1ère **ANNEE**

INITIATION AU LANGAGE

MATLAB

ENSEIGNANT: M.LABENI

Sommaire

1. Introduction

L'objectif de ce cours est d'initier l'étudiant aux concepts de base de MATLAB pour lui permettre un apprentissage rapide des commandes du logiciel.

MATrix LABoratory (MATLAB) est un logiciel de calcul scientifique, dont les objets privilégiés sont les matrices (réelles, complexes), les vecteurs (réels, complexes) et les scalaires. Les opérations sous MATLAB sont donc naturelles au sens de ces objets. MATLAB est un outil pédagogique indispensable à la compréhension (par la simulation) de cours comme le traitement numérique du signal (déterministe et aléatoire), l'algebre linéaire (résolution de systèmes etc.).

Ce système est doté d'une invite de commandes dont la syntaxe est hybride du ***C*** et du ***Pascal.*** Ceci permet (à l'aide de ***Return***) d'exécuter les commandes les unes à la suite des autres**.** MATLAB dispose aussi d'une autre manière pour exécuter les commandes et ça à l'aide des scripts (Blocs de de commandes dans des fichiers .m) et d'exécuter ces scripts en tapant le nom du fichier directement au niveau de l'invite de commande ou utiliser la commande *Save and Run* de l'environnement de développement de MatLab.

2. Les Commandes Sous Matlab:

Pour toutes les commandes de MATLAB, il existe une aide directement accessible à partir de l'invite de commandes: cette aide est réalisée par: » help **COMMANDE**

Exemple :

```
>> help sin
 SIN    Sine of argument in radians.
    SIN(X) is the sine of the elements of X.

    See also asin, sind.

    Overloaded methods:
       distributed/sin
       codistributed/sin
       sym/sin

    Reference page in Help browser
       doc sin
```

L'invite de commande se présente sous la forme du prompt (curseur clignotant):

```
fx >> _
```

Si l'on désire affecter la valeur 1,2 à la variable ***var_***, on écrira :

```
>> var_=1.2
```

Ce qui produira:

```
var_ =

    1.2000
```

Un exemple de déclaration d'une variable logique (booléenne) est:

```
>> bool=true

bool =

     1
```

On peut consulter les variables existent dans la mémoire par la commande *who* :

```
>> who

Your variables are:

bool  var_
```

On trouvera sous MATLAB les commandes suivantes :

Commande	Exemple	Signification

whos	`>> whos`
	`Name  Size  Bytes  Class  Attributes`
	`bool  1x1  1  logical`
	`var_  1x1  8  double`
		Affiche les variables actuellement présentes en mémoire ainsi qu'une série d'informations comme leur taille (size: lignes×colonnes), leur taille en octets (bytes), leur type (class) et leur attribut (complex, global, ...etc)

exist ('Variable')	`>> exist('var_')`
	`ans =`
	`1`
		Affiche la valeur **1 (à l'aide de la variable automatique ans = answer)** si la variable 'Variable' existe, et la valeur **0** si la variable n'existe pas.

clear Variable	`>> clear var_`
	`>> var_`
	`Undefined function or variable 'var_'.`
		Efface de la mémoire la variable 'Variable'.

clear	`>> clear`
	`>> who`
	`>>`
		Efface toutes les variables en mémoire.

Il existe également une description interactive des différentes fonctions accessible au moyen de l'option ***Function Browser*** du menu ***Help.***

3. Les Vecteurs

3.1. Déclaration de scalaires

Pour définir un scalaire on écrira:

```
>> x=3.2

x =

    3.2000
```

Cette commande définit la variable réelle (double) x =3,2

La commande:

```
>> z=.0+2.*i
```

définit le nombre complexe z = 0,0+2,0i et affiche

```
z =

        0 + 2.0000i
```

Pour connaitre la valeur d'une variable on doit taper son nom en actionnant la touche **Return**.

```
>> x

x =

    3.2000
```

Il existe également quelques valeurs et symboles particuliers :

ans	Variable créée automatiquement quand le résultat d'une expression n'est pas affecté à une variable.
pi	Constante correspondant à $\boldsymbol{\pi}$**.**
i, j	Constante correspondant à $\sqrt{-1}$**.**
;	Utilisé à la suite d'une commande, ce caractère supprime l'écho à l'écran de cette commande et de son résultat.
...	Ajouté au cours d'une commande, ce caractère permet la continuation de la commande â la ligne suivante.

3.2. Création d'un vecteur

La commande suivante nous permet de créer un vecteur ligne (ses éléments sont séparés par des espaces ou virgules) :

```
>> v=[ 1.2 2 sin(5) 6 tan(0)]

v =

    1.2000    2.0000   -0.9589    6.0000         0
```

v est un vecteur ligne de 5 colonnes.

3.3. Opérations sur les vecteurs

Pour accéder à un élément du vecteur précédent :

```
>> v(4)

ans =

     6
```

Pour remplacer v(2) par v(1) et v(1) par v(4), on écrira:

```
>> v(2)=v(1);v(1)=v(4)

v =

    6.0000    1.2000   -0.9589    6.0000         0
```

Pour passer d'un vecteur ligne à un vecteur colonne, on doit transposer le vecteur en utilisant l'opérateur unaire **'** :

```
>> v'

ans =

    6.0000
    1.2000
   -0.9589
    6.0000
         0
```

Pour additionner ou soustraire des vecteurs, il faut que ces vecteurs aient la même structure (ligne ou colonne) et la même dimension:

```
>> v+v

ans =

   12.0000    2.4000   -1.9178   12.0000         0
```

Il existe deux multiplications possibles des vecteurs v et u de même dimension :

La première $v * u$ représente le produit matriciel de deux vecteurs et n'est donc possible que si le nombre de lignes de v est égal au nombre de colonnes de u:

```
>> u=v';v*u

ans =

   74.3595
```

La seconde $v.* u$ est une multiplication élément par élément .* (element wise product) qui est indépendante du mode de représentation ligne ou colonne.

```
>> v.*v

ans =

   36.0000    1.4400    0.9195   36.0000         0

>> u.*u

ans =

   36.0000
    1.4400
    0.9195
   36.0000
         0
```

3.4.Génération automatique de vecteurs

La structure ":" permet la création d'une séquence ordonnée de valeurs, ainsi la commande :

`>> vect=[debut:pas:fin]` génère un vecteur $vect$ dont la première composante est $début$, la dernière fin et le pas entre deux composantes successives est pas. Si pas est omis il est pris par défaut égal à 1. Le nombre d'éléments est égal à la partie entière de $\frac{fin-debut}{pas} + 1$.

