Prise en main du logiciel MATLAB

Ce tutoriel contient les bases permettant l'utilisation du logiciel Matlab dans divers domaines tels que les graphiques, la résolution de systèmes, les statistiques, le temps de calcul, la programmation, le traitement d'images.

Réalisé par Arnaud Brouwers
Année académique 2009-2010

ArGEnCo - MS2F
Hydrologie, Hydrodynamique Appliquée
Constructions Hydrauliques (HACH)

Université de Liège

Table des matières

Chapitre 1

Pour débuter

1.1 Définitions de scalaires

La définition d'éléments est assez intuitive. Il est cependant toujours recommandé d'utiliser des conventions pour la nomenclature des variables. Par exemple, que tous les noms des scalaires commencent par une minuscule et tous les noms des matrices par une majuscule, etc. De plus, il est vivement conseillé de choisir des noms de variables explicites sans pour autant les rendre trop long (exemple : MatriceDeDilatation, nbr_itaration, ...)

Remarques :
- Matlab est sensible à la case des variables.
- Un nom de variable doit commencer par une lettre. Il peut ensuite contenir des chiffres et d'autres symboles.

```
% — Définitions de scalaires ———————————————————————— %

a = 1;
A = 2;
A = A * 2;

disp(a);        % a = 1
disp(A);        % A = 4

z = 3 + 4*i;    % Nombre complexe
```

1.2 Définitions de vecteurs et de matrices

Pour définir un vecteur sous Matlab, il suffit de mettre ses valeurs entre crochets et de les séparer par un espace. Pour obtenir un vecteur vertical, il suffit de le transposer via l'ajout d'un apostrophe après le crochet fermant.

```
% — Vecteurs ————————————————————————————————- %

vecH = [ 1 2 3 4 5] ;          % Vecteur horizontal
vecV = [ 1 2 3 4 5]' ;         % Vecteur vertical

% — Matrices ———————————————————————————————— %

mat = [ 1 2 3 ;                % Ligne 1
4 5 6 ] ;                      % Ligne 2

% — Matrices Complexes ————————————————————————— %

Z = [1 2 ; 3 4] + i * [5 6 ; 7 8]
```

1.3 Création de matrices particulières

Matlab permet de créer de manière simplifiée des matrices spéciales comme, par exemple, des matrices remplies de 1 ou de 0, etc. (*m et n sont les dimensions de la matrice souhaitée et A est une matrice*)

```
% — Définitions de matrices spéciales ———————————————————— %

eye(m,n)            % Matrice unité
ones(m,n)           % Matrice dont tous les éléments sont égaux à un
zeros(m,n)          % Matrice dont tous les éléments sont égaux à zéro
rand(m,n)           % Matrice d'éléments aléatoires
diag(m,n)           % Matrice diagonale
meshgrid(m :n,k :l) % Renvoie deux matrices de grilles définissant un quadrillage
```

1.4 Opérateurs de bases

Les opérateurs ci-dessous sont valables pour les scalaires comme pour les matrices. Matlab effectuera la bonne opération en fonction du type des termes de l'opérateur.

```
% — Définitions de matrices spéciales ———————————————————— %

'    % Transposition (matrice)
+    % Addition
-    % Soustraction
*    % Multiplication
/    % Division à droite
\    % Division à gauche
^    % Puissance
```

1.5 Fonctions sur les scalaires

Voici une liste non-exhaustive des fonctions de bases sur les nombres scalaire.(*dans l'exemple : z représente un nombre complexe*).

```
% — Fonctions sur les scalaires ————————————————————————- %

abs(z)      % Valeur absolue ou module complexe
conj(z)     % Complexe conjugué
imag(z)     % Partie imaginaire
real(z)     % Partie réelle
norm(z)     % Norme
prod(z)     % Produit de tous les éléments d'un argument vectoriel
sum(z)      % Somme de tous les éléments d'un argument vectoriel
round(z)    % Arrondit tous les éléments aux entiers les plus proches
```

1.6 Fonctions sur les matrices

Voici une liste non-exhaustive des fonctions de bases sur les vecteurs et matrices. (*Dans l'exemple : V et A représentent des vecteurs ou des matrices*).

```
% —Fonctions sur les matrices ————————————————————————- %

cumsum(V)     % Somme cumulée
cumprod(V)    % Produit cumulé
sum(V)        % Soustraction
prod(V)       % Multiplication
tril(A)       % Extraction de la partie triangulaire inférieure d'une matrice
triu(A)       % Extraction de la partie triangulaire supérieure d'une matrice
sort(A)       % Trie la matrice
fliplr(A)     % Retourne la matrice horizontalement
flipud(A)     % Retourne la matrice verticalement
rank(A)       % Rang de la matrice
trace(A)      % Somme des éléments diagonaux
```

1.7 Opérations élément par élément

Ajouter un point devant un des opérateurs classiques signifie que l'on effectue l'opération élément par élément. (*Exemple A,B,C étant des matrice, C = A .+ B ≡ C(i,j) = A(i,j) + B(i,j)*).

```
% — Opérations élément par élément ———————————————————————— %

.+    % Addition
.-    % Soustraction
.*    % Multiplication
./    % Division
.^    % Puissance
```

1.8 Constantes prédéfinies

Matlab dispose de constantes prédéfinies. En voici la liste :

```
eps       % Floating-point relative accuracy
i         % Imaginary unit
Inf       % Infinity
intmax    % Largest value of specified integer type
intmin    % Smallest value of specified integer type
j         % Imaginary unit
NaN       % Not-a-Number
pi        % Ratio of circle's circumference to its diameter
realmax   % Largest positive floating-point number
realmin   % Smallest positive normalized floating-point number
```

1.9 Le typage de données

Matlab effectue ce que l'on appel du typage dynamique, autrement dit il adapte le type des opérateurs de manière à permettre aux opérations de s'effectuer. Cette pratique à des avantages comme des inconvénients d'un coté cela évite au programmeur de devoir définir à l'avance le type de chaque variable et d'être coincé tout au long de l'exécution par ce type. D'un autre coté, lorsque le typage est dynamique, Matlab effectue des conversions de type implicites sans le signaler, ce qui peut augmenter le temps d'exécution et parfois mener à des résultats surprenant.

Il est cependant possible de forcer Matlab à faire des conversions explicites.

Fonctions de conversions :

```
% — Fonctions de conversion ———————————————————————— %
cast                          % Cast variable to different data type
double                        % Convert to double precision
int8, int16, int32, int64     % Convert to signed integer
single                        % Convert to single precision
typecast                      % Convert data types without changing underlying data
uint8, uint16, uint32, uint64 % Convert to unsigned integer
```

```
% — Fonctions de conversion string vers nombre ————————————————— %
base2dec        %Convert base N number string to decimal number
bin2dec         %Convert binary number string to decimal number
cast            %Cast variable to different data type
hex2dec         %Convert hexadecimal number string to decimal number
hex2num         %Convert hexadecimal number string to double-precision number
str2double      %Convert string to double-precision value
str2num         %Convert string to number
unicode2native  %Convert Unicode characters to numeric bytes
```

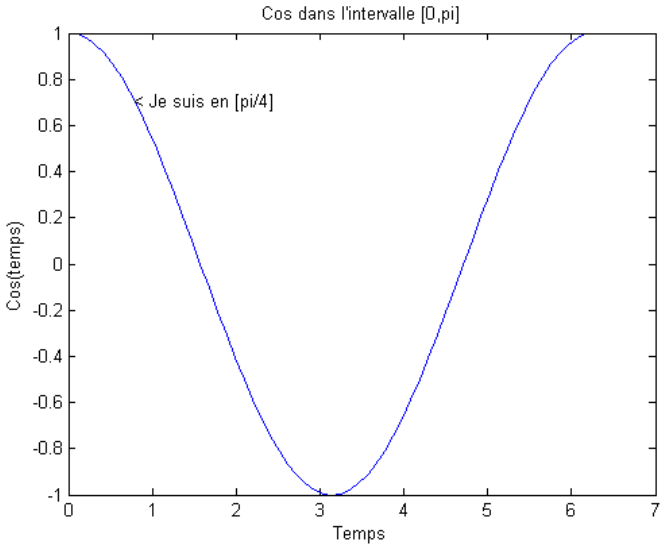
Chapitre 2

Graphiques

2.1 Graphique simple

Les quelques commandes suivantes montrent comment afficher un graphique simple. Matlab ne sachant travailler qu'en valeurs discrètes, il faut d'abord commencer par définir l'intervalle de valeurs de l'abscisse. Une fois ceci fait, on calcul la valeur de la fonction (dans notre cas un cosinus) pour chaque échantillon de temps et l'on arrive au vecteur y.

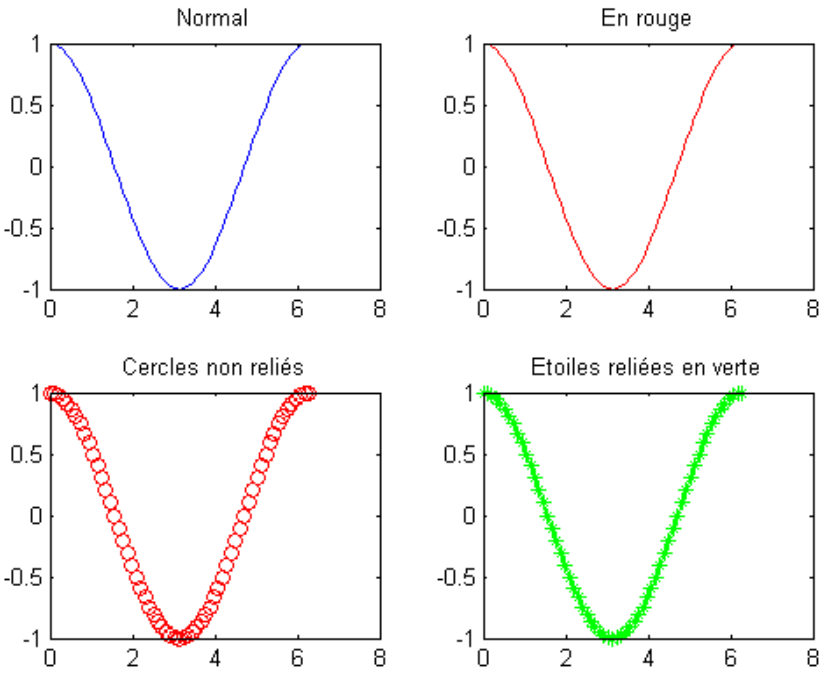
```
% — Graphique simple ———————————————————————————— %

t = 0 :pi/30 :2*pi ;                    % Abscisse de 0 à 2π par pas de π/30
y = cos(t) ;                            % Valeurs de la fonction dans le temps

figure ;                                % Nouvelle fenêtre
plot(t,y)                               % Tracer
title('Cos dans l"intervalle [0,pi]') ; % Titre
xlabel('Temps') ;                       % Légende en abscisse
ylabel('Cos(temps)') ;                  % Légende en ordonnée
text(pi/4,cos(pi/4),'¡ Je suis en [pi/4]')  % Chaîne de caractères à la position x,y
```



2.2 Graphique avancé

Matlab permet de personnaliser les graphiques. Par exemple, il est possible de changer la couleur du trait, de marqué les points avec différents symbole tels que x,o,*, ... etc. Voici un aperçu simple de ce qui est possible de faire.

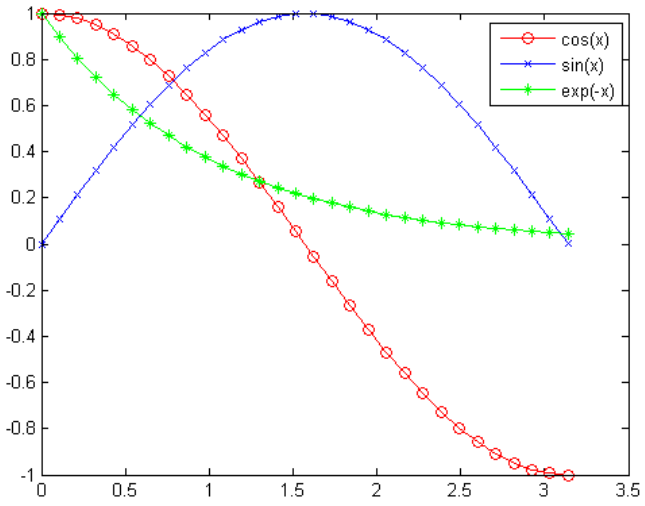
```
% — Graphique avancé ———————————————————— %

figure ;                              % Nouvelle fenêtre

t = 0 :pi/30 :2*pi ;                  % Définition de temps
y = cos(t) ;                          % Fonction du temps

subplot(2,2,1)                        % Zone supérieure gauche
plot(t,y) ;                           % Tracer
title('Normal') ;                     % Titre

subplot(2,2,2)                        % Zone supérieure droit
plot(t,y,'r') ;                       % Tracé en rouge
title('En rouge') ;                   % Titre

subplot(2,2,3)                        % Zone inférieur gauche
plot(t,y,'or') ;                      % Tracé points ronds en rouge
title('Cercles non reliés') ;         % Titre

subplot(2,2,4)                        % Zone inférieur droite
plot(t,y,'*-g') ;                     % Tracé point étoiles relié en vert
title('Etoiles reliées en verte') ;   % Titre
```



2.3 Graphiques multiples 1

Il est possible de tracer plusieurs courbes sur le même graphique. Pour ce faire, une méthode consiste à mettre l'ensemble des fonctions à tracer dans les parenthèses qui suivent la commande `plot`.

