Introduction à Matlab[1]

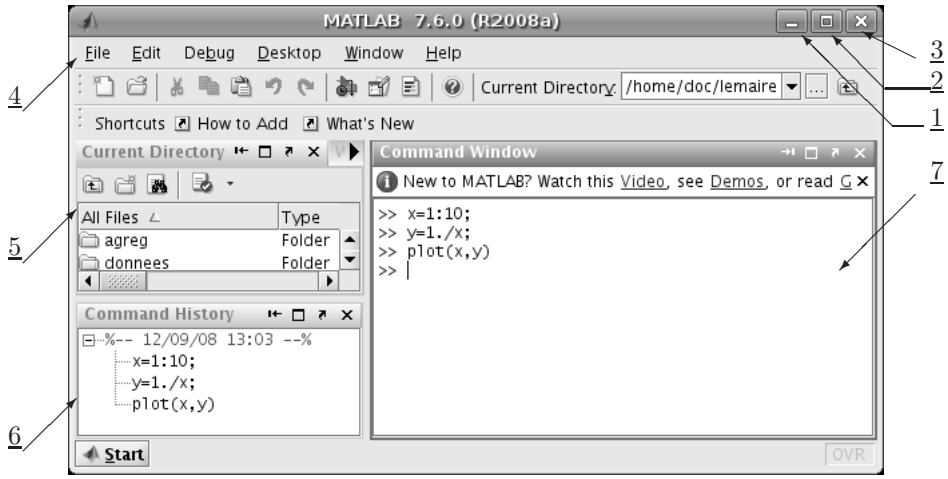
8 Janvier 2009

1 Les caractéristiques principales

- MATLAB qui est l'abréviation de MATrix LABoratory est un langage dont l'élément de base est la matrice
- C'est un langage interprété
- Il n'est pas nécessaire de faire des déclarations de variables, ni de préciser la taille des matrices utilisées

2 L'accès

La version de Matlab accessible sur les ordinateurs est la version 7.6.0. Vous pouvez l'obtenir en cliquant sur son icône.



1. bouton pour mettre la fenêtre en icône (cliquer dessus avec le bouton droit de la souris) ; l'icône de la fenêtre est mise dans la barre en bas de l'écran, pour réouvrir la fenêtre, cliquer sur l'icône.
2. bouton d'agrandissement-réduction : permet de mettre la fenêtre en plein écran.
3. bouton pour "tuer" une fenêtre (à éviter : utiliser le menu file pour quitter Matlab)
4. barre de menus : le menu *file* permet d'ouvrir des fichiers textes dans un éditeur.
5. *Current directory* : gestionnaire de fichiers, permettant de choisir le répertoire courant de Matlab et d'ouvrir un fichier dans l'éditeur de texte de Matlab. *Workspace* (caché sur la figure) : donne la liste des variables en mémoire.
6. zone donnant l'historique des commandes tapées par l'utilisateur dans la fenêtre de commandes.
7. Fenêtre servant à taper les commandes.

3 L'environnement

► Une fenêtre de commandes (fenêtre ci-dessus) dans laquelle on peut taper des commandes, qui seront exécutées les unes après les autres. Une ligne de commandes est exécutée lorsqu'on tape sur la touche `enter`. Le caractère '`>>`' signifie que l'ordinateur attend une instruction.

Exemple :

```
>> quit
```

est la commande qui permet de quitter Matlab.

NB : Taper en même temps sur les touches `Ctr` et `c` arrête l'exécution d'une commande Matlab.

[1] Cette introduction est écrite pour la version 7.6.0 de Matlab dans l'environnement Linux Ubuntu

`clc`	efface le contenu de la fenêtre de commandes
touches `Ctr` + `c`	arrête l'exécution d'une commande Matlab.
$\rightarrow, \leftarrow, \uparrow\downarrow$	permet de se déplacer dans les lignes de commandes tapées dans la fenêtre de commandes.
`%`	caractère qui marque le début d'un commentaire (la suite de la ligne n'est pas exécutée par Matlab).

► Un espace de travail qui garde en mémoire les variables que vous avez créées et les lignes de commandes que vous avez exécutées dans la fenêtre de commandes.

`pwd`	affiche le nom du répertoire courant pour Matlab
`cd` *rep*	change le répertoire courant pour Matlab qui devient *rep*
`who`	donne la liste des variables présentes dans l'espace de travail
`whos`	donne la liste des variables présentes dans l'espace de travail ainsi que leurs propriétés
`what`	donne la liste des fichiers `.m` et `.mat` présents dans le répertoire courant
`clear` var_1 ... var_n	efface les variables $var_1, \ldots, var_n$ de l'espace de travail
`clear`	efface toutes les variables créées dans l'espace de travail
`save` *nom-f* var_1 ... var_n	sauve la valeur des variables var_1, ..., var_n dans un fichier binaire *nom-f.mat*
`save` *nom-f*	sauve l'ensemble des variables, existant dans l'espace de travail de Matlab, dans un fichier binaire *nom-f.mat*
`save` -ASCII *nom-f* var_1 .. var_n	sauve la valeur des variables $var_1, ..., var_n$ dans un fichier ascii *nom-f.*
`load` *nom-f*	permet de récupérer toutes les données sauvegardées dans le fichier *nom-f.mat*
`addpath` *chemin-repertoire*	permet d'utiliser des fichiers `.m`, `.fig` et `.mat` de ce nouveau répertoire en plus de ceux du répertoire courant.

