



La commande `plot` prend un troisième argument permettant de spécifier la couleur du tracé et le symbole de représentation.
Retraçons l'exemple précédent en rouge avec des étoiles pour chaque point.

```
>> x = -pi : .6 : pi;
>> y = sin(x);
>> plot(x,y,'r*')
```



Différentes options sont disponibles (consulter le `help plot`) :
Voici quelques exemples parmi les manipulations les plus simples :

```
>> xlabel('valeur x')
>> ylabel('valeur y')
>> title('mon graphique')
>> legend('ma courbe')
>> grid on
>> axis([xmin xmax ymin ymax])
```

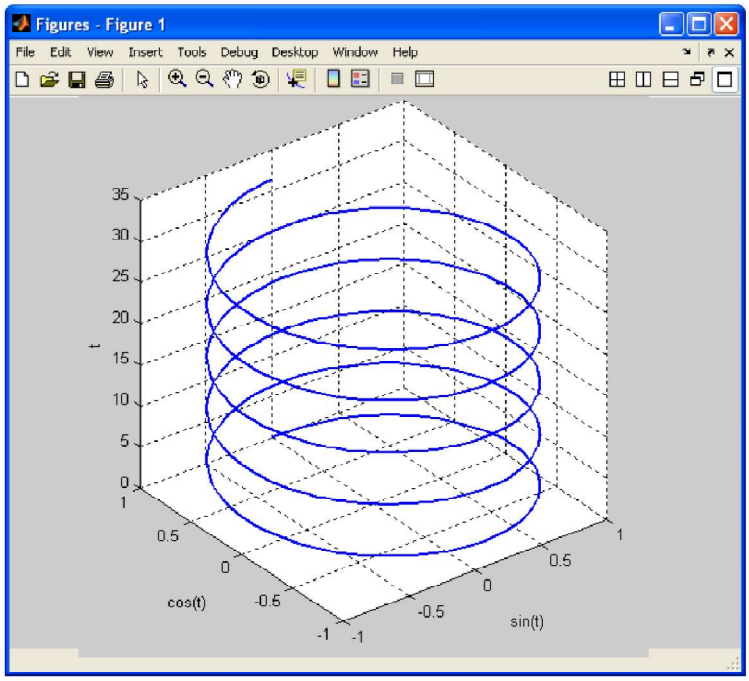
Graphiques 3D

Nous montrons ici les possibilités de Matlab en graphisme 3D sur quelques exemples.

Tracé de courbes dans l'espace

```
>> t = 0:pi/50:10*pi;
>> plot3(sin(t),cos(t),t)
>> grid on
>> axis square
>> xlabel('sin(t)'), ylabel('cos(t)'), zlabel('t')
```

La fonction prend en argument 3 vecteurs de même taille. Son fonctionnement est similaire à celui de `plot`. Elle affiche dans un système d'axe à 3 dimensions les triplets `[x(i),y(i),z(i)]`.


Figures - Figure 1
File Edit View Insert Tools Debug Desktop Window Help
35
30
25
20
15
10
5
0
t
1
0.5
0
-0.5
-1
cos(t)
-1
-0.5
0
0.5
1
sin(t)