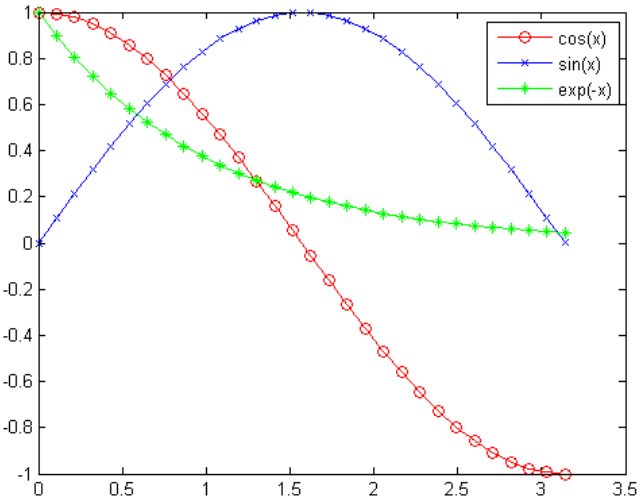
```
% — Graphique multiple 1 ———————————————————————————— %

figure ;                                          % Nouvelle fenêtre

x = linspace(0,pi,30) ;                           % Fonction du temps
plot(x,cos(x),'o-r',x,sin(x),'x-b',x,exp(-x),'*-g')  % Tracer
legend('cos(x)','sin(x)','exp(-x)')               % Légende sur graphique
```



2.4 Graphiques multiples 2

Une autre méthode pour tracer plusieurs courbes sur un même graphique consiste à activer la commande `Hold On` ce qui stipule à Matlab de dessiner les graphiques les uns sur les autres.

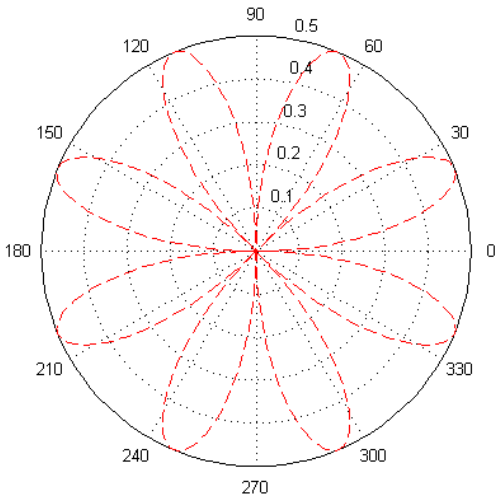
```
% — Graphique multiple 2 ———————————————————————————— %

figure ;                                          % Nouvelle fenêtre

x = linspace(0,pi,30) ;                           % Fonction du temps

hold on                                           % Maintient du graphique
plot(x,cos(x),'o-r')                              % Tracer points ronds reliés en rouge
plot(x,sin(x),'x-b')                              % Tracer points croix reliés en vert
plot(x,exp(-x),'*-g')                             % Tracer points étoiles reliés en bleu
legend('cos(x)','sin(x)','exp(-x)')               % Légende sur graphique
```



2.5 Graphiques polaires

Il est aussi possible de dessiner des graphiques polaires sous Matlab.

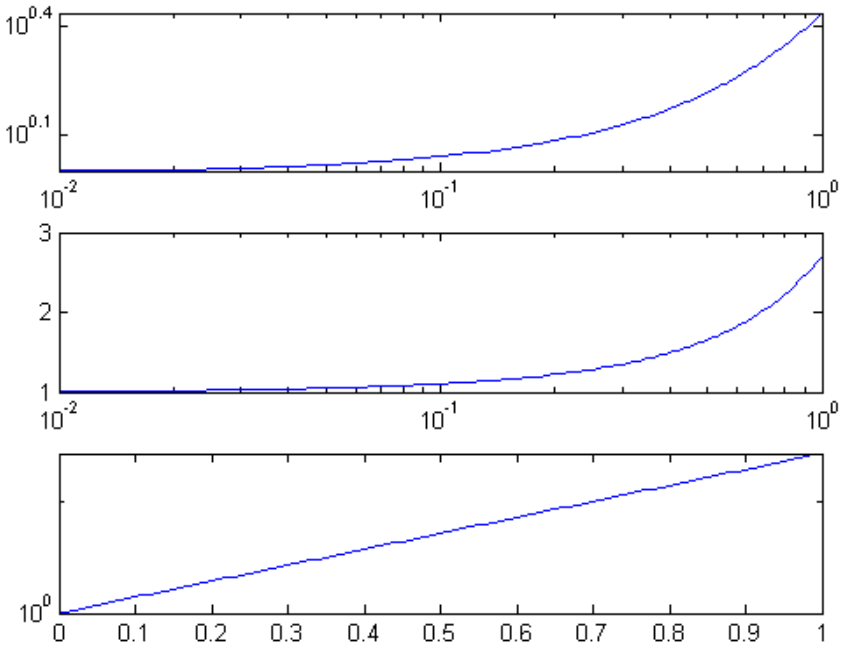
```
% — Graphique polaire ———————————————————————— %

figure ;                          % Nouvelle fenêtre

t = 0 :.01 :2*pi ;                % Échelle du temps
polar(t,sin(2*t).*cos(2*t),'–r')  % Graphique polaire en pointillé rouge
```



2.6 Graphiques à échelles logarithmiques

Il est aussi possible de dessiner des graphiques avec un ou plusieurs axes sous forme logarithmique.

```
% — Graphique à échelles logarithmiques ———————————————————— %

figure ;                % Nouvelle fenêtre

x = 0 :0.01 :1 ;
subplot(3,1,1) ;        % Zone supérieure
loglog(x,exp(x)) ;      % Echelle logarithmique deux axes

subplot(3,1,2)          % Zone centrale
semilogx(x,exp(x))      % Echelle logarithmique sur l'axe Ox

subplot(3,1,3)          % Zone inférieure
semilogy(x,exp(x))      % Echelle logarithmique sur l'axe Oy
```



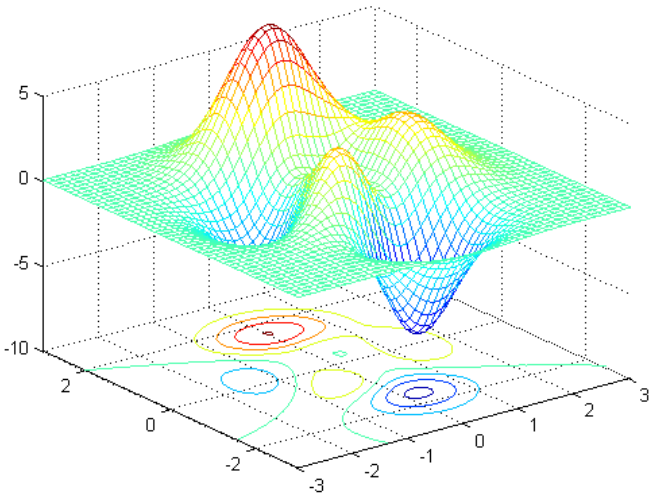
2.7 Graphiques à 3 dimensions

Matlab permet la représentation de graphique à trois dimensions pour ce faire, il faut passer en arguments à la fonction `meshc` trois matrices représentant les coordonnées selon les trois axes des points de la fonction.

Dans l'exemple ci-dessous, les matrices X et Y sont construite à l'aide de la fonction `meshgrid` de Matlab qui définis une grille carrée. Quand à la matrice Z elle est définie grâce

à la fonction `peaks` qui représente une Gaussienne en fonction des matrices X et Y.

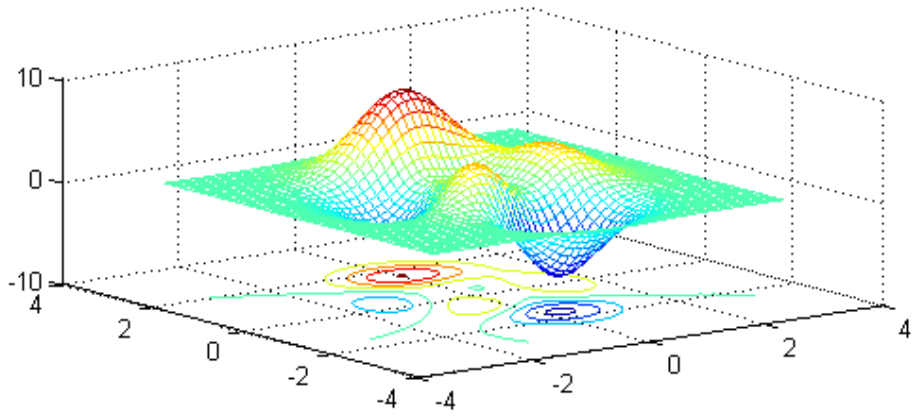
```
% — Graphique 3 dimensions ———————————————————————— %
figure ;                          % Nouvelle fenêtre

[X,Y] = meshgrid(-3 :.125 :3) ;   % Génération d'une grille de -3 à 3 pas de 0.125
Z = peaks(X,Y) ;                  % Distribution gaussienne en Z
meshc(X,Y,Z) ;                    % Affichage 3 dimensions
axis([-3 3 -3 3 -10 5])           % Étalonnage des axes
```

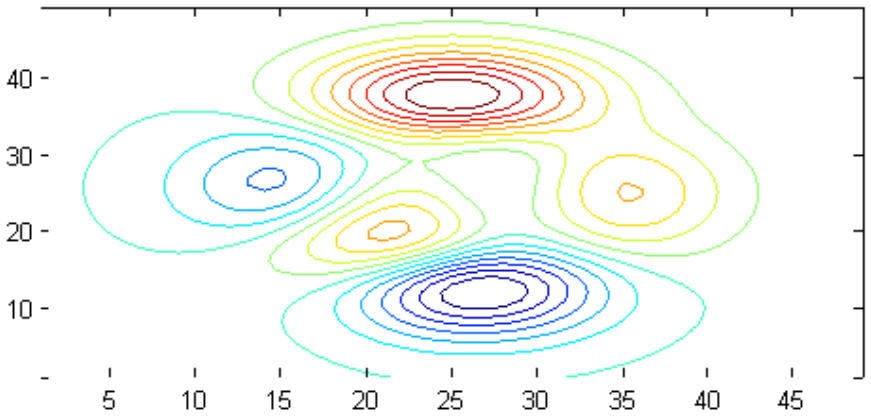


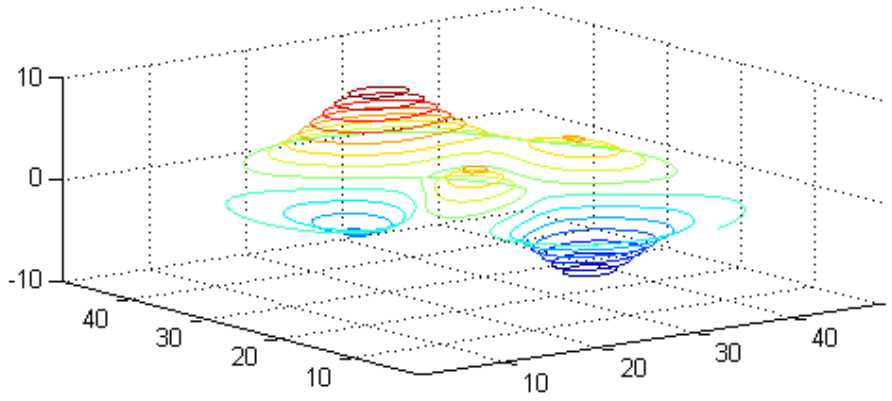
2.8 Graphique de contours 3 dimensions

Il est aussi possible de réaliser des contours. Ces fonctions relient les points de même hauteur (valeur) par des courbes. De plus, la couleur des courbes est liée à la valeurs qu'elle représente.

```
% — Graphique de Contours 3 dimensions ————————————————— %

figure ;                          % Nouvelle fenêtre

[X,Y] = meshgrid(-3 :.125 :3) ;   % Génération d'une grille de -3 à 3 pas de 0.125
Z = peaks(X,Y) ;                  % Distribution gaussienne en Z

subplot(3,1,1) ;                  % Zone supérieure
meshc(X,Y,Z) ;                    % Affichage 3 dimensions

subplot(3,1,2)                    % Zone centrale
contour(Z,15)                     % Contour 15 niveaux

subplot(3,1,3)                    % Zone inférieure
contour3(Z,15)                    % Contour 3 dimensions 15 niveaux
```