Exemple :

```
 >> pwd
ans =
/home/lemaire/etudiant

 >> clear
 >> u=(2+i)^2
 u =
    3.0000 + 4.0000i

 >> v=1:10
 v =
     1     2     3     4     5     6     7     8     9    10
 >> whos
 Name      Size            Bytes  Class

   u         1x1                16  double array (complex)
   v         1x10               80  double array

 Grand total is 11 elements using 96 bytes

 >> save fich
 >> what
 MAT-files in the current directory /home/lemaire/etudiant
 fich
```

```
>> clear, m='message'; whos
  Name      Size            Bytes  Class

  m         1x7                14  char array

Grand total is 7 elements using 14 bytes

>> load fich, whos
Name      Size            Bytes  Class

  m         1x7                14  char array
  u         1x1                16  double array (complex)
  v         1x10               80  double array

Grand total is 18 elements using 110 bytes
```

NB : Lorsqu'on termine une commande par un point-virgule, le résultat de la commande n'est pas affiché. Pour écrire plusieurs commandes dont on veut voir les résultats, sur une même ligne, il suffit de les séparer par une virgule.
NB : On peut changer le répertoire courant de Matlab et visualiser la liste des fichiers de ce répertoire en utilisant la zone *current Directory* en haut à gauche de la fenêtre de Matlab.
En utilisant le sous-menu *Set Path...* dans le menu *file* de la fenêtre Matlab ou en tapant la commande `path`, on peut visualiser l'ensemble des répertoires dont le contenu est visible par Matlab.

NB : On peut voir dans la zone *Workspace* en haut à gauche de la fenêtre Matlab, l'ensemble des variables créées.

- une fenêtre graphique qui peut être ouverte en tapant `figure` ou lorsqu'on exécute une commande affichant un graphique (`plot, bar, hist,...`)

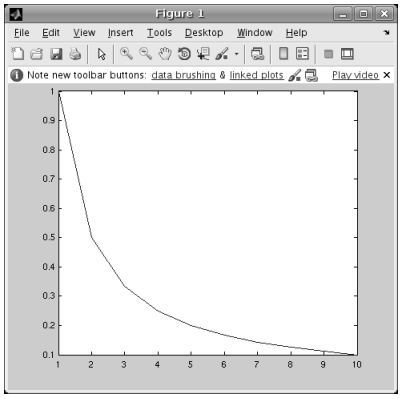


FIG. 1 – Fenêtre graphique contenant le tracé approché de $x \mapsto 1/x$ à partir du calcul de 10 points entre 1 et 10.

En utilisant les menus de la fenêtre graphique, on peut rajouter un titre, changer la couleur du tracé, changer la partie visible du graphique...

4 L'aide en ligne

Pour trouver les fonctions prédéfinies dans Matlab afin d'effectuer une tache donnée ou pour savoir comment utiliser une commande Matlab, le bon réflexe est de consulter l'aide en ligne de Matlab.

`helpwin`	ouvre une fenêtre pour l'aide en ligne contenant la liste des commandes Matlab ainsi que leurs documentations
`help`	donne la liste de toutes les commandes par thème
`help` *nom*	décrit la fonction *nom.m*
`lookfor` *nom*	permet de rechercher une instruction à partir du mot clé *nom*; Cette instruction donne la liste des commandes dont la documentation contient le mot *nom*.

Exemple :

```
>> help abs
 ABS    Absolute value.
    ABS(X) is the absolute value of the elements of X. When
    X is complex, ABS(X) is the complex modulus (magnitude) of
    the elements of X.

    See also sign, angle, unwrap.

    Overloaded functions or methods (ones with the same name in other directories)
       help iddata/abs.m
       help sym/abs.m

    Reference page in Help browser
       doc abs
```

NB : Matlab fait la distinction entre les majuscules et les minuscules ; les commandes prédéfinies dans Matlab sont toujours en minuscules (contrairement à ce que laisse à penser l'aide en ligne).

5 Les types de données

Dans Matlab, il y a un seul type de données : le type MATRICE. Par exemple :
- un scalaire est une matrice 1×1,
- un vecteur (ligne ou colonne) est une matrice à une ligne ou une colonne,
- un polynôme est un vecteur ligne constitué des coefficients du polynôme entrés dans l'ordre des puissances décroissantes,
- une chaîne de caractères est une matrice à une ligne dont le nombre de colonnes correspond au nombre de caractères de la chaîne.

Rappelons que dans Matlab, il n'y a pas besoin de déclarer les variables ni leurs dimensions avant de les utiliser. Les matrices peuvent être construites de plusieurs façons :

▶ en entrant explicitement la liste des éléments :
- pour une matrice, on entre les éléments d'une ligne en les séparant par un blanc ou une virgule. Pour séparer les éléments de deux lignes, on met un point-virgule.

Exemple :

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

$A(i, j)$ renvoie à l'élément de la i-ème ligne et de la j-ème colonne d'une matrice A. On peut aussi définir une matrice A en entrant successivement les éléments $A(i, j)$.

Exemple :

```
>> A(2,2)
ans =
     2

>>  A(2,end)
ans =
     3
```

```
>> A(2,2)=0
A =
     4     5     6
     1     0     3
     7     8     9

>> v=A(2,[1,3])
v =
     1     3
```

NB : i et j sont nécessairement des entiers strictement positifs, `end` ici fait référence à l'indice du dernier élément d'une ligne ou d'une colonne.
NB : pour un vecteur, il suffit d'écrire $v(i)$ pour avoir le i-ème élément que le vecteur soit en ligne ou en colonne.

Exemple :

```
>> v(2)
ans=
     3
```

– pour une chaîne de caractères, on entre les caractères en commençant et en terminant par une quote '.

Exemple :
```
>> ch='Lisez attentivement les messages d''erreur'
ch =
Lisez attentivement les messages d'erreur
```
– pour un polynôme, on crée un vecteur ligne contenant les coefficients du polynôme entrés dans l'ordre des puissances décroissantes.

Exemple : Le polynôme $2x^2 + 3$ sera représenté par le vecteur
```
>> Q=[2 0 3];
```
► En utilisant des fonctions prédéfinies de Matlab.

`linspace`(x, y, n)	définit un vecteur ligne constitué de n nombres espacés régulièrement entre x et y
`deb :pas :fin`	définit un vecteur $x = [x(1), \ldots, x(n)]$ avec n le plus grand entier tel que $x(n) \leq$ `fin` et $x(i) =$`deb`$+(i-1)*$`pas` pour tout i
`zeros`(n, m)	crée une matrice de taille $n \times m$ dont tous les éléments sont nuls
`ones`(n, m)	crée une matrice de taille $n \times m$ dont tous les éléments valent 1
`eye`(n, m)	crée une matrice de taille $n \times m$ avec des 1 sur la diagonale et des zéros ailleurs.
`rand`(n, m)	crée une matrice de taille $n \times m$ dont les éléments sont tirés "au hasard" entre 0 et 1.