Exemple :

```
>> vect=[4:10]
```

Définit le vecteur :

```
vect =

     4     5     6     7     8     9    10
```

Il existe des fonctions prédéfinies pour créer des suites ordonnées de valeurs, le tableau suivant récapitule les plus utilisées:

Fonction	Exemple	Signification

linspace(Y_{min},Y_{max},N)	`>> v=linspace(-pi,pi,4)`
	`v =`
	`-3.1416 -1.0472 1.0472 3.1416`
		Définit un vecteur ligne dont les composantes sont linéairement espacées entre les valeurs Ymin et Ymax, ce vecteur contient N éléments.

logspace(d,f,N)	`>> u=logspace(-pi,pi,4)`
	`u =`
	`0.0007 0.0118 0.1924 3.1416`
		Définit un vecteur ligne dont les composantes n sont logarithmiquement espacées entre les valeurs 10^d et 10^f. Le vecteur contient N éléments.

4. Les Matrices

4.1. Création d'une matrice

Pour créer une matrice à partir de scalaires:

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

MAT =

     2     3     6
     4     5     6
     7     8     9
    10    11    12
```

A partir de vecteurs, soient vl, v2, v3 trois vecteurs:

Si vl, v2 et v3 sont des vecteurs lignes, on écrira:

```
>> mat=[v1;v2;v3]
```

Si vl, v2 et v3 sont des vecteurs colonnes:

```
>> mat=[v1 v2 v3]
```

A partir d'autres matrices :

```
>> mat=[MAT MAT]

mat =

     2     3     6     2     3     6
     4     5     6     4     5     6
     7     8     9     7     8     9
    10    11    12    10    11    12
```

La matrice vide:

```
>> M=[]

M =

     []
```

Cette commande définit une matrice M ensemble vide.

Pour accéder à un élément de la matrice :

```
>> element=matrice(ligne,colonne)
```

Exemple:

```
>> ele=mat(3,4)

ele =

     7
```

Pour extraire un vecteur de la matrice :

- *Vecteur colonne :*

```
>> vcol=mat(v1,c)
```

c est le numéro de la colonne à extraire, $v1$ un sous-ensemble des indices de ligne de mat.

Exemple:

```
>> vcol=mat([1:3],3)

vcol =

     6
     6
     9
```

- *Vecteur ligne :*

```
>> vlig=mat(l,v2)
```

l est le numéro de la ligne à extraire, $v2$ un sous-ensemble des indices de colonne de mat

Exemple:

```
>> vlig=mat(2,[3:6])

vlig =

     6     4     5     6
```

Pour accéder à une sous-matrice :

```
>> sousMat=mat([2 3],[2:4])

sousMat =

     5     6     4
     8     9     7
```

Les fonctions suivantes permettent de créer des matrices usuelles:

Fonction	Exemple	Signification

ones(m,n)	`>> ones(2,4)`
	`ans =`
	`1 1 1 1`
	`1 1 1 1`
		Définit une matrice de dimension $m \times n$ avec tous les éléments égaux à 1.

zeros(m,n)	`>> zeros(2,4)`
	`ans =`
	`0 0 0 0`
	`0 0 0 0`
		Définit une matrice $m \times n$ avec tous les éléments égaux à 0.

eye(m,n)	`>> eye(3,3)`
	`ans =`
	`1 0 0`
	`0 1 0`
	`0 0 1`
		Définit une matrice identité de dimension $m \times n$.

diag(v)	`>> v=[1:5];diag(v)`
	`ans =`
	`1 0 0 0 0`
	`0 2 0 0 0`
	`0 0 3 0 0`
	`0 0 0 4 0`
	`0 0 0 0 5`
		Si v est un vecteur à n composantes, alors $diag(v)$ est une matrice carrée de dimension n qui porte sur la diagonale principale les cléments du vecteur v.

size(matrice)	`>> size(mat)`
	`ans =`
	`4 6`
		Définit un vecteur dont la première composante est le nombre de lignes de $matrice$ et la seconde son nombre de colonnes.

On trouvera également sous Matlab quelques fonctions matricielles élémentaires utilisables si M est une matrice carrée:

» det(M) : Déterminant de la matrice M.

» inv(M) : Inverse de la matrice M.

» expm(M) : Exponentielle de la matrice M.

» logm(M) : Logarithme népérien de la matrice M.

» sqrtm(M): Racine carrée de la matrice M.

» eig(M) : Vecteur colonne contenant les valeurs propres de la matrice M.

» var(M) : Vecteur ligne contenant la variance de chaque colonne de la matrice M

4.2. Opérations sur les matrices

Toutes les opérations applicables sur les vecteurs sont aussi valables pour les matrices:
Soit la matrice:

```
MAT =

     2     3     6
     4     5     6
     7     8     9
    10    11    12
```

La transposée (matrice que l'on obtient en effectuant la permutation des lignes et des colonnes d'une autre matrice) de MAT est:

```
>> MAT'

ans =

     2     4     7    10
     3     5     8    11
     6     6     9    12
```

Pour additionner des matrices, elles doivent avoir mêmes dimensions:

```
>> A=[1 2;3 4];B=A;C=A+B

C =

     2     4
     6     8
```

Pour multiplier une matrice A de m lignes et k colonnes avec une matrice B de k lignes et n colonnes le résultat est C de m lignes et n colonnes, alors l'opération A*B (A*B≠B*A):

```
>> A=[1,2,3;4,5,6]

A =

     1     2     3
     4     5     6
```

```
>> B=[1,2;3,4;5,6]

B =

     1     2
     3     4
     5     6
```

```
>> C=A*B

C =

    22    28
    49    64
```

Il existe un autre produit matriciel, qui multiplie chaque élément la matrice A par l'élément correspondant de la matrice B. A et B ayant les mêmes dimensions:

```
>> A=[1,2,3;4,5,6];B=A;P=A.*B

P =

     1     4     9
    16    25    36
```

L'opérateur ^ est utilisé pour élever une matrice carrée à une puissance n :

`>> A^n` : Correspond à A^n. Cette opération est définie si A est une matrice carrée et si n est un scalaire.

`>> A.^n` : Correspond à une élévation en puissance, élément par élément de A.

Exemple:

```
>> A=[1,2;4,5];A^2

ans =

     9    12
    24    33
```

```
>> A.^2

ans =

     1     4
    16    25
```

Les fonctions suivantes opèrent sur les colonnes d'une matrice (ou sur les éléments d'un vecteur):

Soient A une matrice de 2×2 éléments et $vect$ un vecteur ligne de 7 éléments:

```
A =

     1     2
     4     5
```

```
vect =

     4     5     6     7     8     9    10
```

Fonction	Exemple	Signification

max()	`>> max(A)`
	`ans =`
	`4 5`
	`>> max(vect)`
	`ans =`
	`10`
		Définit un vecteur contenant la valeur maximale de chaque colonne pour une matrice, ou fournit l'élément le plus grand pour un vecteur.

mean()	`>> mean(A)`
	`ans =`
	`2.5000 3.5000`
	`>> mean(vect)`
	`ans =`
	`7`
		Définit un vecteur contenant la valeur moyenne (moyenne arithmétique) calculée sur chaque colonne d'une matrice, ou calcule la valeur moyenne d'un vecteur.

sort()	`>> sort(A)`
	`ans =`
	`1 2`
	`4 5`
	`>> sort(vect)`
	`ans =`
	`4 5 6 7 8 9 10`
		Classe les éléments de chaque colonne d'une matrice selon l'ordre croissant, ou classe les éléments d'un vecteur en ordre croissant.

sum()	`>> sum(A)`
	`ans =`
	`5 7`
	`>> sum(vect)`
	`ans =`
	`49`
		Définit le vecteur contenant la somme calculée sur chaque colonne d'une matrice, ou calcule la somme de tous les éléments d'un vecteur.

std()	`>> std(A)`
	`ans =`
	`2.1213 2.1213`
	`>> std(vect)`
	`ans =`
	`2.1602`
		Définit le vecteur ligne contenant l'écart type calculé sur chaque colonne d'une matrice, ou calcule l'écart type de tous les éléments d'un vecteur.

Exercice:
Soit le vecteur ligne $x = [2,4,6,8,5]$;
Calculer sa moyenne $\boldsymbol{x\_}$ et son écart type $\boldsymbol{s}$?

Solution:
```
x=[2,4,6,8,5]
x_=mean(x)
s=std(x)
```

Résultats:
```
x_ =

     5


s =

    2.2361
```

4.3. Fonctions mathématiques prédéfinies

Comme les autres langages de programmation MATLAB fournit un ensemble de fonctions mathématiques prédéfinies, parmi lesquelles:

Fonction	Signification
» sin(A)	Sinus des cléments de la matrice A.
» cos(A)	Cosinus des éléments de la matrice A.
» tan(A)	Tangente des éléments de la matrice A.
» asin(A)	Arc sinus des éléments de la matrice A.
» acos(A)	Arc cosinus des éléments de la matrice A.
» atan(A)	Arc tangente des éléments de la matrice A.

» abs(A)	Valeur absolue des éléments de la matrice A.
» sqrt(A)	Racine carrée des éléments de la matrice A.
» exp(A)	Exponentielle des éléments de la matrice A.
» log(A)	Logarithme népérien des éléments de la matrice A.

5. Les Boucles

Matlab possède des instructions similaires à celles que l'on trouve dans la majorité des langages de programmation (comme Fortran, C, Pascal, ...etc.).

5.1. La Boucle FOR

Matlab possède sa propre version de la boucle For, cette boucle permet à une instruction, ou à un bloc d'instructions d'être répété un nombre fixe ou déterminé de fois. La boucle For doit impérativement se terminer par le mot-clé end.

La syntaxe de la boucle For est la suivante :

```
for compteur = expression
    instruction;
end
```

Exemple :

```
m=input('Saisir le nombre de lignes:');
n=input('Saisir le nombre de colonnes:');
for i= 1: m
 for j = 1: n
     A(i,j)=i*j;
 end
end
disp(A)
```

A pour effet de créer une matrice A de taille $m\ x\ n$ dans laquelle chaque élément A_{ij} est une expression $i * j$ de ses coordonnées.

Exercice:
La même question de l'exercice à la fin de la ***section 4.2.*** mais cette fois on n'utilise pas les fonctions prédéfinies $mean()$ et $std()$?

NB. En s'aidant des équations suivantes:

La moyenne arithmétique	L'écart type
$\bar{x} = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i$	$\sigma = \sqrt{\frac{1}{n-1} \cdot \sum_{i=1}^{n} (x_i - \bar{x})^2}$

5.2. La Boucle WHILE

Comme la boucle For, la boucle $While$ permet de répéter une instruction ou un groupe d'instructions un certain nombre de fois mais cette fois sous la contrainte d'une condition logique.

La syntaxe de la boucle $While$ est la suivante :

```
while expression
   instructions;
end
```

L'instruction est répétée tant que les éléments contenus dans la matrice expression sont non égaux à zéro. La matrice expression est généralement un scalaire (dimension 1) de sorte que non égal à zéro correspond à la valeur 1 (vrai).

Par exemple :

```
  n= 1;
while prod(1:n)< 1.e100
   n = n+1;
end
disp(n)
```

La boucle de ce script recherche et affiche le premier entier n tel que $n! > 10^{100}$.

5.3. Les fonctions any et all

Si expression n'est pas un scalaire, elle peut être réduite en utilisant les fonctions prédéfinies any & all.

5.3.1. La fonction any

Cette fonction détermine les colonnes (éléments) non nuls d'une matrice (vecteur).

Syntaxe:

$$any(A)$$

$$any(A, dim)$$

Si A est un vecteur, $any(A)$ retourne la valeur logique 1 (true) si au moins un élément de A est non nul ou est une valeur logique (booléenne) 1 (true), et retourne la valeur logique 0 (false) si tous les éléments de A sont nuls.

Exemple:

```
>> v=[1,0,0];any(v)

ans =

     1
```

```
>> v=[0,0,0];any(v)

ans =

     0
```

Si A est une matrice, $any(A)$ traite les colonnes de A comme vecteurs, retournant un vecteur ligne de valeurs logiques (des 1 et des 0).

Exemple:

```
>> A=[1,0,0;4,5,0;0,0,0];any(A)

ans =

     1     1     0
```

$any(A, dim)$ teste le long d'une dimension (spécifiée par le scalaire dim) de A.

Exemple:

```
>> A=[0,0,0;4,5,0;0,0,0];any(A,1)

ans =

     1     1     0
```

```
>> A=[0,0,0;4,5,0;0,0,0];any(A,2)

ans =

     0
     1
     0
```

5.3.2. La fonction *all*

Détermine si tous les éléments de chaque colonne d'une matrice (vecteur) sont non nuls.

Syntaxe:

$$all(A)$$

$$all(A, dim)$$

$all(A)$ teste, le long des différentes dimensions d'une matrice, si tous les éléments sont non nuls ou corresponds à la valeur logique 1 (true).

Si A est vecteur, $all(A)$ retourne la valeur logique 1 (true) si tous ses éléments sont non nuls et retourne la valeur logique 0 (false) si un élément ou plus sont nuls.

Exemple:

```
>> v=[0,0,0];all(v)

ans =

     0
```

```
>> v=[1,4,3];all(v)

ans =

     1
```

Si A est une matrice, $all(A)$ traite les colonnes de A comme vecteurs, retournant un vecteur ligne de valeurs logiques 1 et 0.

Exemple:

```
>> A=[0,0,0;4,5,0;0,0,0];all(A)

ans =

     0     0     0
```

La commande $all(A, dim)$ teste le long d'une dimension de A spécifiée par le scalaire dim.

Exemple:

```
>> A=[3,0,0;4,5,0;1,0,0];all(A,1)

ans =

     1     0     0
```

```
>> A=[3,0,0;4,5,0;1,0,0];all(A,2)

ans =

     0
     0
     0
```

5.4. Instructions IF et BREAK

La syntaxe de l'instruction conditionnelle If est la suivante:

```
if expression1
    statements1
elseif expression2
    statements2
else
    statements3
end
```

L'exemple suivant illustre l'instruction if dans MATLAB :

```
% Ce Script affiche le signe d'un nombre n.
n=input('Entrer la valeur de n:');
if n<0
 disp(' n est négatif')
elseif n == 0
 disp('n est nul')
else
 disp(' n est positif ')
end
```

L'instruction break permet de sortir d'une boucle en cas de nécessité. Par exemple:

```
while 1
 n=input('Entrer une valeur positive :');
 if n<=0
  disp('Strictement positive!');
 else
  disp('Sortir!!')
  break
 end
end;
```

Le résultat d'exécution de ce script:

```
Entrer une valeur positive :-6
Strictement positive!
Entrer une valeur positive :7
Sortir!!
```

Remarque :

L'instruction $disp$ permet d'afficher un message textuel à l'écran. L'instruction $input$ permet d'afficher un message textuel et de rentrer une valeur au clavier. Le caractère ; à la fin de chaque instruction évite que MATLAB affiche le résultat de l'opération en cours.

6. Scripts et Fonctions

On peut utiliser Matlab soit en mode ligne (actif) : lorsqu'on entre une commande au prompt » Matlab l'exécute immédiatement et affiche le résultat, soit en mode passif dans lequel Matlab peut exécuter des séquences de commandes écrites dans des fichiers dont l'extension est .m. Un fichier .m (script ou fonction) est composé d'un bloc d'instructions et inclut la possibilité de faire appel à d'autres fichiers .m. Les fichiers .m sont créés avec l'éditeur intégré de Matlab ou par un éditeur de texte comme notepad.

6.1. Les Scripts

Les Scripts (dites aussi Sous-programmes) sont des séquences de commandes plus ou moins longues. Lorsqu'un Script est appelé Matlab exécute les commandes qu'il a lu dans son fichier.

Exemple:

Le fichier $essai.m$ contient les commandes suivantes :

```
% Un script qui calcule n racines carrées:
n=input('Donner n :');
for i=1 :n
  while 1
   nbr=input('Donner un nombre :');
   if nbr>=0
    break;
   else
    disp('Le nombre doit être strictement positif!');
   end;
  end;
disp('la racine carrée de : ');disp(nbr);disp(' = ');disp(sqrt(nbr));
end
```

En tapant le nom essai au prompt »_ Matlab exécute les instructions lues dans $essai.m$: il calcule la racine carrée de n nombres.

Remarque:

Les commentaires destinés à expliquer les commandes (qui ne sont pas considérés par Matlab) sont insérés à l'aide du symbole : %.

6.2. Les Fonctions

Les fichiers Fonctions permettent d'ajouter de nouvelles fonctions à celles qui existent déjà dans la bibliothèque de Matlab. La possibilité de créer de nouvelles fonctions représente une puissance du langage Matlab pour résoudre des problèmes spécifiques.

Le mot-clé d'un fichier fonction est $function$ écrit au début de la première ligne de ce fichier.

La syntaxe d'une fonction sous Matlab est la suivante:

```
function resultat=nom_fonction(parametres)
  resultat=expression;
end
```

Par exemple le fichier $f.m$ qui représente un fichier Fonction contient les instructions suivantes :

```
%Une fonction qui calcule l'expression f(x,y)=exp(x)+sinus(y)
function res=f(x,y)
  res=exp(x)+sin(y);
end
```

Remarque: Le fichier fonction doit porter le même nom que la fonction.

La nouvelle fonction $f(x,y)$ est utilisée comme les autres fonctions prédéfinies de Matlab.

Un exemple d'exécution de cette fonction est:

```
>> f(0,0)

ans =

     1
```

```
>> f(0,pi)

ans =

    1.0000
```

```
>> f(0,pi/2)

ans =

     2
```

On peut déclarer la même fonction avec plusieurs paramètres de sortie:

```
%Une fonction qui calcule l'expression g(x,y)=exp(x)-sinus(y)
% ainsi que x puisssance y :
function [resultat1 ,resultat2]=g(x,y)
 resultat1=exp(x)-sin(y)
 resultat2=x^y
end
```

Une exécution de cette fonction est la suivante:

```
>> g(2,3)