10
0
-10
4
2
0
-2
-4
-4
-2
0
2
4

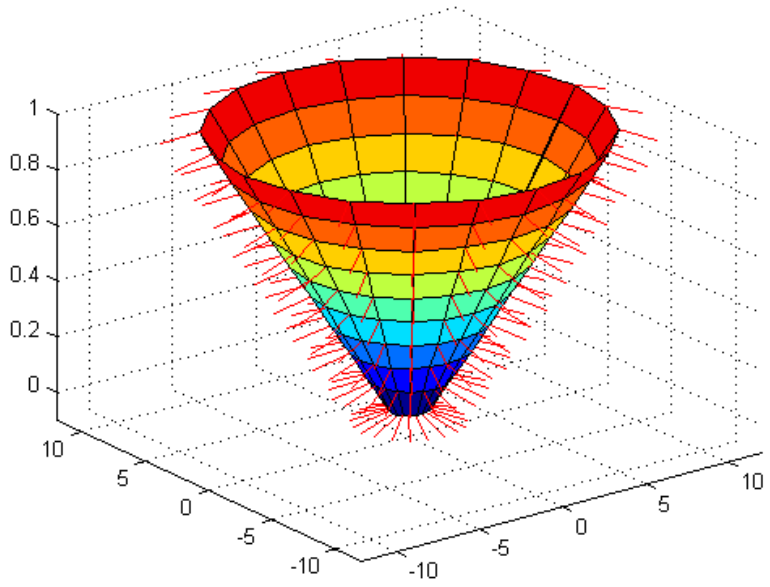

40
30
20
10
5
10
15
20
25
30
35
40
45


10
0
-10
40
30
20
10
10
20
30
40

2.9 Normales de surfaces

Enfin, il est possible sous Matlab de représenter des surfaces à trois dimensions ainsi que leur normale.

```
% — Normales de surfaces ———————————————————————————— %

figure ;                          % Nouvelle Fenêtre

[x,y,z] = cylinder(1 :10) ;       % Creation d'un cylindre 3 dimensions
surfnorm(x,y,z)                   % Affichage des surfaces et normes
axis([-12 12 -12 12 -0.1 1])      % Etalonnage des axes
```



Chapitre 3

Résolution de systèmes

3.1 Décompositions de matrice

Matlab permet de séparer des matrices selon plusieurs décompositions tels que LU, QR, SVD, etc.

```
% — Décomposition de matrice ————————————————————————————- %

A = [ 1 2 3 ;          % Définition de A
      4 5 6 ;
      7 8 9] ;

[L U] = lu(A)          % Décomposition LU

[Q R] = qr(A)          % Décomposition QR

[U S V] = svd(A)       % Décomposition USV
```

3.2 Valeurs et vecteurs propres

Voici comment demander à Matlab de déterminer les valeus propres et les vecteurs propres d'une matrice.

```
% — Valeurs et vecteurs propres ————————————————————————————-%

A = [ 1 2 3 ;              % Définition de A
      4 5 6 ;
      7 8 9] ;

[VecA, ValA] = eig(A)      % Renvoie les valeurs et vecteurs propres
```

3.3 Déterminant

Pour connaître le déterminant d'une matrice, il suffit d'utiliser la commande `det`

```
% — Valeurs et vecteurs propres ————————————————————————————-%

A = [ 1 2 3;      % Définition de A
      4 5 6;
      7 8 9];

det(A)            % Déterminant de la matrice A
 % ans =
     0
```

3.4 Simplification d'équations

Matlab dispose de fonctions permettant la simplification d'équations symboliques. Pour se faire, il suffit de déclarer les différents symboles via la commande `syms` et ensuite de données à la fonction `simple` l'équation.

```
% — Simplification d'équations ———————————————————————————— %

syms x;                     % Déclaration des symboles
f = cos(x)^2 + sin(x)^2;    % Fonction à simplifier
f = simple(f)               % Simplification
 % f =
 %     1

syms x;                     % Déclaration des symboles
g = cos(3*acos(x));         % Fonction à simplifier
g = simple(g)               % Simplification
 % g =
 %     4*x^3-3*x
```

3.5 Résolution symbolique

3.5.1 Dérivées et intégrales

Matlab est capable de dériver ou d'intégrer des expressions symboliques. Pour ce faire, il faut déclarer les symboles via la commande `sims`.Il est aussi possible de forcer Matlab à dériver par rapport à une variable précise en l'ajoutant en argument à la fonction

```
% — Dérivée symbolique ———————————————————————————— %

syms x ;                % Déclaration des symboles
diff(sin(x^2))          % Dérivation symbolique
% ans =
%     2*x*cos(x ^2)

% — Dérivée symbolique six fois ————————————————————————- %

syms t ;                % Déclaration des symboles
diff(t^6,6)             % Dérivation symbolique
% ans =
%     720)

% — Dérivée symbolique par rapport à une variable précise ———————— %

syms x t ;              % Déclaration des symboles
diff(sin(x*t^2), t)     % Dérivation symbolique
% ans =
%     2*t*x*cos(t^2*x))
```

```
% — Intégrale symbolique ———————————————————————————— %

syms x ;                    % Déclaration des symboles
int(-2*x/(1 + x^2)^2)       % Intégrale symbolique
% ans =
%     1/(x^2 + 1)

% — Intégrale symbolique par rapport à z ———————————————————— %

syms x z ;                  % Déclaration des symboles
int(x/(1 + z^2), z)         % Intégrale symbolique
% ans =
%     x*atan(z)

% — Intégrale de 0 à 1 ————————————————————————————- %

syms x ;                    % Déclaration des symboles
int(x*log(1 + x), 0, 1)     % Intégrale symbolique
% ans =
%     1/4

% — Intégrale symbolique de sin(t) à 1 – ———————————————————- %

syms x t ;                  % Déclaration des symboles
int(2*x, sin(t), 1)         % Intégrale symbolique
% ans =
%     cos(t)^2
```

3.5.2 Équations et systèmes

Pour résoudre symboliquement une équation ou un système d'équations, il faut tout d'abord déclarer les symboles via `syms`. Une fois fait, la commande `solve` permet de résoudre les équations.

Par défaut, Matlab définis des priorités dans les symboles pour déterminer par rapport à quel symbole il va résoudre. Cependant, il est possible d'imposer le symbole comme montré ci-dessous en passant un argument supplémentaire à la fonction.

```
% — Résolution symbolique ————————————————————————————- %

syms a b c x;                   % Définition des symboles
solve('a*x^2 + b*x + c')        % Résolution
% ans =
%     -1/2*(b-(b^2-4*a*c)^(1/2))/a
%     -1/2*(b+(b^2-4*a*c)^(1/2))/a

syms t m;                       % Définition des symboles
solve('sin(t + m)')             % Résolution
% ans =
%     -m

syms a b c x;                   % Définition des symboles
solve('a*x^2 + b*x + c','b')    % Résolution pour la variable b
% ans =
%     -(a*x^2+c)/x
```

3.6 Résolution de systèmes linéaires

Voici comment résoudre un système linéaire. Dans ce cas-ci, l'intersection de deux droites.

```
% — Résolution d'un système ———————————————————————————— %

syms x;                                 % Définition des symboles
S = solve('x + 2*y = 1','x - 1*y = 5');; % Résolution
S = [S.x S.y];                          % Récupération des solutions
% S =
%     [ 11/3, -4/3]
```

3.7 Résolution symbolique de systèmes linéaires

Matlab permet aussi de résoudre symboliquement des systèmes linéaires. Il faut bien entendu commencer par préciser les symboles via la commande `syms`.

```
% — Résolution symbolique de système paramétrique ——————————————————- %

syms a x ;                                  % Définition des symboles
S = solve('x + a*y = 1','x - 1*y = 5') ;    % Résolution
S = [S.x S.y]                               % Récupération des solutions
% S =
%     [ (1+5*a)/(1+a), -4/(1+a)]
```

3.8 Résolution d'équations différentielles

Matlab permet la résolution d'équations différentielles que l'on dispose de conditions initiales ou non.

```
% — Résolution d'équations différentielles 1 ———————————————————————— %

dsolve('Dy = a*y', 'y(0) = b')                  %
% ans =
%     b*exp(a*t)

% — Résolution d'équations différentielles 2 ———————————————————————— %

dsolve('D2y = -a^2*y', 'y(0) = 1', 'Dy(pi/a) = 0')     %
% ans =
%     exp(a*i*t)/2 + 1/(2*exp(a*i*t))
```

```
% — Résolution de systèmes d'équations différentielles sans condition initiale ———- %

syms x y ;                                  % Définition des symboles
Z = dsolve('Dx = y', 'Dy = -x') ;           % Résolution différentielle
Z = [Z.x Z.y]                               % Récupération des solutions

% z =
%    [ C1*sin(t)+C2*cos(t), C1*cos(t)-C2*sin(t)]
```

```
% — Résolution de systèmes d'équations différentielles avec conditions initiales ——— %

syms x y ;                                                  % Définition des symboles
Z = dsolve('Dx = y', 'Dy = -x','y(0) = 2,x(0) = 1') ;       % Résolution différentielle
Z = [Z.x Z.y]                                               % Récupération des solutions

% z =
%     [ cos(t)+2*sin(t), -sin(t)+2*cos(t)]
```

Chapitre 4

Statistiques

4.1 Statistiques descriptives

L'ensemble des fonctions suivantes prennent en arguments soit un vecteur, soit une matrice. Dans le cas d'une matrice chaque colonne est traitée comme une variable différente et le résultat des fonctions est donc un vecteur.

```
% — Statistiques descriptives ———————————————————————— %

x = [1 5 74;      % Définition de la matrice
     1 5 74;
     3 6 10;
     5 9 20;
     7 8 99];

geomean(x),       % Moyenne géométrique
harmmean(x),      % Moyenne harmonique
mean(x),          % Moyenne arithmétique
median(x)         % Élément milieu du vecteur trie
mode(x),          % Valeur la plus fréquente
var(x),           % Variance (n-1)
std(x),           % Déviation standard (n-1)
range(x),         % Intervalle des valeurs

max(x),           % Valeur maximum
min(x),           % Valeur minimum

prod(x)           % Produit des éléments par colonnes
sum(x)            % Somme des éléments par colonnes
sort(x)           % Tri par colonnes
```

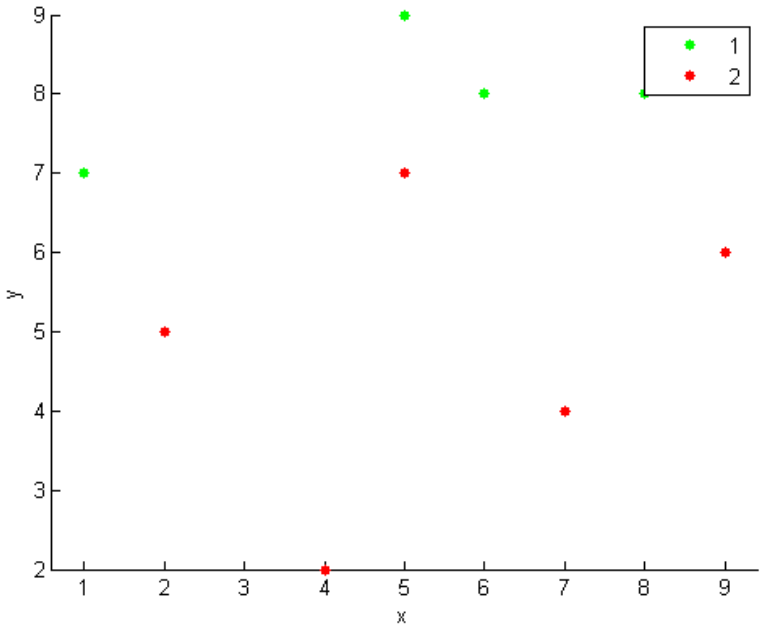
4.1.1 Affichage simple de données

Matlab permet l'affichage de nuages de points. Pour ce faire, il faut lui fournir les coordonnées des points en deux vecteurs x et y.