Avec Matlab, il n'y a pas besoin de déclarer les types des variables utilisées, ni de spécifier leur taille. On peut donc changer la taille d'une variable au cours d'un programme.

Exemple :

```
>>ch=[ch ' !!!']
ch =
Lisez attentivement les messages d'erreur !!!

>> A=[A ones(3,1)]
```

```
A =
     4     5     6     1
     1     0     3     1
     7     8     9     1
```

NB : Dans certains cas, il est utile de fixer au début d'un programme la taille des matrices que l'on va utiliser par exemple en utilisant la commande `zeros`.

$s =$`size`(A)	retourne un vecteur ligne dont le 1er élément est le nombre de lignes de A et le 2ème élément le nombre de colonnes de A.
$[l, c] =$`size`(A)	retourne deux scalaires, l étant le nombre de lignes de A et c étant le nombre de colonnes de A.
$x =$`size`(A, i)	retourne un scalaire x qui est le nombre de lignes de A si $i = 1$ et le nombre de colonnes de A si $i = 2$.
$l =$`length`(A)	retourne un scalaire l qui est la longueur de A si A est un vecteur et le maximum entre le nombre de lignes et le nombre de colonnes si A est une matrice.

6 Les opérations

6.1 Les opérations matricielles

Les opérations matricielles telles que, + l'addition, − la soustraction, * la multiplication, ^ la puissance, \ la division à gauche et / la division à droite, s'obtiennent avec la même syntaxe que pour les opérations sur les scalaires. La transposition d'une matrice A s'écrit $A.'$
NB : A' est la transposée de la conjuguée de la matrice A.
Matlab contient beaucoup de fonctions pour le calcul matriciel telles que `trace, det, poly, eig, null, lu ...` La liste de ces fonctions peut être obtenue en tapant `help matfun`.

Exemple :

```
>>C=A*A.'
C =
    78    23   123
    23    11    35
   123    35   195

>> [P, D]=eig(C)
```

```
P =
    0.0629    0.8469    0.5280
    0.9736   -0.1685    0.1542
   -0.2196   -0.5044    0.8351

D =
    4.5917         0         0
```

```
         0    0.1685         0
         0         0  279.2398

>> P*D*P^(-1)

ans =
   78.0000   23.0000  123.0000
   23.0000   11.0000   35.0000
  123.0000   35.0000  195.0000
```

6.2 Les opérations sur chaque élement d'une matrice

En faisant précéder d'un point les opérateurs $*$, \, / et ^, on réalise des opérations élément par élément.
Exemple :

```
>> B.^2
ans =
     1     4     9     1
     4     9     1     4
     9     1     4     9
```

La plupart des fonctions scalaires de Matlab sont faites pour travailler aussi bien sur un scalaire que sur tous les éléments d'une matrice. Il faut en profiter, cela évite d'écrire des boucles.
Exemple : Si X est une matrice $n \times m$ de réels, alors $Y = \sin(X)$ est une matrice $n \times m$ telle que $Y(i,j) = \sin(X(i,j))$ pour tout i, j.

6.3 Les fonctions vectorielles

Certaines fonctions de Matlab opèrent essentiellement sur les vecteurs (lignes ou colonnes). Lorsqu'on les applique à des matrices, elles opèrent colonne par colonne.
Le tableau suivant décrit le résultat de quelques-unes de ces fonctions lorsqu'elles sont appliquées à un vecteur x :

`sum`(x)	retourne la somme des éléments de x
`cumsum`(x)	retourne un vecteur contenant la somme cumulée des premiers éléments de x : $[x(1) \quad x(1)+x(2) \quad ...]$
`prod`(x)	retourne le produit des éléments de x
`cumprod`(x)	retourne un vecteur contenant le produit cumulé des premiers éléments de x : $[x(1) \quad x(1)x(2) \quad ...]$
`diff`(x)	retourne le vecteur des différences entre deux éléments consécutifs de x : $[x(2)-x(1) \quad x(3)-x(2) \quad ...]$
`max`(x)	retourne la valeur maximale des éléments du vecteur x
`min`(x)	retourne la valeur minimale des éléments du vecteur x
`sort`(x)	retourne un vecteur de même taille que x, dont les éléments sont triés par ordre croissant
$[y,\ I]=$ `sort`(x)	retourne dans y les élements de x ordonnés par ordre croissant et dans I l'image des indices de x par la permutation qui a permis d'ordonner les éléments de x : $x(I) = y$

Exemples d'utilisations sur une matrice :

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

>> sum(A)
ans =
     0    -1     1

>> prod(A)
ans =
    -2    -8   -72

>> cumsum(A)
ans =
     1     2     3
    -1    -2    -3
     0    -1     1

>> cumprod(A)
ans =
     1     2     3
    -2    -8   -18
    -2    -8   -72

>> diff(A)
ans =
    -3    -6    -9
     3     5    10

>> max(A)
ans =
     1     2     4
```

```
>> [As,I]=sort(A)
As =
    -2    -4    -6
     1     1     3
     1     2     4

I =
     2     2     2
     1     3     1
     3     1     3
```

NB : On peut utiliser ces fonctions sur des matrices en rajoutant un paramètre supplémentaire qui vaut 1 si on veut que la fonction opère sur les colonnes et 2 si on veut que la fonction opère sur les lignes.
Exemple :

```
>> X = [0 1 2 ; 3 4 5]
X =
     0     1     2
     3     4     5

>> sum(X,1)
ans =
     3     5     7

>> sum(X,2)
ans =
     3
    12
```

7 Les opérateurs relationnels et logiques

Opérateurs relationnels	$<$,	$<=$,	$>=$,	$==$ (égalité),	$\sim=$ (différent)
Opérateurs logiques	& (et),	| (ou),	xor (ou exclusif),	$\sim$ (non)	

NB : Bien faire la différence entre "==" et "=" qui sert pour affecter une valeur à une variable.
NB : 0 sert à désigner le booléen 'faux' et 1 le booléen 'vrai', mais tout réel non nul ou caractère est identifié par Matlab au booléen 'vrai'.
Exemples :