resultat1 =

    7.2479


resultat2 =

     8
```

Les arguments (paramètres) d'entrée sont x, y et les arguments de sortie sont resultat1 & resultat2.

7. Graphiques Sous Matlab

Matlab dispose d'une grande variété de techniques sophistiquées pour présenter et visualiser des données : tracés 2D, tracés 3D, conteur 2D et 3D, représentations en fausses couleurs, cartes de couleurs...etc.

7.1. Graphiques 2D

Le graphisme est basé sur la façon dont Matlab traite les matrices. Pour tracer un ensemble de points (x,y). x et y étant de même dimension on utilise les instructions suivantes :

» $plot(x)$

Trace un graphique des éléments de x en fonction de leur indice dans le vecteur.

» $plot(x, y)$

Trace un graphique des éléments de y en fonction des éléments de x.

Il est possible de spécifier les différents types de lignes et de couleurs comme arguments de plot. Par exemple, l'instruction suivante :

» $plot(x, y1, 'r-', x, y2, 'g--')$

Trace un graphique sur lequel sont représentées deux courbes : l'une des éléments de y1 en fonction des éléments de x et l'autre des éléments de y2 en fonction de x. Les instructions supplémentaires 'r-' et 'g- -' indiquent que la première courbe est tracée par une ligne rouge continue et que la seconde est tracée par une ligne verte en tiret.

Les chaînes de caractères peuvent être construites à l'aide des caractères répertoriés dans les deux tableaux suivants :

Color Specifiers

Specifier	Color
r	Red
g	Green
b	Blue
c	Cyan
m	Magenta
y	Yellow
k	Black
w	White

Line Style Specifiers

Specifier	Line Style
-	Solid line (default)
--	Dashed line
:	Dotted line
-.	Dash-dot line

Il est également possible d'ajouter une courbe à un graphique existant par l'intermédiaire de la commande $hold$. L'instruction :

» $hold\ on$

Autorise l'ajout d'un nouveau graphique sans effacement des courbes existantes, et ajuste les échelles en fonction des nouvelles données.

» $hold\ off$

Est l'instruction contraire à la précédente : les nouveaux tracés effaceront les précédents. L'échelle des graphiques est automatique, il est néanmoins possible de changer ses valeurs :

» $axis([Xmin\ Xmin\ Ymin\ Ymin])$

D'autres commandes permettent de commenter ou de compléter un graphique:

» $xlabel('Axe\ des\ x')$: ajoute une étiquette sous l'axe des x.

» $ylabel('Axe\ des\ y')$: ajoute une étiquette sous l'axe des y.

» $title\ ('Titre\ au-dessus\ du\ graphique')$

Pour une représentation graphique adaptée à l'écran il est également possible de choisir le type d'échelle désiré pour les axes x et y avec les fonctions suivantes :

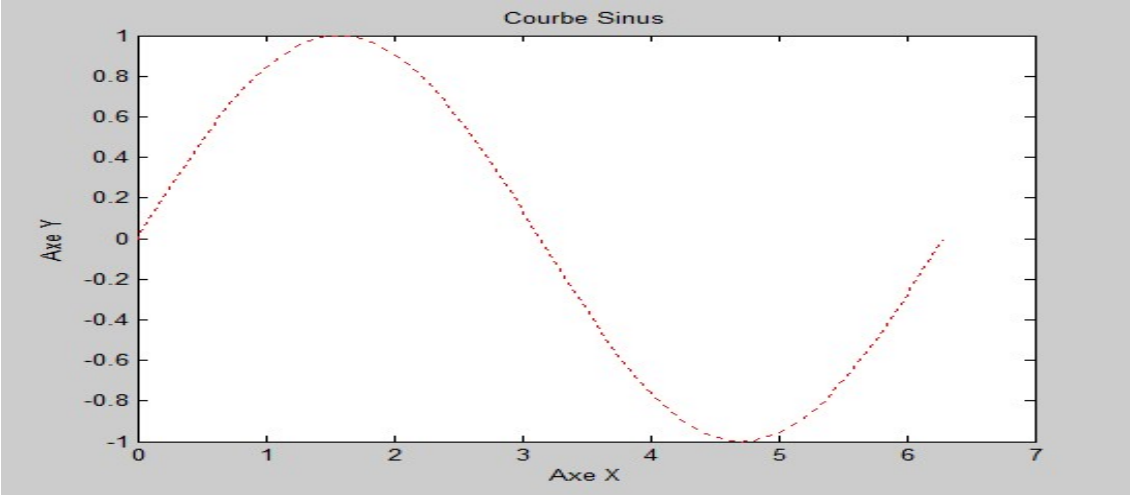
plot	Graphe linéaire en x et y
loglog	Graphe logarithmique en x et y
semilogx	Graphe logarithmique en x, linéaire en y
semilogy	Graphe linéaire en x logarithmique en y

Exemples:

Un script traçant la courbe de la fonction *Sinus* est le suivant:

```
x=[0:pi/100:2*pi]
y=sin(x)
plot(x,y,'r:')
title('Courbe Sinus')
xlabel('Axe X');ylabel('Axe Y')
```

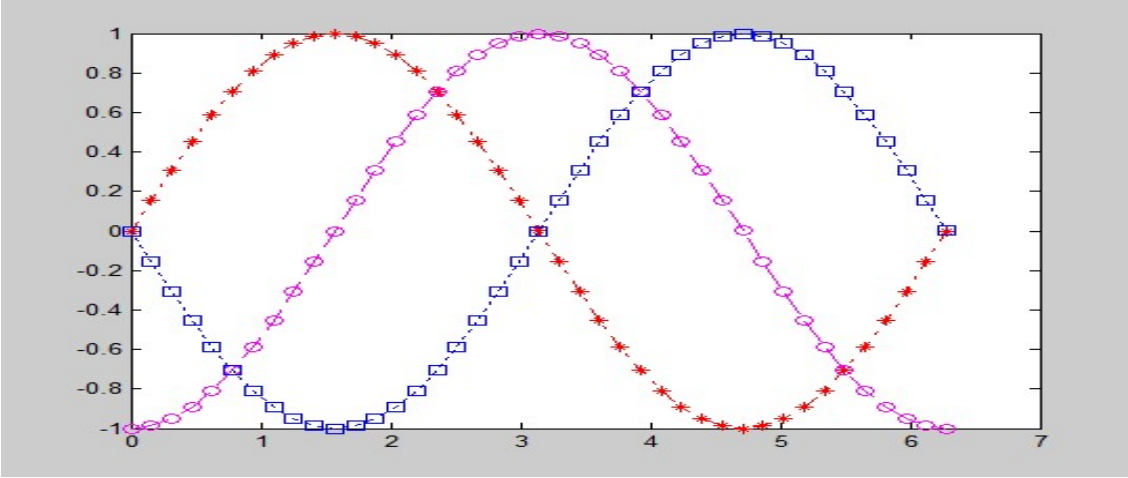
Le résultat de ce script est:



Un exemple de script qui trace plusieurs courbes avec marqueurs différents sur le même graphique:

```
t = 0:pi/20:2*pi;
plot(t,sin(t),'-.r*')
hold on
plot(t,sin(t-pi/2),'--mo')
plot(t,sin(t-pi),':bs')
hold off
```

Ce script trace la fonction *Sinus* à travers 03 différents intervalles utilisant différents styles de ligne, couleurs, et marqueurs (markers):



Les marqueurs disponibles sous Matlab sont:

Marker Specifiers

Specifier	Marker Type
+	Plus sign
o	Circle
*	Asterisk
.	Point (see note below)
x	Cross
'square' or s	Square
'diamond' or d	Diamond
^	Upward-pointing triangle
v	Downward-pointing triangle
>	Right-pointing triangle
<	Left-pointing triangle
'pentagram' or p	Five-pointed star (pentagram)
'hexagram' or h	Six-pointed star (hexagram)

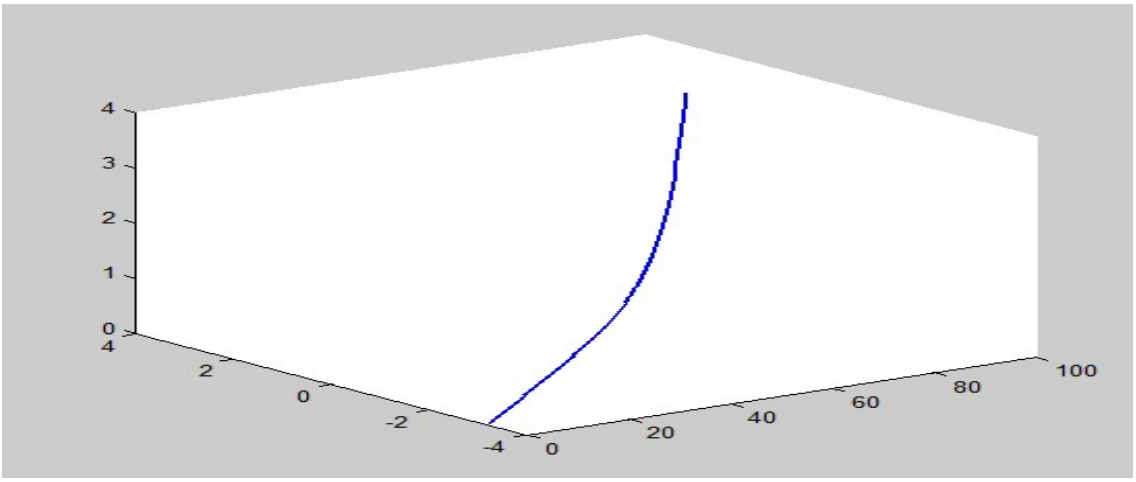
7.2. Graphiques 3D

Matlab dispose aussi des fondions pour représenter des graphiques en 3D. Certaines tracent les lignes en 3D, d'autres tracent des surfaces en utilisant des fausses couleurs en créant une quatrième dimension.

Soient v1, v2 et v3 trois vecteurs (ayant la même dimension):

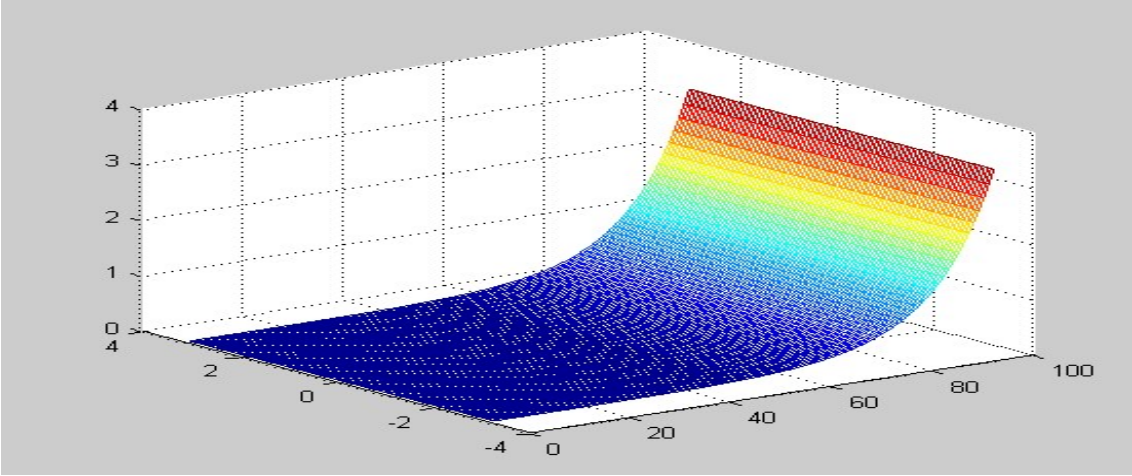
```
v1=[1,100];
v2=linspace(-pi,pi,100);
v3=logspace(-pi,pi,100);
plot3(v1,v2,v3,'b','LineWidth',2)
```

La commande $plot3$ trace une ligne continue dans l'espace dont les coordonnées sont les éléments des vecteurs v1, v2 et v3. Le résultat de $plot3$ est la suivante:



A l'aide de la fonction **mesh**. Matlab définit une surface quadrillée par les coordonnées v3 en fonction des coordonnées d'une grille rectangulaire contenue dans le plan v1-v2 ($mesh(x, y, z)$ définit une surface quadrillée avec des couleurs déterminées par z donc les couleurs sont proportionnelles à la hauteur de la surface. Si x et y sont des vecteurs, $longueur(x) = n$ et $longueur(y) = m$ alors $size(z) = [m, n]$):

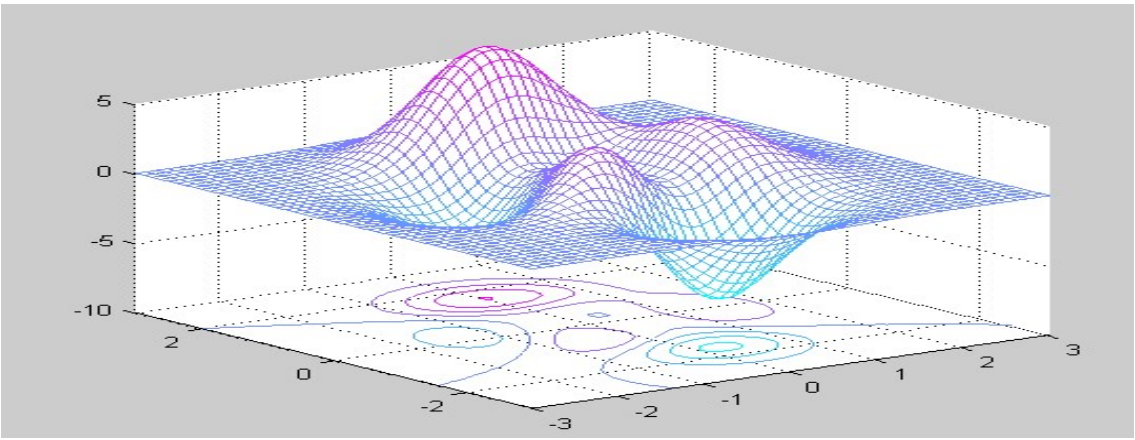
```
v1=[1:100];
v2=linspace(-pi,pi,100);
v=logspace(-pi,pi,100);
% we can use also :
% v3=meshgrid(logspace(-pi,pi,100));
v3=[]
for i=1:100
    v3=[v3;v];
end
mesh(v1,v2,v3)
```



Exemples:

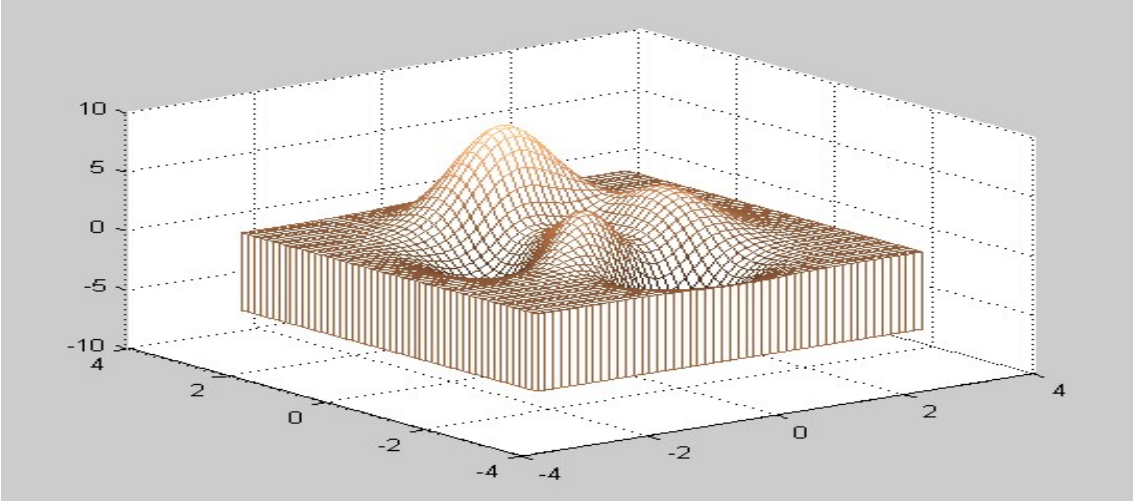
The command *meshc* produces a combination mesh and contour plot of the peaks surface:

```
[X,Y] = meshgrid(-3:.125:3);
Z = peaks(X,Y);colormap(cool)
meshc(X,Y,Z);
axis([-3 3 -3 3 -10 5])
```



The command meshz generates the curtain plot for the peaks function:

```
[X,Y] = meshgrid(-3:.125:3);
Z = peaks(X,Y);colormap(copper)
meshz(X,Y,Z);
```



Remarques:

- Les commandes $mesh, meshc, and\ meshz$ n'acceptent pas les complexes comme entée.
- La commande $meshgrid$ génère des matrices pour des graphiques 3-D.

```
>> X = meshgrid(-2:1:2)

X =

    -2    -1     0     1     2
    -2    -1     0     1     2
    -2    -1     0     1     2
    -2    -1     0     1     2
    -2    -1     0     1     2
```

- $peaks$ est une fonction d'une ou de deux variables, elle est utile pour les commandes: $mesh, pcolor, contour$ ...etc.

```
>> z=peaks(3)

z =

    0.0001   -0.2450   -0.0000
   -0.0365    0.9810    0.0331
    0.0000    0.2999    0.0000
```

```
>> [X,Y] = meshgrid(-2:1:2)

X =

    -2    -1     0     1     2
    -2    -1     0     1     2
    -2    -1     0     1     2
    -2    -1     0     1     2
    -2    -1     0     1     2

Y =

    -2    -2    -2    -2    -2
    -1    -1    -1    -1    -1
     0     0     0     0     0
     1     1     1     1     1
     2     2     2     2     2
```

```
>> z=peaks(X,Y)

z =

    0.0468   -0.5921   -4.7596   -2.1024   -0.0616
   -0.1301    1.8559   -0.7239   -0.2729    0.4996
   -1.3327   -1.6523    0.9810    2.9369    1.4122
   -0.4808    0.2289    3.6886    2.4338    0.5805
    0.0797    2.0967    5.8591    2.2099    0.1328
```

7.3. D'autres possibilités graphiques de Matlab:

Les fonctions **contour -2D-** et **contour3 -3D-** génèrent des tracés composés de lignes de niveau de la matrice indiquée en argument. Le nombre de lignes de niveau est optionnel. Les commandes :

```
>> contour(v3,10,'r--')% Graphique 2D
>> contour3(v3,10,'r--')% Graphique 3D
```

Tracent 10 lignes (style en tiret et couleur rouge) de contour du contenu de v3 en 2D et 3D.
Les fonctions **pcolor** (i.e. pseudo-color) et **colormap** créent des tracés en fausses couleurs. Les instructions :

```
>> pcolor(v3);colormap(gray)
```

Créent un tracé en fausses couleurs dans lequel les valeurs de v3 sont utilisées par la table des couleurs définie par **colormap** pour déterminer une couleur et l'afficher à l'écran. La table des couleurs choisie (gray) indique que les valeurs de v3 seront représentées par 255 niveaux de gris. D'autres **colormap** sont disponibles: $hot, hsv, cool, copper\ et\ pink$.

7.4. La fonction *subplot*:

Il est parfois utile de représenter plusieurs graphiques sur la même figure. La fonction *subplot* permet de diviser une figure en plusieurs petits graphiques. La syntaxe est la suivante:

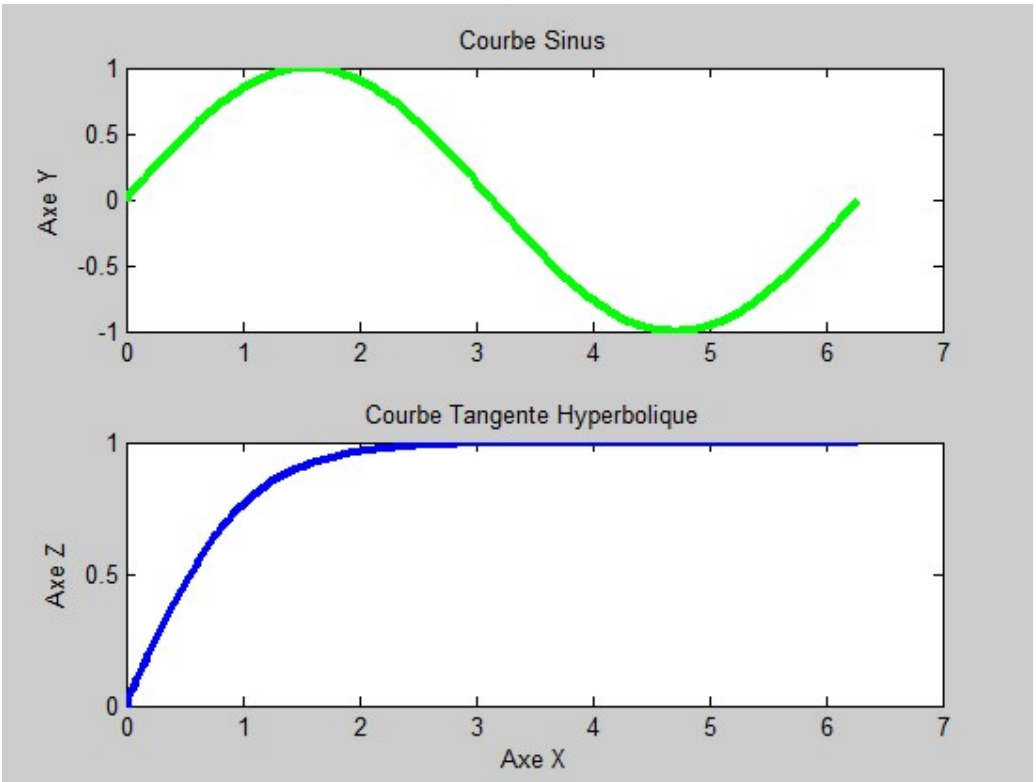
$subplot(m,n,p)$, m et n définissent le nombre $m \times n$ de graphiques dans la figure et p sélectionne le $p^{ème}$ graphique pour le tracé en cours.

La commande figure permet d'établir une nouvelle figure: **figure(N)** représente la $N^{ème}$ figure avec tous ses tracés (courbes).

Reprenons l'exemple graphique 2D précédent (§7.1) et ajoutons la courbe $z = tanh(x)$:

```
x=[0:pi/100:2*pi];
y=sin(x);
z=tanh(x);
subplot(2,1,1);
%LineWidth: Specifies the width of the line (here 3 points).
plot(x,y,'g-','LineWidth',3);
title('Courbe Sinus');
ylabel('Axe Y');
subplot(2,1,2);
plot(x,z,'b-','LineWidth',3);
title('Courbe Tangente Hyperbolique');
xlabel('Axe X');
ylabel('Axe Z');
```

Ce script affiche la figure suivante (avec deux courbes):



8. Les Fichiers

Les fichiers sont des objets destinés à stocker des données recueillies dans un format quelconque ou d'enregistrer des donnés générées par Matlab.

8.1. Ouverture et fermeture de fichiers

Pour ouvrir un fichier nous utilisons la commande fopen en spécifiant de plus le mode d'ouverture (les autorisations de traitement). Par exemple la commande :

```
>> fichier=fopen('courbe.m','r')
```

Ouvre le fichier $courbe.m$ pour lecture seule, $fichier$ est le fichier logique (i.e. pointeur) attribué au fichier physique $courbe.m$. Matlab considère $fichier$ comme un argument pour identifier le fichier physique, écrire ou fermer. Si un problème apparaît à l'ouverture du fichier physique (e.g. fichier inexistant) la valeur de $fichier$ est -1.

Les différents modes d'ouverture d'un fichier sont:

Mode	Autorisation
'r'	Lecture seule
'w'	Suppression et écriture
'a'	Ajouter à la fin
'r+'	Lecture et écriture

Une fois ouvert le fichier peut être lu ou écrit. La fermeture du fichier s'effectue par la commande suivante :

```
>> fclose(fichier)
```

L'instruction suivante ferme tous les fichiers ouverts:

```
>> fclose('all')
```

8.2. Lecture et écriture de fichiers textes formatés ou de chaînes

Matlab utilise la fonction $fscanf$ pour lire les fichiers textes, $fscanf$ a pour arguments le fichier logique ouvert et une chaîne de contrôle de format contenant les spécifications de la conversion. Les spécifications de la conversion commencent par le caractère % et en général elles comprennent:

- $\%s$ pour reconnaître une chaîne;
- $\%d$ pour reconnaître un nombre décimal;
- $\%e, \%f, \%g$ pour reconnaître une virgule flottante.

Exemples:

Soit le script suivant:

```
file= fopen('cosinus.txt','r');
titre = fscanf(file,'%s',[2])%reads two strings
[table,compte] = fscanf(file,'%f %f',[2 inf]);
% It has two rows now, then we transpose to get de desired result
table=table',compte
fclose(file);
```

Le premier appel à $fscanf$ reconnaît le titre du fichier grace à la specification $\%s$. Le second appel charge le fichier lu dans la matrice $table$ en reconnaissant deux valeurs virgules flottantes jusqu'à ce qu'il arrive à sa fin. La valeur $compte$ indique le nombre de valeurs lues. Un argument optionnel de la fonction $fscanf$ peut être utilisé pour indiquer le nombre de valeurs à lire, la commande suivante lit 20 valeurs réelles et les charge dans le vecteur colonne $vcol$:

```
>> vcol=fscanf(file,'%g',20)
```

Un autre exemple de script:

```
file = fopen('cosinus.txt');
while 1
    line = fgetl(file);
    if ~ischar(line)
        break
    end
    disp(line)
end
fclose(file);
```

Ce script nous permet de lire un fichier texte et de l'afficher au $Command\ Window$.

En écriture, la fonction $fprintf$ convertit des données en chaînes de caractères et les écrit dans un fichier. Des spécifications de conversion des données indiquent le format des données écrites. Les spécifications commencent par le caractère %. Les conversions classiques sont les suivantes :

- $\%e$ pour la notation exponentielle
- $\%f$ pour la notation virgule fixe

- $\%g$ pour une sélection automatique entre les deux notations précédentes

Exemple:

```
x =[-pi:0.1:pi];
y = [x;cos(x)];
file= fopen('cosinus.txt','a');
fprintf(file, 'Table Cosinus:\n');
fprintf(file, '%6.3f  %12.9f \n',y);
fclose(file);
```

Ces instructions créent un fichier texte appelé $cosinus.txt$ contenant une table de la fonction cosinus. Le premier appel à la fonction $fprintf$ crée un titre suivi d'un retour à la ligne indiqués par $\backslash n$. Le second appel crée la table dans le fichier.

La chaîne `'%6.3f  %12.9f \n'` (dite chaine de formatage) spécifie le format de chaque ligne du fichier :

- Une valeur avec point fixe de 6 caractères au total dont 3 décimaux;
- Une valeur avec virgule fixe de 12 caractères au total dont 9 décimaux;
- Ces deux valeurs sont séparées par des espaces.

Voici un aperçu du fichier $cosinus.txt$:

```
Table Cosinus:
-3.142  -1.000000000
-3.042  -0.995004165
-2.942  -0.980066578
-2.842  -0.955336489
-2.742  -0.921060994
-2.642  -0.877582562
-2.542  -0.825335615
-2.442  -0.764842187
-2.342  -0.696706709

 2.958  -0.983268438
 3.058  -0.996542097
```

8.3. Les fonctions *save* et *load*

Les fonctions $save$ et $load$ permettent de sauvegarder et de lire des données dans un fichier texte dont l'extension est **.txt** (données en format ascii).

Considérons les instructions suivantes :

```
vlig = [1,2,3,4,5,6];
save vectors.txt vlig -ascii
```

La première commande crée le vecteur $vlig$ et la seconde sauvegarde les données qu'il contient dans le fichier $vectors.txt$ en format ascii (texte).

Pour lire les données contenues dans $vectors.txt$ on doit taper :

```
>> load vectors.txt -ascii
```

On regarde maintenant les variables en mémoire en tapant:

```
>> whos
  Name          Size              Bytes  Class     Attributes

  vectors       1x6                  48  double
```

La variable $vectors$ contient maintenant les données de $vlig$:

```
>> vectors

vectors =

     1     2     3     4     5     6
```