Il est de plus possible de classer les points en différents groupes via une troisième matrice. Les points ayant la même étiquette dans cette matrice sont considérés comme faisant partie

du même groupe.

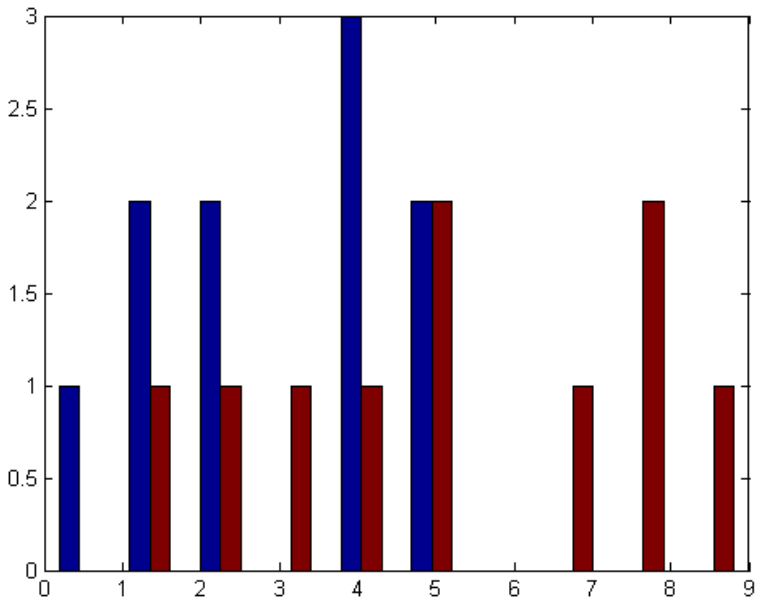
```
% — Affichage simple de données ———————————————————————— %

x = [ 1 2 8 5 7 9 6 4 7 5] ;          % Coordonnée en x des points
y = [ 7 5 8 9 4 6 8 2 4 7] ;          % Coordonnée en y des points
group = [ 1 2 1 1 2 2 1 2 1 2] ;      % Regroupement des points
gscatter(x,y,group) ;                 % Affichage
```

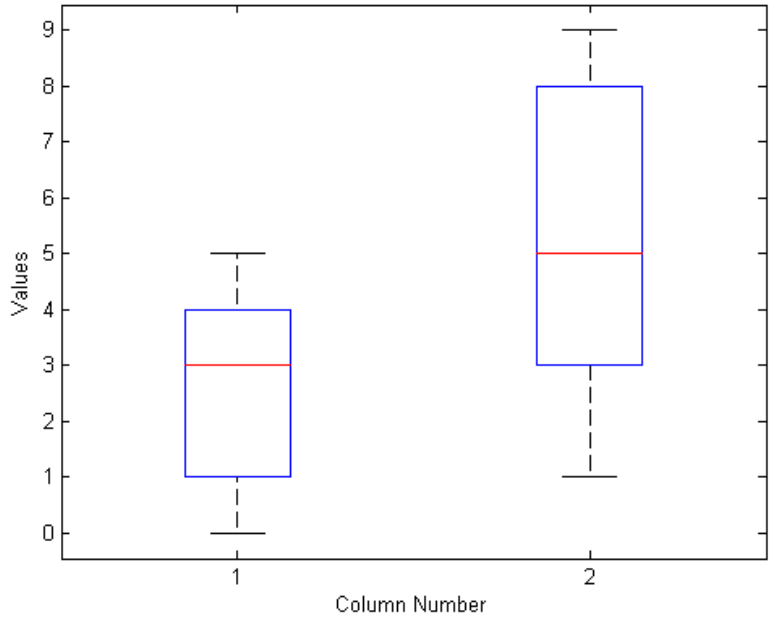


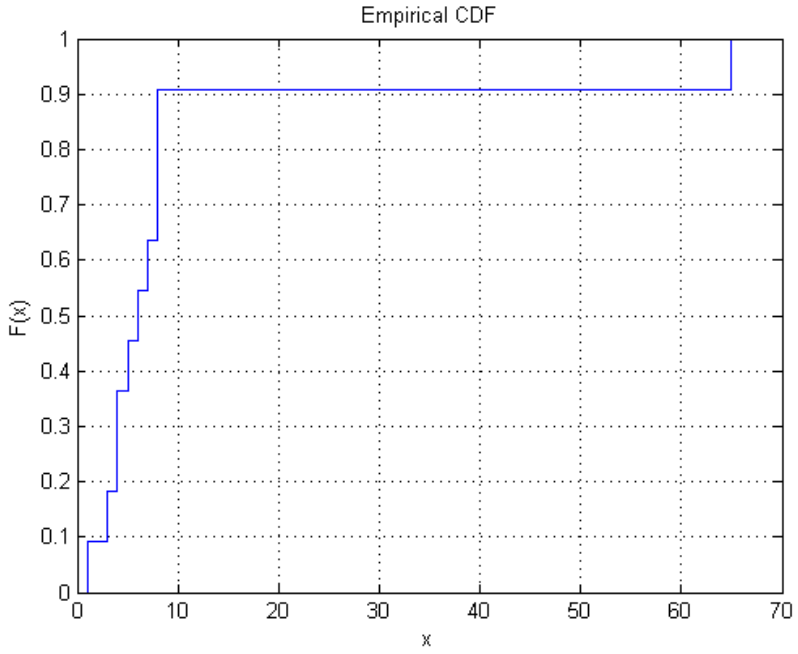
4.1.2 Affichage avancé de données

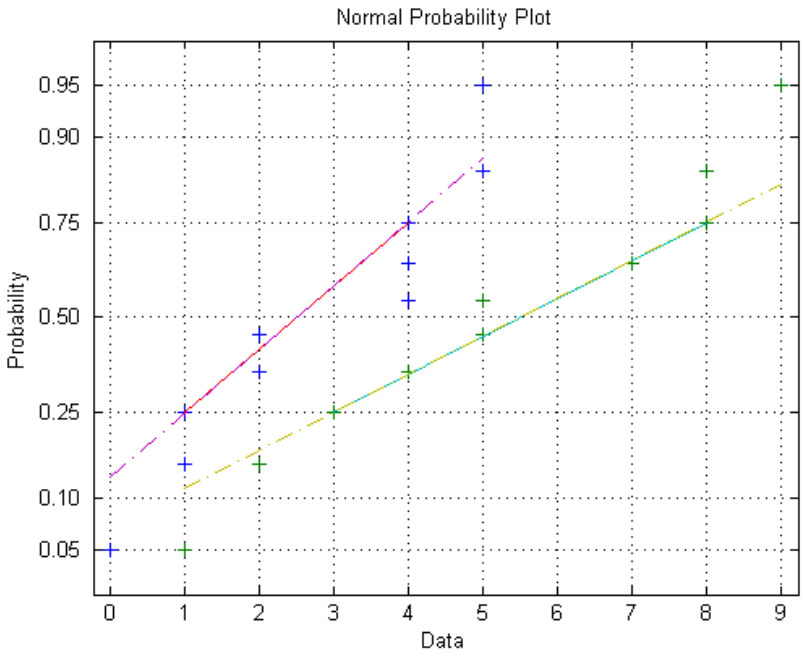
Il est aussi possible d'afficher les données de différentes manière comme par exemple en histogramme, en boxplots, en distribution cumulative, etc.

```
% Affichage avancé de donnée ---------- %
x = [ 1 2;                          % Vecteur de données
      2 4;
      1 3;
      4 8;
      2 5;
      5 8;
      4 9;
      5 7;
      0 1;
      4 5];

y = [ 1 5 65 8 4 8 6 8 3 4 7];      % Autre vecteur de données

figure;                             % Nouvelle fenêtre
hist(x)                             % Histogram (1 par colonne)

figure;                             % Nouvelle fenêtre
boxplot(x)                          % Boxplots (1 par colonne)

figure;                             % Nouvelle fenêtre
cdfplot(y)                          % Distribution cumulative

figure;                             % Nouvelle fenêtre
normplot(x)                         % Probabilité normale (un par colonne).
```




Values
Column Number
0
1
2
3
4
5
6
7
8
9
1
2


Empirical CDF
F(x)
x
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
10
20
30
40
50
60
70



4.1.3 Corrélation entre variables

La fonction `corrcoef` permet de calculer la corrélation entre différentes variables placées dans une matrice. Chaque colonne représentant une variable.

```
% — Corrélation ———————————————————————————— %
% R = corrcoef(X) returns a matrix R of correlation coefficients calculated
% from an input matrix X whose rows are observations and whose
% columns are variables.

x = [ 1 2;
      2 4;
      1 3;
      4 8
      2 5]

disp('Corrélations :')    % Affichage
corrcoef(x),              % Calcul de corrélation
```

4.2 Distribution de probabilités

La toolBox statistics de Matlab fournis 5 fonctions qui peuvent être utilisées avec les différentes loi de probabilité.

– Fonction de densité de probabilité :

Ici, la fonction est appelée dans le cas d'une loi normal :

```
% — densité de probabilité ———————————————————————— %

x = [-3 :0.1 :3] ;
f = normpdf(x,0,1) ;
```

Pour chaque pdf, le premier argument est l'ensemble des données, les arguments suivants sont les paramètres nécessaire pour les lois de probabilités. (*dans l'exemple nous avons une loi normale de $\mu = 0$ et de $\sigma = 1$*)

– Fonction de distribution cumulative :

```
% — Distributions cumulatives ————————————————————— %

x = [-3 :0.1 :3] ;
p = normcdf(x,0,1) ;
```

La variable p contient les probabilités associées avec la distribution cumulative de probabilité de $\mu = 0$ et de $\sigma = 1$.

– Fonction de distribution cumulative inverse :

```
% — Distributions cumulatives inverse ————————————————— %

x = [-3 :0.1 :3] ;
xnew = norminv(normcdf(x,0,1),0,1) ;
```

– Générateur aléatoire :

Il est possible de générer des valeurs aléatoires pour chaque distribution. (*dans l'exemple : la commande suivante renvoie un nombre aléatoire pour la loi normale de $\mu = 0$ et de $\sigma = 1$*)

```
% —Générateur aléatoire ———————————————————————— %

x = normrnd(0,1)
```

En ajoutant les arguments M et N : normrnd(0,1,M,N), la fonction renvoie alors une matrice de nombre aléatoire suivant la loi de probabilité.

– Moyenne et variance

La commande suivante permet de récupérer la moyenne et la variance d'une fonction de probabilité.

```
% — Moyenne et variance ———————————————————————— %

[m,v] = normstat(0,1)
```

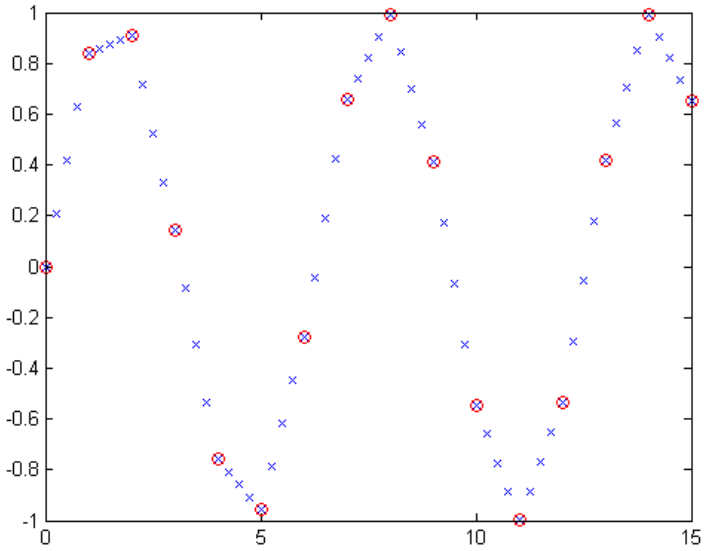
Les différentes lois de probabilité sont :

```
% — Différentes lois de probabilités ———————————————————————— %

beta :   Beta distribution
bino :   Binomial distribution
chi2 :   Chi-Square distribution
exp :    Exponential distribution
f :      F distribution
gam :    Gamma distribution
geo :    Geometric distribution
hyge :   Hypergeometric distribution
logn :   Lognormal distribution
mvn :    Multivariate normal distribution
mvt :    Multivariate t distribution
nbin :   Negative binomial
poiss :  Poisson cumulative distribution
unid :   Uniform (discrete) distribution
unif :   Uniform (continuous) distribution
weib :   Weibull distribution
```