```
>> 2 & 3
ans =
     1

>> 0 & 1
ans =
     0
```

Ordres de priorité par ordre décroissant d'importance : opérations arithmétiques puis opérations relationnelles puis opérations logiques.

Exemple :

```
>> 2+1>2
ans =
     1
```

Comme beaucoup de fonctions de Matlab, les opérateurs relationnels et logiques s'utilisent avec des matrices. L'opérateur est évalué sur chaque élément de la matrice.
Exemple : $y = (x > 0)\&(x < 5)$ est une matrice de mêmes dimensions que x telle que

$$y(i,j) = \begin{cases} 1 & \text{si } x(i,j) > 0 \text{ et } x(i,j) < 5 \\ 0 & \text{sinon} \end{cases}$$

Exemple :

```
>> x=[0 -1 2 -10 3]
x =
     0    -1     2   -10     3
>> y=(x>1)
y =
     0     0     1     0     1
```

```
>> C=[1 2 ; 1 4];
>> D=[0 1 ; 3 2];
>> C>D
ans =
     1     1
     0     1
```

`any`(x)	si x est un vecteur, retourne 1 lorsqu'au moins un des éléments du vecteur x est non nul et retourne 0 sinon. Si x est une matrice, retourne un vecteur ligne, résultat de la fonction `any` à chaque colonne de la matrice.
`all`(x)	retourne 1 si tous les éléments du vecteur x sont non nuls et retourne 0 sinon. Si x est une matrice, retourne un vecteur ligne, résultat de la fonction `all` à chaque colonne de la matrice.
I = `find`(x)	retourne les indices des éléments non nuls du vecteur x.
[I, J]=`find`(x)	retourne les indices des lignes (dans le vecteur I) et des colonnes (dans le vecteur J) des éléments non nuls de la matrice x.

Exemple :

```
>> all(D)
ans =
     0     1

>> any(any(D))
ans =
     1

>> [I,J]=find(D>1)
I =
     2
     2
J =
     1
     2

>> K=find(D(1,:)==1)
K =
     2
```

8 Les fichiers `.m`

8.1 Les programmes

Un programme est un fichier texte dont le nom a pour extension `.m`. Il est constitué simplement d'une suite d'instructions. On crée un tel fichier à l'aide d'un éditeur de texte celui de Matlab ou un éditeur extérieur comme par exemple `emacs`.
Les instructions d'un programme appelé *nom.m* sont exécutées lorsqu'on tape *nom* dans la fenêtre de commandes.
NB : On peut insérer des commentaires dans un programme en utilisant le caractère % : une ligne commençant par un % n'est pas exécutée.

Exemple : Pour créer un programme appelé `tirage` qui tire 100 nombres "au hasard" entre 0 et 1 et calcule leur moyenne, il suffit d'ouvrir un nouveau fichier dans un éditeur de texte, de taper les trois lignes suivantes :

```
clear
x=rand(1,100);
moyenne=mean(x)
```

puis de donner à ce fichier le nom `tirage.m`. Une exécution de ce programme donne :

```
>> tirage
moyenne =
    0.5159
```

8.2 Les fonctions

Une fonction est un fichier texte dont le nom a pour extension `.m` et qui commence par la ligne

`function` [*res1*,...,*resn*]=*nomdelafonction*(*var1*,...,*vars*)

suivie de lignes de commentaires décrivant la fonction puis d'une suite d'instructions définissant les valeurs des variables *res1*,...,*resn*.

- *nomdelafonction* doit être le même que le nom du fichier (sans l'extension `.m`)
- *var1*,...,*vars* sont les arguments d'appel de la fonction
- *res1*,...,*resn* sont les arguments de sortie

La majorité des commandes définies dans Matlab sont des fonctions. Un programme est souvent une succession d'appels à des fonctions, chacune des fonctions étant chargée de faire un calcul particulier effectué à partir des valeurs des paramètres d'entrée. Les fonctions que l'on crée se rajoutent aux fonctions définies par Matlab et peuvent être appelées dans n'importe quel programme ou nouvelle fonction.
Les variables utilisées dans le corps de la fonction autres que les arguments de sortie sont des variables internes. Les modifications apportées à un argument d'entrée dans le corps de la fonction ne sont pas visibles à l'extérieur

de la fonction, sauf si l'argument d'entrée est aussi un argument de sortie de la fonction.

Exemple : En tapant `type flipud`, on obtient le listing du fichier `flipud.m`. Il contient les lignes suivantes :

```
function y = flipud(x)
%FLIPUD Flip matrix in up/down direction.
%   FLIPUD(X) returns X with columns preserved and rows flipped
%   in the up/down direction.  For example,
%
%   X = 1 4      becomes  3 6
%       2 5               2 5
%       3 6               1 4
%
%   Class support for input X:
%      float: double, single
%
%   See also FLIPLR, ROT90, FLIPDIM.

%   Copyright 1984-2004 The MathWorks, Inc.
%   $Revision: 5.9.4.3 $  $Date: 2004/07/05 17:01:15 $

if ndims(x)~=2
  error('MATLAB:flipud:SizeX', 'X must be a 2-D matrix.');
end
y = x(end:-1:1,:);
```

Cette fonction demande un paramètre en entrée (une matrice) et retourne un seul résultat (qui est une matrice de même taille que celle entrée).
Les premières lignes de commentaires dans un fichier `nomdelafonction.m` jusqu'à la première ligne qui ne commence pas par % apparaissent dans l'aide en ligne en tapant `help nomdelafonction`.

Exemple :

```
>> help flipud
FLIPUD Flip matrix in up/down direction.
    FLIPUD(X) returns X with columns preserved and rows flipped
    in the up/down direction.  For example,

    X = 1 4      becomes  3 6
        2 5               2 5
        3 6               1 4

    Class support for input X:
       float: double, single

    See also fliplr, rot90, flipdim.

    Reference page in Help browser
       doc flipud
```

Il est commode de ranger les variables de sortie *res1*,...,*resn* d'une fonction par ordre décroissant d'interêt car pour avoir accès à la valeur de la variable *res2* par exemple, il faut appeler la fonction avec au moins deux variables de sortie ; si on tape la commande

$$[a1, a2] = nomdelafonction(b1, \ldots, bs)$$

on aura dans $a1$ (resp. $a2$) la valeur de $res1$ (resp. $res2$) calculée à partir des paramètres d'entrées $b1, \ldots, bs$.

Exemple : La fonction `sort` qui permet d'ordonner les élements de chaque colonne d'une matrice par ordre croissant a un paramètre d'entrée et deux paramètres de sortie (voir page 6) :

```
>> x=[2 1 3 0];

>> xord=sort(x)
xord =
    0     1     2     3

>> [xord, perm]=sort(x)
xord =
    0     1     2     3

perm =
    4     2     1     3

>> x(perm)
ans =
    0     1     2     3
```

8.3 Quelques outils

`disp`(*var*)	affiche le contenu de *var* dans la fenêtre de commande
rep=`input`(*'texte'*)	affiche la chaîne de caractères *texte* et donne la main à l'utilisateur pour qu'il entre la valeur de la variable *rep*.
`pause`	arrête l'exécution d'un programme jusqu'à ce que l'utilisateur appuie sur une touche du clavier
`pause`(n)	arrête l'exécution d'un programme pendant n secondes
`waitforbuttonpress`	arrête l'exécution d'un programme jusqu'à ce que l'utilisateur appuie sur une touche du clavier ou sur un bouton de la souris
`keyboard`	arrête l'exécution d'un programme et donne la main à l'utilisateur jusqu'à ce que celui-ci tape `return` en toute lettre, l'exécution du programme reprend alors.
`type` *nom-fich*	affiche, dans la fenêtre de commandes, le contenu du fichier *nom-fich*
`help` *nom-fich*	affiche, dans la fenêtre de commandes, l'aide en ligne de la fonction *nom-fich* (c'est-à-dire les lignes de commentaires du fichier *nom-fich* qui sont directement après la ligne commençant par `function`).

Pour afficher un commentaire suivi d'un résultat numérique, on peut le convertir en une chaîne de caractères. Cela est surtout utile lorsqu'on veut afficher le résultat d'un calcul dans une fenêtre graphique.

Conversion	
`num2str`(x, n)	convertit un nombre x en une chaîne de caractères, n précisant le nombre maximal de chiffres affichés (n est facultatif)
`mat2str`(x, n)	convertit une matrice x en une chaîne de caractères, n précisant le nombre maximal de chiffres affichés pour chaque élément de la matrice (n est facultatif)

Exemple : Le fichier `etude.m` contient les lignes suivantes :

```
% Etude statistique d'une serie de nombres
val=input('entrer un vecteur contenant des reels ');
moy=mean(val);
ecart=std(val);
disp(['etude statistique des nombres ' mat2str(val) ' : '])
disp(['la moyenne est : ' num2str(moy)])
disp(['l''ecart-type est : ' num2str(ecart)])
```

Voici un exemple d'exécution du programme `etude` :

```
>> etude
entrer un vecteur contenant des reels [1 5  3 4 9]
etude statistique des nombres [1 5 3 4 9] :
la moyenne est : 4.4
l'ecart-type est : 2.9665
```

On dispose de plusieurs fonctions pour contrôler le "coût" d'exécution d'un programme :

`tic`	lance un chronomètre
`toc`	affiche le temps écoulé (en secondes) depuis le dernier `tic`
`cputime`	retourne le nombre de secondes qui se sont écoulées depuis le début de la session MATLAB

Exemple :