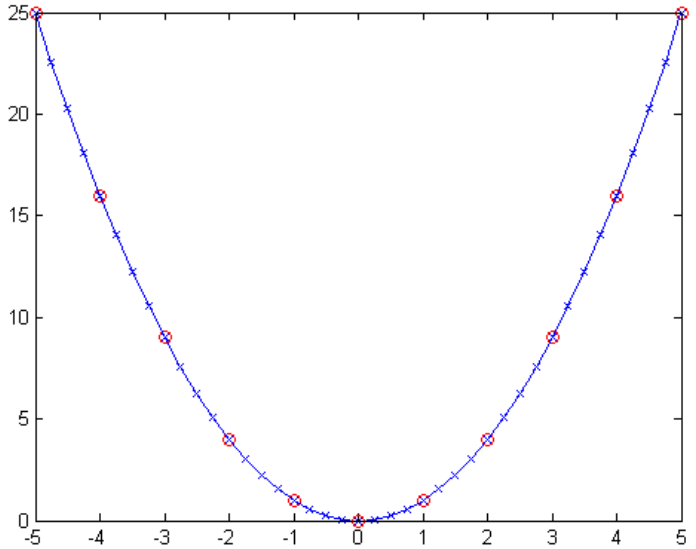
4.3 Interpolations

Matlab permet d'interpoler des points entre les données qui lui sont fournies. Il est possible d'effectuer des interpolations linéaire, cubic, etc.

```
% — Interpolation linéaire ———————————————————————————— %

figure ;                                    % Nouvelle fenêtre

x = 0 :15 ;                                 % Vecteur du temps
y = sin(x) ;                                % Sinus correspondant à x
plot(x,y,'or') ;                            % Affichage
hold on                                     % Maintient du graphique

xInterp = 0 :.25 :15 ;                      % Interpolation entre chaque point
yInterp = interp1(x,y,xInterp, 'linear') ;  % Interpolation de y en fonction de x
plot(xInterp,yInterp,'xb') ;                % Affichage
```




```
% — Interpolation cubic ———————————————————————— %

figure ;                                  % Nouvelle fenêtre

x = - 5 :5 ;                              % Vecteur du temps
y = x.^3 ;                                % Élévation au cube correspondant à x
plot(x,y,'or') ;                          % Affichage
hold on                                   % Maintient du graphique

xInterp = -5 :.25 :5 ;                    % Interpolation entre chaque points
yInterp = interp1(x,y,xInterp, 'spline') ; % Interpolation de y en fonction de x
plot(xInterp,yInterp,'x-b') ;             % Affichage
```



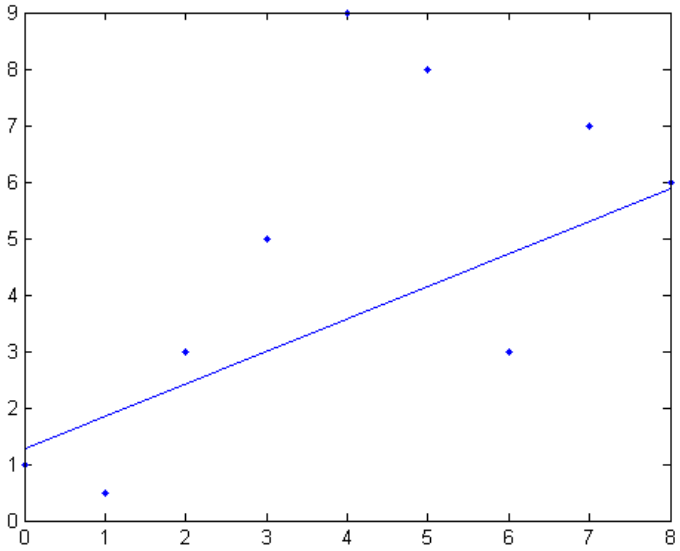
Différent type d'interpolation :

```
% — Différents types d'interpolation ———————————————————————— %

nearest   Nearest neighbor interpolation
linear    Linear interpolation (default)
spline    Cubic spline interpolation
pchip     Piecewise cubic Hermite interpolation
cubic     (Same as 'pchip')
v5cubic   Cubic inte
```

4.4 Régression linéaire

Matlab permet aussi d'effectuer des régressions linéaires. Pour ce faire, il faut disposer de deux vecteurs, l'un contenant les données en x et leur occurrence et l'autre contenant les données correspondantes en y.

```
% — Régression linéaire ———————————————————————————— %

X = [1 1 1 1 1 1 1 1 1];                          % Nombre d'occurrence des valeurs
1 0.5 3 5 3 6 7 9 8]';                             % Valeurs en x
y = [0 1 2 3 6 8 7 4 5]';                         % valeurs en y

[B,BINT,R,RINT,STATS] = regress(y,X,0.05)         % Fonction de régression

figure;                                           % Nouvelle fenêtre
plot(y,X( :,2),'b.')                              % Affichage des points
refline(B(2),B(1))                                % Affichage de la droite
```



Chapitre 5

Temps

5.1 Connaître le temps d'exécution

Pour connaitre le temps d'exécution d'un script, il suffit de le placer les commandes d'intérêts entre les mot-clés `tic` et `toc`. `tic` déclenche le chronomètre tandis que `toc` arrête le chronomètre et affiche l'intervalle de temps.

```
% — Temps d'exécution ———————————————————————————— %

tic ;                       % Lancement du chronomètre

for i = 0 :0.01 :1000       % Instructions d'intérêts
  a = i * 3 ;
end

toc                         % Arrêt du chronomètre
```

5.2 Programmation parallèle

Matlab permet le travail en parallèle. Cependant pour ce faire la librairie suivante est requise : *Parallel Computing Toolbox*

Pour travailler en parallèle, il fait tout d'abord créer des `pools` dans la session Matlab et ensuite utiliser la commande `parfor` pour répartir le travail entre les différents `pools`. L'exemple suivant montre comment effectuer simultanément sur 3 cœurs différents la recherche de valeurs propres de grandes matrices.

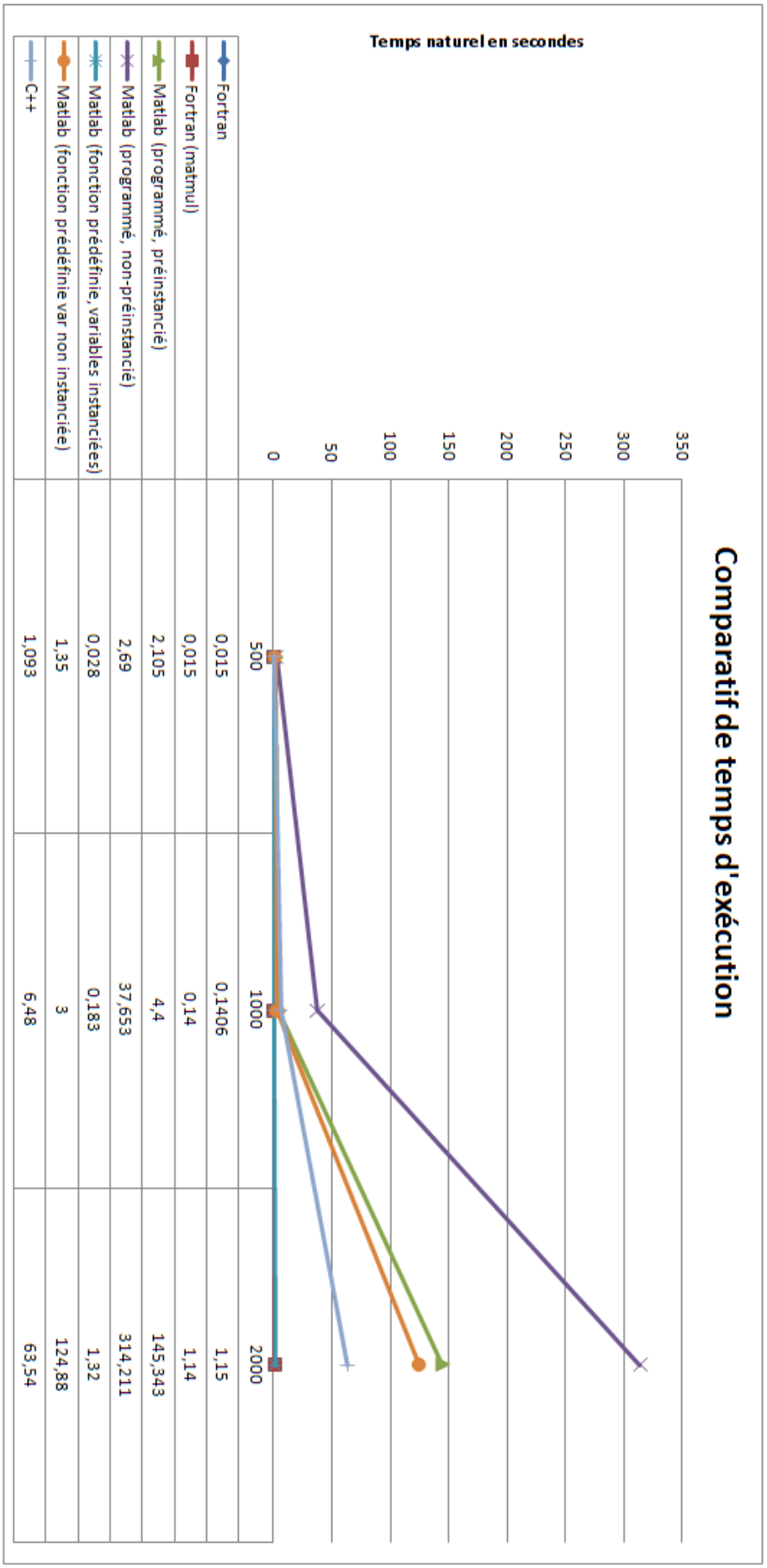
```
% — Exécutions parallèles ———————————————————————- %

matlabpool(3)                   % Création des pools
parfor i=1 :3,                  % Répartition entre les pools
  c( :,i) = eig(rand(1000)) ;   % Tâches
end
```

5.3 Comparatif Fortran, C++, Matlab

Le graphique suivant représente l'étude du temps d'exécution dans différentes langages. L'étude est faite sur base de la comparaison du temps d'exécution d'un petit algorithme de calcul de produit matriciel pour des matrices carrées de 500,1000,2000 lignes. Sous matlab, quatre cas sont envisagé. Soit les variables sont pré-instanciées soit pas, soit l'algorithme est implémenté manuellement soit c'est la fonction prédéfinie qui est utilisée.

Voici l'algorithme vers Matlab :

```
% — Algorithme 1 ————————————————————————————- %

clearvars ;                                  %

A = zeros(2000,2000) ;                       % Instanciation des variables
B = zeros(2000,2000) ;                       % Instanciation des variables
C = zeros(2000,2000) ;                       % Instanciation des variables

tic                                          % Déclenche le Chronomètre

for i = 1 :2000                              % Initialisation des variables
  for j=1 :2000
    A(i,j) = i ;
    B(i,j) = i ;
    C(i,j) = 0 ;
  end
end


for i = 1 :2000                              % Calcul du produit matriciel
  for j=1 :2000
    for k=1 :2000
      C(i,j) = C(i,j) + (A(i,k) * B(k,j)) ;
    end
  end
end

toc                                          % Arrêt du chronomètre
```



	500	1000	2000
Fortran	0,015	0,1406	1,15
Fortran (matmul)	0,015	0,14	1,14
Matlab (programmé, préinstancié)	2,105	4,4	145,343
Matlab (programmé, non-préinstancié)	2,69	37,653	314,211
Matlab (fonction prédéfinie,variables instanciées)	0,028	0,183	1,32
Matlab (fonction prédéfinie var non instanciée)	1,35	3	124,88
C++	1,093	6,48	63,54

Le tableau précédent met en évidence la lenteur de Matlab dans lorsque le programmeur n'est pas extrêmement méticuleux dans sa manière de faire. Par exemple, dans notre cas le temps d'exécution est multiplié par trois lorsque les matrices ne sont pas pré-instanciées. De plus, l'utilisation de fonction prédéfinie par Matlab ou non change considérablement le temps d'exécusion.

La conclusion est que le temps de calcul de Matlab reste tout à fait raisonnable lorsque le code est bien réfléchis mais que la moindre erreur peut entrainer des temps de calcul extrêmement long.

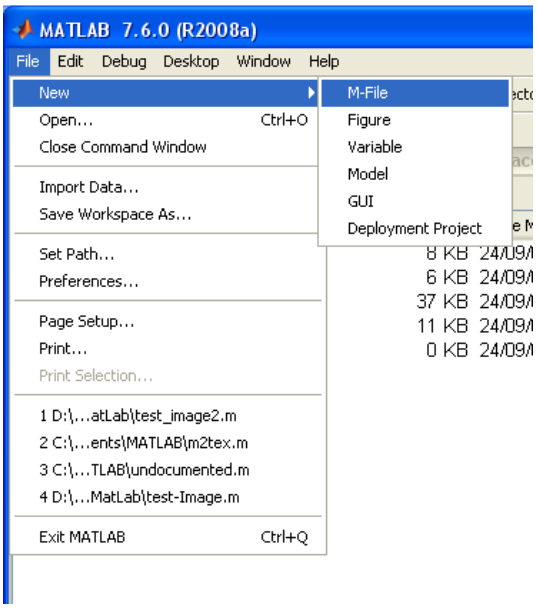
Chapitre 6

Programmation

6.1 Les m-files

Un m-file est un fichier contenant une suite d'instructions que Matlab peut exécuter. Un m-file peut aussi être utilisé comme fichier de librairie contenant des fonctions définies par l'utilisateur.

Pour créer un m-file, utilisez le menu *File →new →M-file.*



6.2 Hello world

Une fois le M-file créé, tapez ceci :

```
% — Hello world ———————————————————————————— %

disp('Hello world')
```

```
Hello world
>>
```

6.3 Les opérateurs logiques

```
% — Opérateurs logiques ———————————————————————————— %

<          ... est plus petit que ...
>          ... est plus grand que ...
<=         ... est plus petit ou égal à ...
>=         ... est plus grand ou égal à ...
==         ... est égal à ...
~=         ... n'est pas égal à ...
&          ... est vrai et ... aussi (pour tableaux)
|          ... est vrai ou ... est vrai, ou les deux (pour tableaux)
&&         ... est vrai et ... aussi
||         ... est vrai ou ... est vrai, ou les deux
~          ... n'est pas vrai
xor(x,y)   ... est vrai ou ... est vrai
any(x)     ... vrai si un des éléments de x est non nul
all(x)     ... vrai si tous les éléments de x sont nuls
```

6.4 Les mots gardés

Les mots ou caractères suivant ont une signification particulière dans le langage de Matlab.

```
% — Mots/symboles gardés ——————————————————————————— %

:       Create vectors, subscript arrays, specify for-loop iterations
( )     Pass function arguments, prioritize operators
[ ]     Construct array, concatenate elements, specify multiple outputs from function
{ }     Construct cell array, index into cell array
.       Insert decimal point, define structure field, reference methods of object
.( )    Reference dynamic field of structure
..      Reference parent directory
...     Continue statement to next line
,       Separate rows of array, separate function input/output arguments,
        separate commands
;       Separate columns of array, suppress output from current command
%       Insert comment line into code
% %     Insert block of comments into code
!       Issue command to operating system
' '     Construct character array
@       Construct function handle, reference class directory
```

6.5 Entrées / sorties

6.5.1 Utilisateurs

Les commandes suivantes permettent l'interaction avec l'utilisateur lors de l'exécution d'un script. L'utilisateur peut dans l'exemple ci-dessous saisir un nombre que le script pourra alors utiliser. Dans le sens inverse la commande `disp` permet d'afficher des variables à l'utilisateur.

```
% — Exemple 1 ——————————————————————————————- %
n = input('Saisissez un nombre :') ;     Saisie de l'utilisateur
disp(n) ;                                Sortie vers l'affichage Matlab
```

6.5.2 Disques

Il est aussi possible d'exporter ou d'importer des données sur le disque dur via les deux commandes suivantes.

```
% — Exemple 1 ——————————————————————————————- %

save 'D : ...save.txt'     Sauvegarde le workspace
load 'D : ... save.txt'    Charge le workspace
```

6.6 Le contrôle de l'exécution

6.6.1 Boucle FOR

La boucle `FOR` permet d'effectuer des opérations pour un nombre d'itérations définis. (*Par exemple : pour n = 0 jusqu'à 20 par pas de deux, effectuer les opérations suivantes*).

L'avantage de la boucle `FOR` sur la boucle `WHILE` est sa simplicité d'écriture dans le cas d'un nombre d'itérations définis et bien connu à l'avance (*par exemple, le parcours d'un tableau*).

Autre remarque, lors de l'exécution de la boucle `FOR`, la variable qui sert à boucler est accessible en lecture et en écriture. Il est donc possible de réduire ou d'augmenter le nombre d'itérations au cours de l'exécution de la boucle.

```
% — Exemple 1 ——————————————————————————————- %

for n = 1 :5                % Boucle pour n allant de 1 à 5 inclus par pas de 1
  disp(n)                   % Affichage
end                         % Fin de boucle

% — Exemple 2 ——————————————————————————————- %

for n = 8 :-2 :0            % Boucle pour n allant de 8 à 0 par pas de 2
  disp(n)                   % Affichage
end                         % Fin de boucle

% — Exemple 3 ——————————————————————————————- %

for n = [ 1 9 3 5 6 7 ]     % Boucle pour n égal chaque valeur du vecteur
  disp(n)                   % Affichage
end                         % Fin de boucle
```

6.6.2 Boucle WHILE

Le boucle Permet d'effectuer des opérations de manière répétée jusqu'à ce qu'une condition soit falsifiée. (*Par exemple : tant que la solution n'est pas précise à 4 décimales, continuer à*

chercher une solution plus précise.)

```
% — Exemple 1 ———————————————— %

n = 5;
m = 8;
while (n < 10 && m > 0)     % Boucle tant que n est inférieur à 10
                            % et m est supérieur à 0
  n = n + 1;                % Incrémente n
  m = m - 1;                % Décrémente m
  disp(n);                  % Affichage
  disp(m);                  % Affichage
end                         % Fin de boucle
```

6.6.3 Instruction de choix IF

L'instruction `IF` est une instruction de choix. Autrement dit, en fonction que son gardien sera évalué vrai ou faux, la commande exécutera un groupe d'instructions ou l'autre.

```
% — Exemple 1 ———————————————— %

n = 5;
m = 8;
if (n > 0)                  % Gardien 1

  if ( n > 5 && m < 0)      % Gardien 1 et gardien 2
    disp ('ici 1')          % Affichage
  elseif (n == 5)           % Si gardien1 et non gardien2 et gardien 3
    disp ('ici 2')          % Affichage
  else                      % Si gardien1 et non gardien2 et non gardien 3
    disp('ici 3')           % Affichage
  end                       %

else                        % Si gardien 1 est faux
  disp('ici 4')             % Affichage
end                         %
```

6.6.4 Instruction de choix SWITCH

L'instruction `SWITCH` est une instruction de choix comme le `IF` mais avec la particularité de pouvoir effectuer plus de branchements que le `IF`.

La commande `SWITCH` doit être utilisée dans le cas ou, par exemple, en fonction de la valeur d'une variable, on effectue différentes opérations. Attention toute fois, que le nombre de valeurs possibles de cette variable doit être restreint pour conserver une certaine lisibilité du code.

Le mot clé `break` signifie que l'on arrête la commande `SWITCH` et que l'on transfert l'exécution au mot clé `end`.

```
% — Exemple 1 ———————————————— %

n = input('Pour sauver jack tapez 1,      % Demande de saisie de l'utilisateur
Jessy tapez 2, Brian tapez 3 :') ;

switch n

  case 1                                  % Si n vaut 1
    disp('Jack est sauvé')
  break

  case 2
    disp('Jessy est sauvé')               % Si n vaut 2
  break

  case 3
    disp('Brian est sauvé')               % Si n vaut 3
  break

  otherwise
    disp('Vous n"avez sauvé personne')    % Si n est différent de 1,2,3

end
```

6.7 Les fonctions

6.7.1 Les fonctions simples

Matlab permet de définir des fonctions. Une fonction est un ensemble d'instructions regroupées de manière à ne pas devoir les répéter régulièrement.

Une fonction peut prendre des arguments et renvoyer des valeurs. (*Par exemple, si l'on fourni à une fonction une date de naissance, cette fonction peut renvoyer l'âge de la personne*)

La définition d'une fonction se fait de la manière suivante :

`function` *arguments de sortie* = nomDeMaFonction (*arguments d'entrée*).

Attention, pour que matlab reconnaisse une fonction et sache l'utiliser à partir de l'espace de travail, il est obligatoire que le nom du M-file soit identique au nom donné à la fonction.

```
% — Fonction simple ————————————————————————————- %

% Fonction isOdd
% Cette fonction prend en argument un nombre et renvoie
% true si ce nombre est impair, false si non.
%
% Argument : un nombre
% Valeur de retour : un boolean
% Code placé dans isOdd.m

function p = isOdd(n)

if (mod(n,2) == 1)        % Si la division modulaire laisse un reste
  p = true;               % Renvoie vrai
else
  p = false;              % renvoie faux
end
```

6.7.2 Les fonctions récursives

Une fonction est dite récursive si elle s'appelle elle-même. La fonction suivante est récursive et permet de calculer la factoriel d'un nombre.

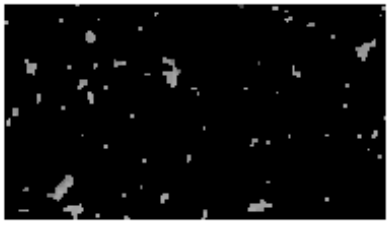
```
% — Fonction récursive ————————————————————————————- %

% Fonction fact
% Cette fonction prend en argument un entier positif
% et renvoie la factoriel correspondante.
%
% Argument : un entier positif
% Valeur de retour : un entier positif
%

function y = fact(n)

if (n <= 1)               % Si l'on est dans le cas de base
  y = 1;
else
  y = n * fact(n-1);      % Si non, appel récursif
end
```

6.8 Exemple de programme

6.8.1 Exemple 1

L'objectif de ce programme est d'analyser le pourcentage de gros éléments visibles dans le parement en béton. Un premier traitement a été effectué pour seuiller l'image et contraster les différentes zones. Le programme ci-dessous va ensuite charger cette image pour calculer le

pourcentage de la zone noir par rapport à la zone blanche.



```
% — Exemple 1 ——————————————————————————————- %

img12 = imread('img12.png');                    % Chargement de l'image
img4 = imread('img4.jpg');                      % Chargement de l'image

figure;                                         % Nouvelle fenêtre

subplot(1,2,1)                                  % Zone supérieure
imshow(img4)                                    % Affichage

subplot(1,2,2)                                  % Zone droite
imshow(img12)                                   % Affichage

[m,n] = size(img12);                            % Dimensions de l'image
blackPoints = 0.0;                              % Initialisation
otherPoints = 0.0;                              % Initialisation

for i = 1 :m                                    % Boucle verticale
  for j = 1 :n                                  % Boucle horizontale

    if (img12(i,j) == 0)                        % Si le point est noir
      blackPoints = blackPoints + 1;            % Incrémentation
    else                                        % Si non
      otherPoints = otherPoints + 1;            % Incrémentation
    end

  end
end

disp(blackPoints)                               % Affichage
disp(otherPoints)                               % Affichage

percent = blackPoints / (blackPoints + otherPoints);    % Pourcentage
disp('Le pourcentage d"éléments non noir est de')
disp(percent)
```

6.8.2 Exemple 2

Chapitre 7

Traitement d'images

7.1 Chargement / affichage d'une image

Code

Le code suivant permet le chargement et l'affichage d'images. La commande `close all` est importante car elle permet de fermer toutes les fenêtres graphiques actuellement ouvertes.

```
% — Chargement/Affichage d'une image———————————————————————— %

close all                          % Fermeture de tous les graphiques

img1 = imread('img1.jpg');         % Chargement de l'image dans le workspace
figure;                            % Nouvelle figure
imshow(img1);                      % Affichage
axis image;                        % Type d'axes de l'image [1]
colorbar                           % Ajoute la bar de coloration à droite

% [1]
% axis equal sets the aspect ratio so that the data units are the same in every direction.
% The aspect ratio of the x-, y-, and z-axis is adjusted automatically according
% to the range of data units in the x, y, and z directions.
% axis image is the same as axis equal except that the plot box fits
% tightly around the data.
```

Résultats

7.2 Modifier une image

Objetcif

L'objectif de ce code et de récupérer ou de modifier la couleur d'un pixel. Ici, l'exemple travail en *RGB* mais il est possible de travailler en *HSV* ou autre.