```
>> tic;prod(1:100);toc
Elapsed time is 0.000257 seconds.
```

9 Représentation graphique des résultats

9.1 Représentations de points dans le plan

Il existe plusieurs possibilités pour représenter un ensemble de points $(x(i), y(i))$. Les plus utilisées sont énumérées dans le tableau :

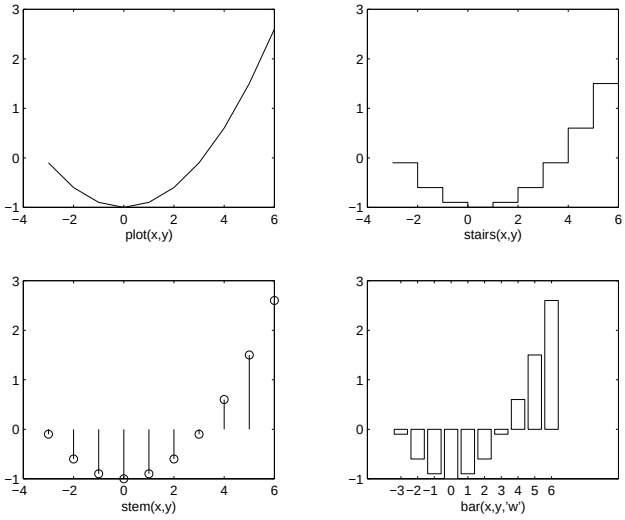
`plot(x,y,'s')`	tracé d'une courbe ou d'un nuage de points
`bar(x,y,'s')`	tracé sous forme d'un diagramme en barres
`stem(x,y,'s')`	tracé sous forme d'un diagramme en bâtons
`stairs(x,y,'s')`	les valeurs discrètes sont reliées par des marches d'escalier

's' est un paramètre facultatif constitué d'une chaîne de caractères qui spécifie le type de tracé (couleur, différents tracés en pointillés, symbole pour le tracé de points). Par défaut, le tracé est continu. On obtient la liste des possibilités en tapant `help plot` :

couleur				type de tracé		marqueur	
k	black	y	yellow	-	solid	.	point
w	white	m	magenta	:	dotted	x	x-mark
b	blue	c	cyan	-.	dashdot	o	circle
r	red	g	green	–	dashed	+	plus
				*	star		

Exemple : Les figures suivantes donnent différentes représentations des points suivants :

x	-3	-2	-1	0	1	2	3	4	5	6
y	-0.1	-0.6	-0.9	-1.0	-0.9	-0.6	-0.1	0.6	1.5	2.6



9.2 Gestion de la fenêtre graphique

`hold on`	les prochains tracés se superposeront aux tracés déjà effectués
`hold off`	le contenu de la fenêtre graphique sera effacé lors du prochain tracé
`clf`	efface le contenu de la fenêtre graphique
`figure`	ouvre une nouvelle fenêtre graphique
`close`	ferme la fenêtre graphique sur laquelle on travaille
`close all`	ferme toutes les fenêtres graphiques
`subplot`(n,m,p)	partage la fenêtre graphique en $n \times m$ espaces graphiques (n dans la hauteur et m dans la largeur) et sélectionne le p-ième en partant de celui en haut, à gauche.
`drawnow`	force Matlab à mettre à jour le contenu de la fenêtre graphique.

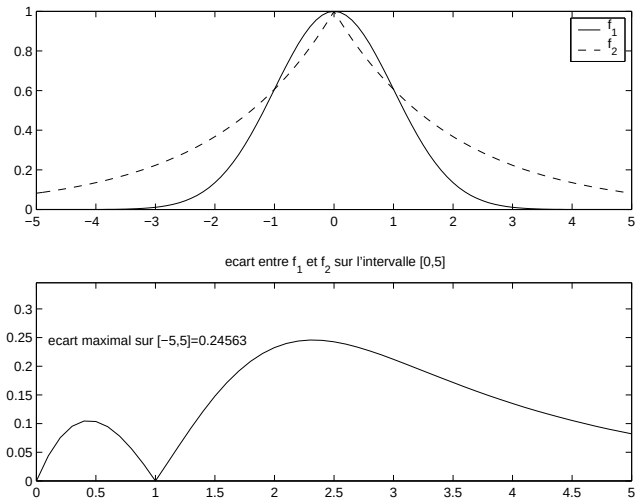
9.3 Axes et légendes

`axis([xmin xmax ymin ymax])`	pour définir les échelles des x et des y
`grid`	quadrillage du graphique
`grid off`	
`title`('titre')	titre pour le graphique
`xlabel`('titre')	légende pour l'axe des x
`ylabel`('titre')	légende pour l'axe des y
`legend`('titre1','titre2',...)	légende pour chaque courbe du graphique à écrire dans l'ordre de leurs tracés
`text`$(x,y,$'texte')	texte explicatif écrit à partir de la position (x,y)
`gtext`('texte')	texte positionné à l'endroit voulu dans la fenêtre graphique en cliquant avec le bouton de droite de la souris

Exemple : Le programme suivant représente les courbes des fonctions $f_1 : x \mapsto \exp(-0.5x^2)$ et $f_2 : x \mapsto \exp(-0.5|x|)$ sur l'intervalle $[-5, 5]$ sur un premier graphique, puis un zoom de l'écart entre les deux fonctions sur l'intervalle [0,5].

```
clear;clf;
x=-5:0.1:5;
y1=exp(-0.5*x.^2);
y2=exp(-0.5*abs(x));
subplot(2,1,1)
plot(x,y1)
hold on
plot(x,y2,'--')
hold off
legend('f_{1}','f_{2}')
subplot(2,1,2)
h=abs(y1-y2);m=max(h);
plot(x,h)
axis([0 5 0 m+0.1])
text(0.1,m,['ecart maximal sur [-5,5]=' num2str(m)])
title('ecart entre f_{1} et f_{2} sur l''intervalle [0,5]')
```

La figure suivante montre le résultat de l'exécution de ce programme.



9.4 Quelques fonctions pour la représentation graphique en trois dimensions

`plot3`	commande analogue à `plot` pour le tracé de points dans l'espace.
`mesh`	représentation d'une surface en "fils de fer"
`surf`	représentation d'une surface
`contour`	représentation des lignes de niveau d'une surface en projection
`contour3`	représentation des lignes de niveau d'une surface en 3D
`pcolor`	représentation 2D d'une surface, la 3ème dimension étant visualisée par des modifications de couleurs.
`meshc`	superposition des graphiques issus de `mesh` et de `contour`
`surfc`	superposition des graphiques issus de `surfc` et de `contour`
`meshgrid`	création d'une grille de points
`colormap`	définition d'une table de couleurs
`colorbar`	affichage de la table de couleurs
`view`	définition de l'angle sous lequel le graphique sera regardé.

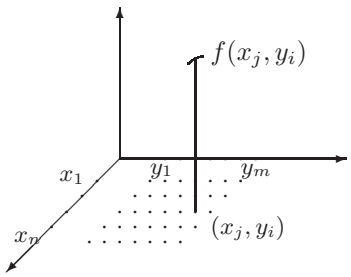
NB : Utiliser `demo` pour voir toutes les possibilités de Matlab concernant les représentations graphiques.

Exemple : Pour représenter une fonction $(x, y) \mapsto f(x, y)$ sur un rectangle $I \times J$, on commence par définir des vecteurs $x = [x_1, \ldots, x_n]$ et $y = [y_1, \ldots, y_m]$ contenant les points d'une discrétisation des intervalles I et J.

La fonction `[Gx,Gy]=meshgrid(x,y)` définit deux matrices de taille $m \times n$

$$Gx = \begin{pmatrix} x_1 & x_2 \dots & x_n \\ \vdots & \vdots & \vdots \\ x_1 & x_2 \dots & x_n \end{pmatrix} \text{ et } Gy = \begin{pmatrix} y_1 & y_1 \dots & y_1 \\ \vdots & \vdots & \vdots \\ y_m & y_m \dots & y_m \end{pmatrix}.$$

Gx et Gy contiennent respectivement les abscisses et les ordonnées des points de la grille de $I \times J$ définie par les vecteurs x et y.

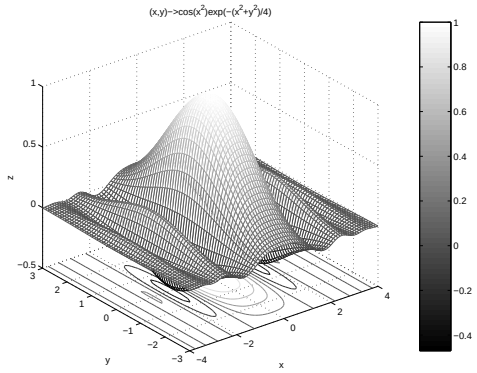


On calcule ensuite la fonction f en chaque point de cette grille :

$$Gz(i,j) = f(Gx(i,j), Gy(i,j)) = f(x_j, y_i) \text{ pour tout } (i,j) \in \{1,\dots,n\} \times \{1,\dots,m\}.$$

On peut alors utiliser l'une des fonctions de représentation d'une surface avec comme variables d'entrée dans l'ordre : Gx, Gy et Gz. Par exemple, le programme suivant représente la fonction $(x,y) \mapsto \cos(x^2)e^{-(x^2+y^2)/4}$ sur le rectangle $[-4,4] \times [-3,3]$:

```
x=[-4:0.1:4];
y=[-3:0.1:3];
[Gx,Gy]=meshgrid(x,y);
Gz=cos(Gx.^2).*exp(-(Gx.^2+Gy.^2)/4);
meshc(Gx,Gy,Gz)
colormap('gray')
xlabel('x');ylabel('y');zlabel('z');
title('(x,y)->cos(x^2)exp(-(x^2+y^2)/4)')
view([-38,30]); %(pour l'altitude et l'azimuth)
colorbar
```



9.5 La sauvegarde d'une figure

Une figure peut être sauvegardée sous plusieurs formats :

- sous un format propre à matlab avec l'extension `.fig`. Pour cela, cliquer sur le bouton `save as` du menu `file` de la fenêtre graphique et entrer un nom de fichier avec l'extension `.fig` dans l'encadré qui apparaît. Un tel fichier peut être visualisé en utilisant le bouton `open` du menu `menu` de la fenêtre graphique.
- sous un format postscript
  - soit en utilisant la commande `print` -deps *nomfichier* : un fichier *nomfichier.eps* est crée dans le répertoire courant.
  - soit en utilisant le bouton `export` du menu `file`; entrer alors un nom pour le fichier suivi de l'extension `.eps`.

  On peut visualiser un fichier postscript en utilisant le logiciel `evince` (ou en le sélectionnant à partir du gestionnaire de fichier) et on peut imprimer la figure en utilisant la commande linux d'impression `lpr` *nomfichier.eps*.

  Si on veut imprimer directement une figure sans faire de sauvegarde, il suffit de cliquer sur le bouton représentant une imprimante dans le menu de la fenêtre graphique.

10 Les commandes structurées

10.1 L'instruction `for`

`for` *variable = vecteur*
instructions
`end`

Les colonnes du *'vecteur'* sont affectées, l'une après l'autre à la variable '*variable*' et pour chacune de ces valeurs, les *'instructions'* sont exécutées.

NB : Il faut privilégier les opérations vectorielles à l'utilisation de boucles.

Exemple : Ces quelques lignes calculent $n!$ pour $n = 1$ à 100.

```
n=100; fact=1;
for k=2:n
  fact=fact*k;
end
```

Mais, on peut aussi taper simplement :

```
n=100;
fact=prod(1:n);
```

L'exécution du premier programme prend plus de temps que l'exécution du deuxième programme.

10.2 L'instruction `if`

`if` *conditions*
    instructions
`end`
Les '*instructions*' ne sont exécutées que si les '*conditions*' sont vérifiées, plus précisément si '*conditions*' a une valeur différente de 0.

Une variante possible est :
`if` *conditions*
    instructions (exécutées si les '*conditions*' sont vérifiées)
`else`
    instructions (exécutées si les '*conditions*' ne sont pas vérifiées)
`end`

Plusieurs instructions `if` peuvent être emboitées, par exemple :
`if` *conditions1*
    instructions (exécutées si les '*conditions1*' sont vérifiées)
`elseif` *conditions2*
    instructions (exécutées si les '*conditions1*' ne sont pas vérifiées et si '*conditions2*' sont vérifiées)
`else`
    instructions (exécutées si les '*conditions1*' et les '*conditions2*' ne sont pas vérifiées)
`end`
Exemple : Le programme suivant simule le lancer d'une pièce.

```
p=0.5;
u=rand;
if u<p
   disp('pile')
else
   disp('face')
end
```

10.3 L'instruction `while`

`while` *conditions*
    instructions
`end`
Les '*instructions*' sont exécutées tant que les '*conditions*' sont vérifiées.

Exemple : Le programme suivant simule un jeu de pile ou face où on mise 1 euro à chaque lancer. Il représente l'évolution de la fortune du joueur au cours du temps. Le jeu s'arrête si le joueur ne peut plus miser.

```
clear
s=input('fortune initiale :');
n=input('nb de lancers a effectuer :');
if (s<1 | round(n)~=n) | (n<=0)
  error('donnees incompatibles avec le jeu');
end
i=0;
som=s;
while (s>=1 & i<=n)
  s=s+2*(rand<0.5)-1;
  i=i+1;
  som=[som s];
end
stairs(0:i,som)
axis([0 n+1 -1 s+n])
xlabel('nb de lancers')
title('evolution de la fortune au cours du jeu')
if i<n
  text(i,0,'poursuite du jeu impossible')
end
```

Instructions de rupture de séquence	
`break`	termine l'exécution de la boucle la plus proche dans laquelle se trouve `break`
`return`	permet de terminer l'exécution d'un fichier `.m` sans aller jusqu'à la fin du fichier
`error`('*message*')	affiche la chaîne de caractères *message* et interrompt l'exécution du programme en cours

11 Lecture et écriture de fichiers de données

En plus des commandes `save` et `load` présentées page 2, il existe des commandes pour lire et écrire des fichiers textes contenant des données. La liste complète de ces commandes s'obtient en tapant `help iofun`.

`fopen`	ouverture d'un fichier
`fclose`	fermeture d'un fichier
`fsprintf`	écriture des données dans un fichier texte ou affichage à l'ecran de données
`fscanf`	lecture de données d'un fichier texte

11.1 Ouverture et fermeture d'un fichier

L'instruction *fid*=`fopen`(*nomfichier*,*permission*) ouvre le fichier *nomfichier* et retourne un entier qui identifie ce fichier dans *fid*, *permission* étant une chaîne de caractères spécifiant les opérations possibles avec ce fichier :
- 'r' pour lire (par défaut, le fichier est ouvert en mode lecture),
- 'w' pour écrire,
- 'a' pour ajouter,
- 'a+' pour lire et ajouter,
- 'w+' pour écrire et lire.

NB : $fid = -1$ si le fichier n'a pas pu être ouvert.

L'instruction `fclose(`*fid*`)` ferme le fichier dont l'identificateur donné par la commande `fopen` est *fid*.
L'instruction `fclose('all')` ferme tous les fichiers ouverts.

11.2 Ecriture de données dans un fichier texte

compt=`fprintf`(*fid, format , var1, var2 ,...*) convertit le contenu des variables *var1, var2, ...* en une chaîne de caractères et les écrit dans le fichier dont l'identificateur est *fid* ; *format* est une chaîne de caractères comprenant
- du texte à écrire tel quel,
- des spécifications sur les formats de conversion des variables *var1, var2,...* commençant par le caractère %.

Cette commande retourne le nombre de caractères écrits dans *compt*.

NB : Lorsque le paramètre fid est omis dans la commande `fprintf`, le résultat de la conversion est affichée à l'écran.
La spécification d'une variable est de la forme : $\% \pm n.mx$ où
- n est le nombre minimal de caractères qu'occupera chaque élément de la variable une fois convertie,
- m donne le nombre de chiffres significatifs ou le nombre de chiffres après la virgule suivant le type de conversion choisi,
- $-$, signifie que l'écriture se fera sans espace à droite,
- $+$, signifie que les nombres seront tous précédés d'un signe + ou - suivant leur signe,
- x est un caractère précisant le type de conversion. Les plus courant sont :
  - d, e, f et g pour la conversion en un nombre décimal suivant le type d'affichage désiré,
  - c pour l'écriture d'un caractère,
  - s pour l'écriture des m premiers caractères d'une chaîne de caractères, si m est spécifie ou de tous les caractères avant le premier espace, si m n'est pas spécifié.

Exemple : différents affichages du vecteur $v = [-2,\ 10,\ 3]$, des réels π et $x = \sqrt{2}$.

```
>> fprintf('%3d',v)
 -2 10  3>>

>> fprintf('%-3d',v)
-2  10 3  >>

>> fprintf('%+3d',v)
 -2+10 +3>>
```

```
>> fprintf('%.1e\n',v)
-2.0e+00
1.0e+01
3.0e+00

>> fprintf('%7.2g et %7.3f\n',x,pi)
```

```
    1.4 et   3.142                                  >> fprintf('%7.2f et %7.3e\n',x,pi)
                                                       1.41 et 3.142e+00
```

NB : Le caractère >> après le chiffre 3 aux lignes 2 et 4, est le prompt de Matlab, qui apparaît à la suite de l'affichage du contenu de la variable v, car aucun passage à la ligne n'a été effectué. `\n` est un caractère spécial qui signifie "passer à la ligne suivante".

11.3 Lecture de données d'un fichier texte

$[A, compt]$=`fscanf`(*fid,format,n*) convertit les n premiers éléments du fichier dont l'indicateur est *fid* suivant le format indiqué dans *format*. Le résultat est une vecteur à n colonnes A (par défaut, tout le fichier est lu c'est-à-dire $n = inf$) ; *compt* contient le nombre de données lues.

$[A, compt]$=`fscanf`(*fid, format*, $[m, n]$) convertit suffisamment d'éléments du fichier dont l'indicateur est *fid* pour remplir une matrice de taille $m \times n$. A est donc une matrice de taille au plus $m \times n$ (n peut être égal à inf mais pas m).
NB : *format* est ici aussi une chaîne de caractères contenant les spécifications des données à lire (commençant par %) et éventuellement des caractères qui sont présents dans le fichier mais qui ne doivent pas être enregistrés dans A. Les spécifications les plus courantes sont : `%f`, `%g`, `%e` pour des nombres à virgule flottante, `%d` pour des entiers, `%c` et `%s` pour des chaînes de caractères.

Exemple :

```
>> u=1:4;                                       >> fprintf(id,'%4.2e %4.2e\n',A.');
>> A=[3.4 9.1 ; 0.21 5.6 ; 2.1 -3];             >> fprintf(id,'%3g',u);
>> id=fopen('donnees.dat','w');                 >> fclose(id);
```

Ces lignes de commandes créent un fichier donnees.dat contenant les quatre lignes suivantes :

```
3.40e+00 9.10e+00
2.10e-01 5.60e+00
2.10e+00 -3.00e+00
  1  2  3  4
```

NB : Lorsque la variable à convertir est une matrice, `fprintf` traite les éléments colonne par colonne. Ainsi dans la commande `fprintf`(id,'%4.2e %4.2e\n',$A.'$), %4.2e spécifie le format pour les premiers élements de chaque colonne de $A.'$ et %4.2e pour les seconds élements de chaque colonne de $A.'$.

Exemple : les lignes de commandes suivantes permettent de lire les données du fichier donnees.mat précédent :

```
>> fid=fopen('donnees.dat','r');                w =
>> B=fscanf(fid,'%g',[2,3])                         1
B =                                                 2
    3.4000    0.2100    2.1000                      3
    9.1000    5.6000   -3.0000                      4

>> w=fscanf(fid,'%hd')                          >> fclose(fid);
```

Exemple : lecture du fichier de données points.dat contenant les lignes suivantes

```
x= 0.8913, y= 0.8214
x= 0.7621, y= 0.4447
x= 0.4565, y= 0.6154
x= 0.0185, y= 0.7919

>> id=fopen('points.dat','r');
>> A=fscanf(id,'x=%f, y=%f\n',[2,4])
A =
    0.8913    0.7621    0.4565    0.0185
    0.8214    0.4447    0.6154    0.7919
>> fclose(id);
```

Table des matières