Code

```
% — Récupérer / Modifier pixel ————————————————————————————- %

img1 = imread('img1.jpg') ;                    % Chargement de l'image dans le workspace

disp('Valeur du pixel 250,250')                % Affichage de texte
disp(strcat('R = ', num2str(img1(250,250,1)))),  % Composante de rouge du pixel (250,250)
disp(strcat('G = ', num2str(img1(250,250,2)))),  % Composante de vert du pixel (250,250)
disp(strcat('B = ', num2str(img1(250,250,3)))),  % Composante de bleu du pixel (250,250)

disp('Composante de rouge (y = 100) :")        % Affichage texte
img1(1 :10,100,1)                              % Plusieurs composantes de rouge
```

Résultats

```
Valeur du pixel 250,250
R =31
G =86
B =151
Composante de rouge de coordonnées y = 100 :

ans =

    7
    5
    2
    2
    4
    5
    4
    3
    4
    4

Valeur du pixel 250,250
R =31
G =86
B =151
Composante de rouge (y = 100):

ans =

    7
    5
    2
    2
    4
    5
    4
    3
    4
    4
```

7.3 Décomposition d'une image en ses composantes couleurs

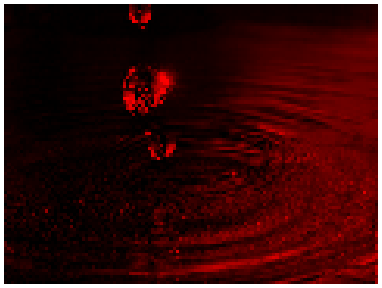
Objectif

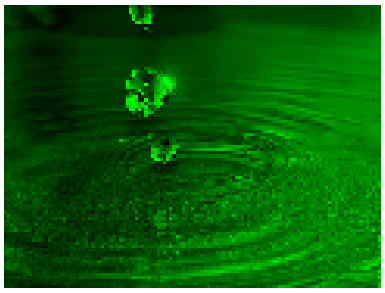
L'objectif est ici de décomposer une image en ses composantes. Dans un premier temps, la décomposition s'effectue selon les composantes *RGB* et dans un second temps selon les composantes *HSV*.

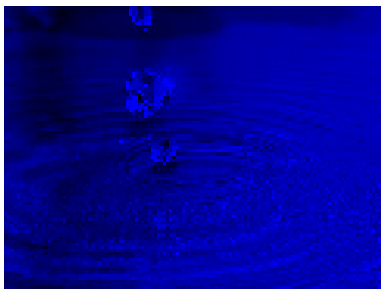
Code

```
% — Diviser une image en ses composantes RGB ———————————————————— %

img1 = imread('img1.jpg') ;                 % Chargement de l'image dans le workspace
figure ;                                    % Nouvelle fenêtre

img1R = img1 ;
img1R( :, :,2) = 0 ;                        % Mise à zéro de la composante de vert
img1R( :, :,3) = 0 ;                        % Mise à zéro de la composante de bleu
subplot(3,1,1) ;                            % Positionnement en zone supérieure
imshow(img1R) ;                             % Affichage

img1G = img1 ;
img1G( :, :,1) = 0 ;                        % Mise à zéro de la composante de rouge
img1G( :, :,3) = 0 ;                        % Mise à zéro de la composante de bleu
subplot(3,1,2) ;                            % Positionnement en zone centale
imshow(img1G) ;                             % Affichage

img1B = img1 ; img1B( :, :,1) = 0 ;         % Mise à zéro de la composante de rouge
img1B( :, :,2) = 0 ;                        % Mise à zéro de la composante de vert
subplot(3,1,3) ;                            % Positionnement en zone inférieure
imshow(img1B) ;                             % Affichage
```

Résultats



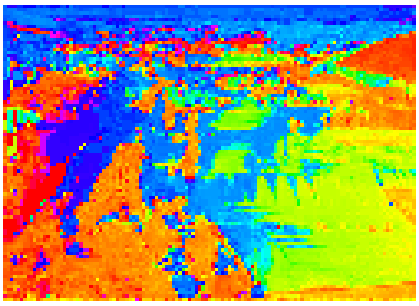


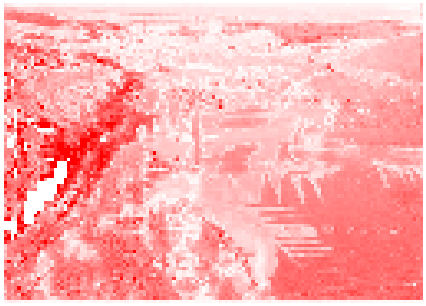


Code

```
% — Diviser une image en ses composantes HSV ——————— %

img1 = imread('img1.jpg') ;     % Chargement de l'image dans le workspace

figure ;                        % Nouvelle fenêtre

img1 = rgb2hsv(img1) ;          % Conversion en HSV

img1H = img1 ;
img1H( :, :,2) = 1 ;            % Mise à un de la saturation
img1H( :, :,3) = 1 ;            % Mise à un de la luminosité
subplot(3,1,1) ;                % Positionnement en zone supérieure
img1H = hsv2rgb(img1H) ;
imshow(img1H) ;                 % Affichage

img1S = img1 ;
img1S( :, :,1) = 1 ;            % Mise à un de la teinte
img1S( :, :,3) = 1 ;            % Mise à un de la luminosité
subplot(3,1,2) ;                % Positionnement en zone centrale
img1S = hsv2rgb(img1S) ;
imshow(img1S) ;                 % Affichage

img1V = img1 ;
img1V( :, :,1) = 0 ;            % Mise à zero de la teinte
img1V( :, :,2) = 0 ;            % Mise à zéro de la saturation
subplot(3,1,3) ;                % Positionnement en zone inférieure
img1V = hsv2rgb(img1V) ;
imshow(img1V) ;                 % Affichage
```

Résultats





7.4 Rotations d'images

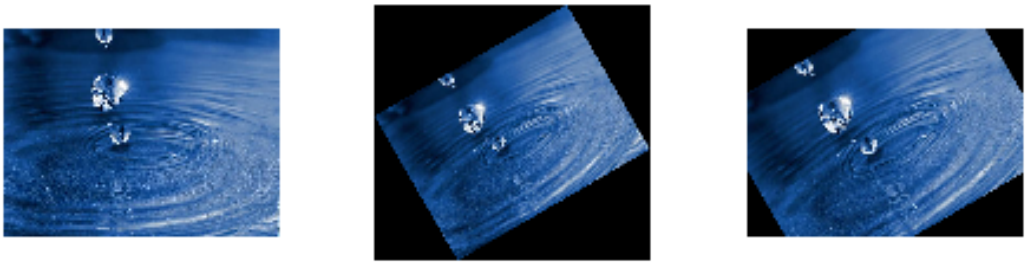
Objectifs

Matlab permet de faire pivoter des images soit en rognant les bords de l'image qui dépasseraient, soit en agrandissant l'image pour ne pas perdre de données.

Code

```
% — Rotation d'images ———————————————————————————— %

img1 = imread('img1.jpg') ;                          % Chargement de l'image dans le workspace

figure ;                                             % Nouvelle fenêtre
subplot(1,3,1) ;                                     % Positionnement en zone de gauche
imshow(img1) ;                                       % Affichage

img1Rot1 = imrotate(img1,30,'bilinear','loose') ;    % Rotation de l'image avec pertes [2]
subplot(3,3,2) ;                                     % Positionnement en zone centrale
imshow(img1Rot1) ;                                   % Affichage

img1Rot2 = imrotate(img1,30,'bilinear','crop') ;     % Rotation de l'image sans perte [2]
subplot(1,3,3) ;                                     % Positionnement en zone droite
imshow(img1Rot2) ;                                   % Affichage

% Argument de imrotate
% - image
% - degrés de rotation (sens trigonométrique)
% - type d'extrapolation,
% - 'loose' ou 'crop'
```

Résultats



7.5 Erosion, dilatation et combinaisons

Objectif

Une érosion consiste à rogner le contour des éléments d'une image à l'aide d'un pattern. Une dilatation à l'inverse, consiste à étendre les bords des éléments d'une image. Enfin, il est possible de combiner les deux pour, par exemple, extraire le contour de formes, etc.

Code

```
% — Erosion, dilatation, combinaison ———————————————————————— %

img6 = imread('img1.jpg') ;                 % Chargement de l'image dans le workspace
figure ;                                    % Nouvelle Fenêtre

pattern1 = ones(10,10) ;                    % Élément servant à éroder/dilater

subplot(2,2,1) ;                            % Positionnement coin supérieur gauche
imshow(img6) ;                              % Affichage

img6Dl = imdilate(img6, pattern1) ;         % Dilatation
subplot(2,2,2) ;                            % Positionnement coin supérieur droit
imshow(img6Dl) ;                            % Affichage

img6Er = imerode(img6, pattern1) ;          % Érosion
subplot(2,2,3) ;                            % Positionnement coin inférieur gauche
imshow(img6Er) ;                            % Affichage

pattern2 = ones(5,5) ;                      % Élément servant à éroder/dilater
imgGrad = imdilate(img6, pattern2)          % Combinaison
  - imerode(img6, pattern2) ;
subplot(2,2,4) ;                            % Positionnement coin inférieur gauche
imshow(imgGrad) ;                           % Affichage
```

Résultats

7.6 Détection de contours

Objectif

Matlab met à disposition des outils permettant d'extraire le contour des éléments d'une image en utilisant des filtre prédéfinis. (*Par exemple : le filtre de Sobel*). Il est de plus possible de régler la sensibilité des filtres en donnant des paramètres aux fonctions.

Code

```
% — Détection de contours ————————————————————————————- %

img2 = imread('img9.jpg') ;                % Chargement de l'image dans le workspace

figure ;                                   % Nouvelle fenêtre
img2Gray = rgb2gray(img2) ;                % Image en niveaux de gris
subplot(1,3,1) ;                           % Positionnement en zone gauche
imshow(img2Gray) ;                         % Affichage

edge1 = edge(img2Gray,'sobel',0.1) ;       % Détetion de bords de type 'sobel'
subplot(1,3,2) ;                           % Positionnement en zone centrale
imshow(edge1) ;                            % Affichage

edge2 = edge(img2Gray,'log') ;             % Détetion de bords de type 'log'
subplot(1,3,3) ;                           % Positionnement en zone centrale
imshow(edge2) ;                            % Affichage
```

Résultats



7.7 Détection améliorée de contours

Objectif

L'objectif est ici d'atteindre un meilleur niveau de détection de contours en combinant un filtre de Sobel avec des érosions et des dilatations.

Code

```
% — Détection de contours améliorée ———————————————————————— %

img9 = imread('img9.jpg') ;                 % Chargement de l'image dans le workspace

figure ;                                    % Nouvelle fenêtre

img9Gray = rgb2gray(img9) ;                 % Image en niveaux de gris
subplot(2,2,1) ;                            % Positionnement en zone gauche
imshow(img9Gray) ;                          % Affichage

edge9 = edge(img9Gray,'log',0.01) ;         % Détection de bords de type 'log'
subplot(2,2,2) ;                            % Positionnement en zone centrale
imshow(edge9) ;                             % Affichage

pattern = ones(1,1)                         % Elément servant à éroder/dilater
edge9Am = imerode(edge9, pattern) ;         % Combinaison
edge9Am = imdilate(edge9Am, pattern)
subplot(2,2,3) ;                            % Positionnement en zone centrale
imshow(edge9Am) ;                           % Affichage

pattern2 = ones(2,2) ;                      % Elément servant à éroder/dilater
edge9Am = imdilate(edge9Am, pattern2)
  - imerode(edge9Am, pattern2) ;            % Combinaison

subplot(2,2,4) ;                            % Positionnement en zone centrale
imshow(edge9Am) ;                           % Affichage
```

Résultats




7.8 Affichage 3 dimensions

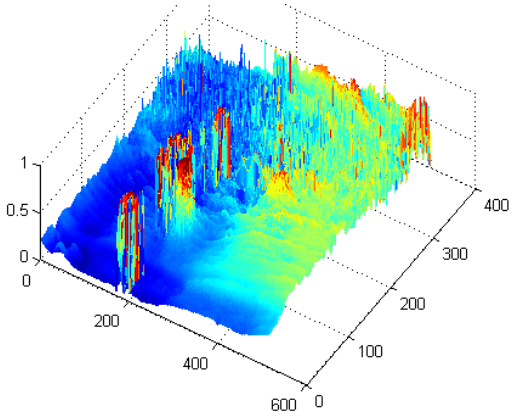
Objectif

Afficher l'image en trois dimensions. La troisième dimension étant le niveau de gris.

Code

```
% — Image 3 Dimensions ————————————————————————————- %

img2 = imread('img1.jpg') ;                    % Chargement de l'image dans le workspace
figure ;                                       % Nouvelle fenêtre

subplot(1,2,1) ;                               % Positionnement en zone de gauche
imshow(img2) ;                                 % Affichage

img2Gray = rgb2gray(img2) ;                    % Transformation de l'image en niveau de gris
img2GrayDouble = im2double(img2Gray) ;         % Image en double précision
subplot(1,2,2) ;                               % Positionnement en zone de gauche
mesh(img2GrayDouble) ;                         % Affichage 3 dimensions
```

Résultats



7.9 Histogramme des composantes

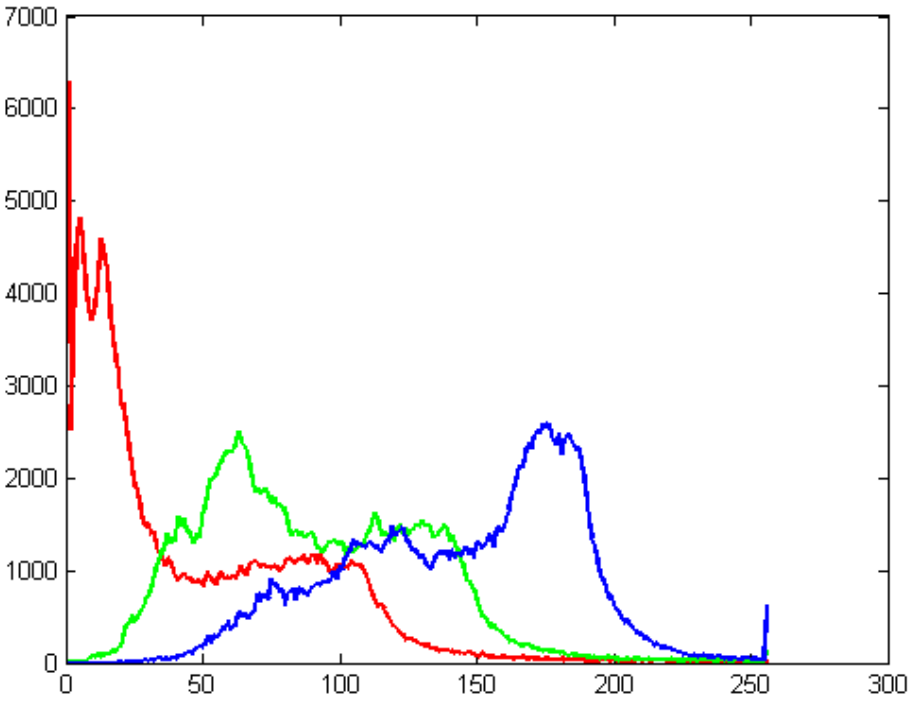
Objectif

L'objectif est ici de représenter un histogramme des composantes RGB de l'image.

Code

```
% — Histogramme des composantes ————————————————————————- %

img1 = imread('img1.jpg') ;                  % Chargement de l'image dans le workspace

histoR = imhist(img1( :, :,1),256) ;         % Histogramme de la composante rouge
histoG = imhist(img1( :, :,2),256) ;         % Histogramme de la composante vert
histoB = imhist(img1( :, :,3),256) ;         % Histogramme de la composante bleu

figure ;                                     % Nouvelle fenêtre

plotR = plot(histoR) ;                       % Affiche l'histogramme rouge
set(plotR,'Color','red','LineWidth',2) ;     % Définition de la couleur et de l'épaisseur

hold on                                      % Maintient le même graphique

plotG = plot(histoG) ;                       % Affiche l'histogramme vert
set(plotG,'Color','green','LineWidth',2) ;   % Définition de la couleur et de l'épaisseur

hold on                                      % Maintient le même graphique

plotB = plot(histoB) ;                       % Affiche l'histogramme bleu
set(plotB,'Color','blue','LineWidth',2) ;    % Définition de la couleur et de l'épaisseur
```

Résultats



7.10 Seuillage

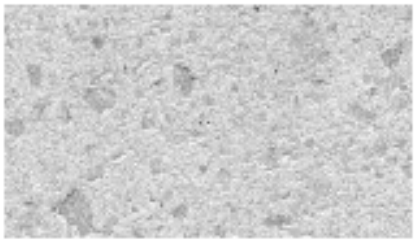
Objectif

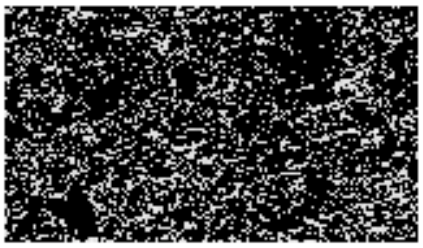
Un seuillage consiste à ne conserver que les pixels dont la valeur est comprise entre un seuil inférieur et un seuil supérieur. Cela permet, par exemple, d'isoler les zones les plus claires d'une image, etc.

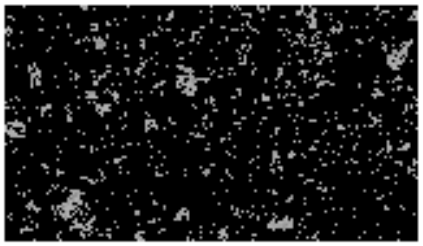
Code

```
% — Seuillage ——————————————————————————————- %

img4 = imread('img4.jpg') ;                          % Chargement de l'image dans le workspace
figure ;                                             % Nouvelle fenêtre

img4R = rgb2gray(img4) ;                             % Transformation de l'image en niveau de gris
img4Double = im2double(img4R) ;                      % Transformation en double précision

subplot(3,1,1) ;                                     % Positionnement en zone de supérieur
imshow(img4Double) ;                                 % Affichage

result1 = (img4Double ¿ 0.85).*img4Double ;          % Seuillage plus de 0.85
subplot(3,1,2) ;                                     % Positionnement en zone de centrale
imshow(result1) ;                                    % Affichage

result2 = (img4Double ¡ 0.72).*img4Double ;          % Seuillage moins de 0.72
subplot(3,1,3) ;                                     % Positionnement en zone inférieur
imshow(result2) ;                                    % Affichage
```

Résultats







7.11 Seuillage amélioré

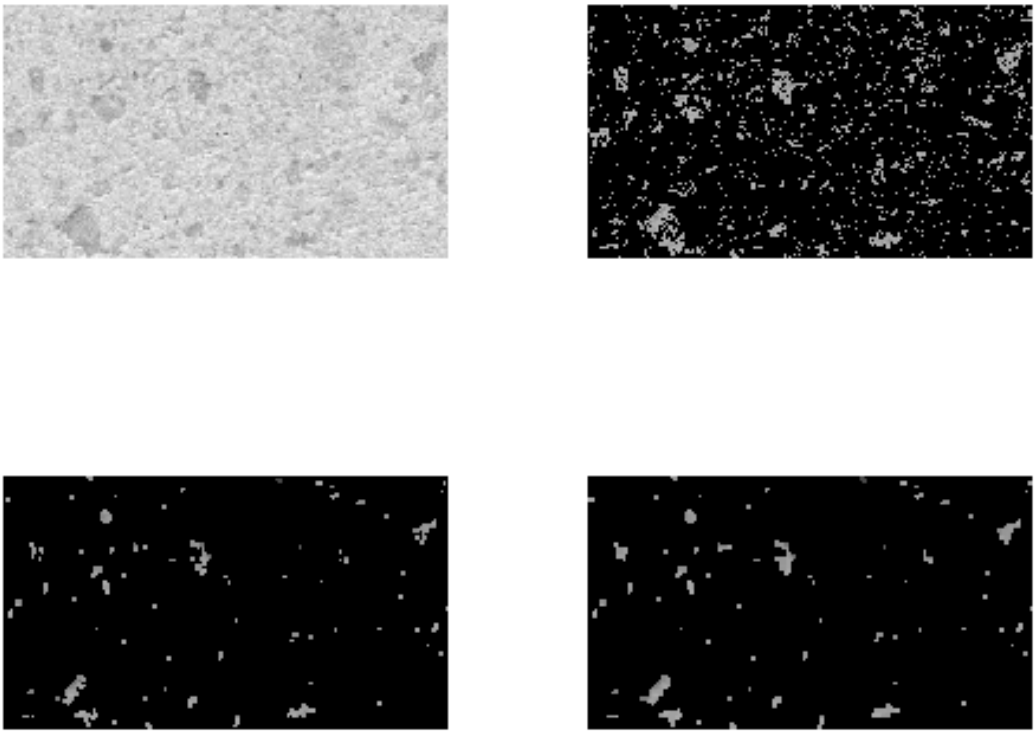
Objectif

Pour améliorer le seuillage, il est possible de combiner un seuillage avec des érosions ou des dilatations de manière, par exemple, à obtenir une image plus net, etc.

Code

```
% — Seuillage amélioré ————————————————————————————- %

img4 = imread('img4.jpg') ;                          % Chargement de l'image dans le workspace
figure ;                                             % Nouvelle fenêtre

img4R = rgb2gray(img4) ;                             % Image en niveau de gris
img4Double = im2double(img4R) ;                      % Transformation en double précision

subplot(2,2,1) ;                                     % Positionnement en zone de supérieur
imshow(img4Double) ;                                 % Affichage

img4Seu = (img4Double ¡ 0.72).*img4Double ;          % Seuillage plus de 0.85
subplot(2,2,2) ;                                     % Positionnement en zone de centrale
imshow(img4Seu) ;                                    % Affichage

 % Suppression des parasites
pattern = ones(3,3)                                  % Élément servant à éroder/dilater
img4SeuAm = imerode(img4Seu, pattern) ;              % Combinaison
img4SeuAm = imdilate(img4SeuAm, pattern) ;

subplot(2,2,3) ;                                     % Positionnement en zone centrale
imshow(img4SeuAm) ;                                  % Affichage

 % Remplissage des trous
pattern2 = ones(5,5) ;
img4SeuAm = imdilate(img4SeuAm, pattern2) ;
img4SeuAm = imerode(img4SeuAm, pattern2) ;           % Combinaison

subplot(2,2,4) ;                                     % Positionnement en zone centrale
imshow(img4SeuAm) ;                                  % Affichage
```

Résultats



7.12 Transformée de Fourier

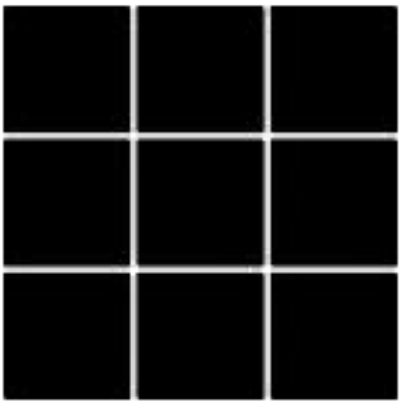
Objectif

Matlab permet d'effectuer la transformée de Fourrier deux dimensions d'image. En voici un exemple.

Code

```
% — Transformée de Fourrier ———————————————— %

img7 = imread('img7.jpg') ;     % Chargement de l'image dans le workspace
figure ;                        % Nouvelle fenêtre

subplot(2,1,1) ;                % Positionnement en zone supérieur
imshow(img7) ;                  % Affichage

img7FFT = fft2(img7) ;          % FFT deux dimensions

subplot(2,1,2) ;                % Positionnement en zone inférieur
imshow(real(img7FFT)) ;         % Affichage
```

Résultats



Références

- Mathworks :
  http ://www.mathworks.com/products/

- Université catholique de Louvain :
  http ://www.elec.ucl.ac.be/enseignement/ELEC2000/index.php ?page=tutmatlab
  http ://www.stat.ucl.ac.be/cours/fsat2/app-ape/MATLABSTAT.pdf

- Université de Genève :
  http ://perso.unige.ch/∼manuel/Matlab/MatLab1.pdf

- Université de Paris (UPMC) Département d'analyse numérique :
  http ://www.ann.jussieu.fr/∼postel/matlab/node26.html

- Université de cergy-Pontoise :
  http ://www.u-cergy.fr/sir/article.php3 ?id_article=67

- Université Carlos III de Madrid :
  http ://halweb.uc3m.es/esp/Personal/personas/amalonso/esp/Matlab_Tutorial_PartII_2005.pdf

- http ://pagesperso-orange.fr/jerome.landre/docs/matlab_ti.pdf