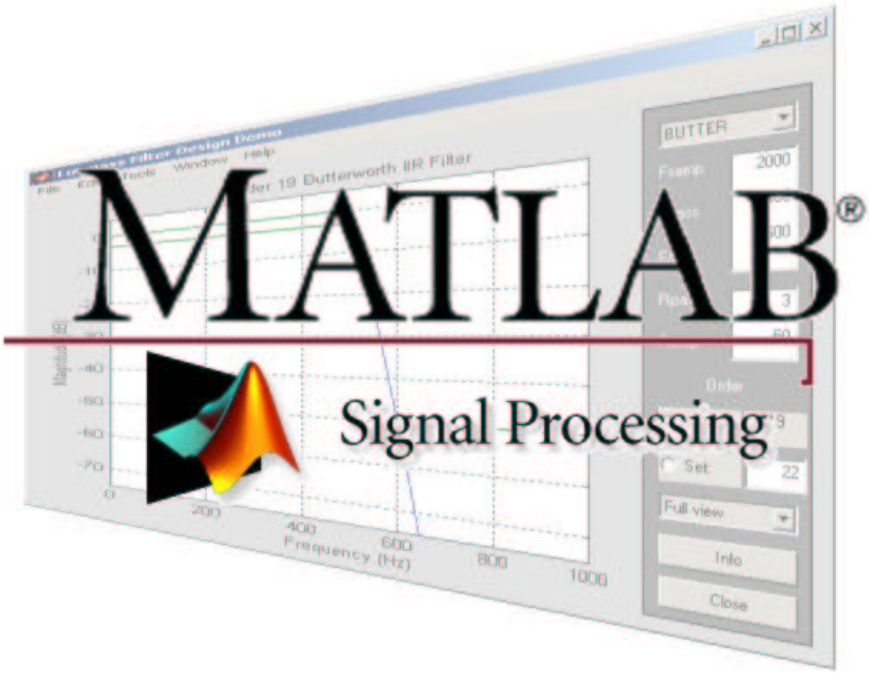
Université de Metz

T.P. de Traitement du Signal

I.U.P. 3 GSI Option TI

Année Universitaire 2004/2005



O. Habert, P. Arnould, Y. Morère

Cette page est laissée blanche intentionnellement

Table des matières

Cette page est laissée blanche intentionnellement

TP 1 : Introduction à Matlab

(2 séances)

1.1 Objectif du TP

Le but de ce TP est de vous familiariser avec le logiciel Matlab qui sera utilisé pour tous les TP de traitement de signal. Matlab (Matrix Laboratory) est un environnement de calcul permettant des calculs numériques et des représentations graphiques. Dans ce TP vous trouverez en première partie un bref descriptif du logiciel et en deuxième partie le TP proprement dit qui se compose de plusieurs exercices.

1.2 Bref descriptif de Matlab

Pour lancer Matlab, cliquez sur l'icône Matlab. Au lancement , apparaît la fenêtre suivante :

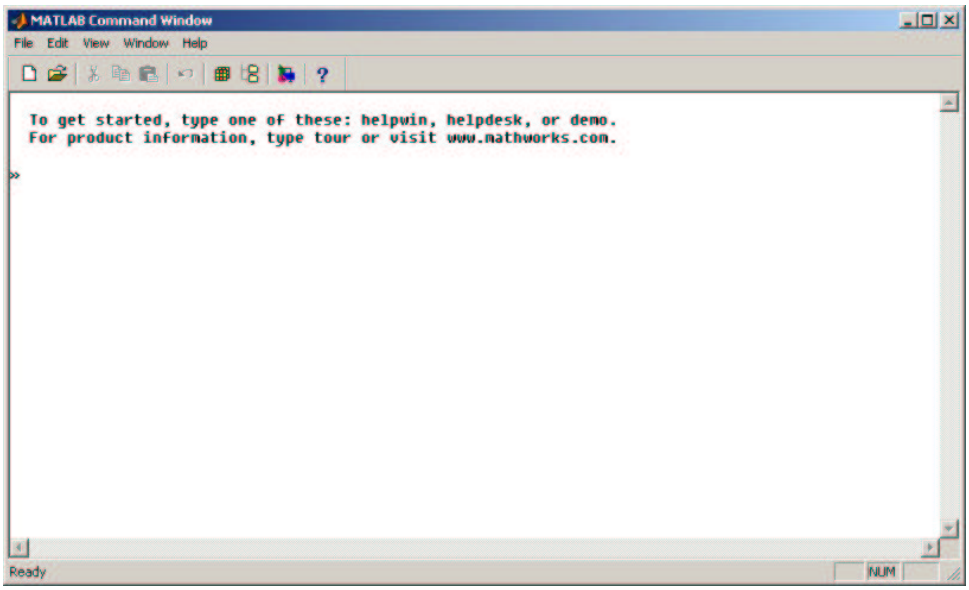


FIG. 1.1 – Fenêtre de commande Matlab

– Le menu `File` : manipulation des fichiers de Matlab,
– le menu `Edit` : utilisation du presse papier de Windows,
– le menu `Options` : positionnement des options d'utilisation,

– Le menu `Windows` permet d'utiliser les différentes fenêtres de Matlab : interpréteur et figures, Le menu `Help` fournit une aide interactive sous la forme d'un fichier Windows en hypertexte.

Matlab est un interpréteur de commandes dont l'élément de base est la matrice. Sous Matlab, la syntaxe générale est de la forme :

```
variable = expression ou
expression
```

Par exemple la matrice suivante $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$ sera rentrée sous Matlab de la manière suivante :

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

ou en remplaçant les ; par des sauts de lignes

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

Les éléments des matrices peuvent être une expression Matlab quelconque :

```
>>x=[-1.3   sqrt(3)   (1+2+3)*4/5]
>>x=
    -1.3 1.7321 4.8000
```

Les éléments des matrices peuvent être accédés en mettant leur indice à l'intérieur de deux parenthèses `()` :

```
>>x(1)
ans=
    -1.3000
```

On peut aussi construire des matrices en utilisant des matrices plus petites :

```
>>x(5)=abs(x(1))
x=
   -1.3 1.7321 4.8000 0 1.3000
>>r=[10 11 12];
>>A=[A;r]
A=
    1  2  3
4  5  6
7  8  9
10 11 12
```

On remarquera que le signe ; permet de ne pas afficher le résultat de la commande.
La commande `who` permet de visualiser les variables et whos permet en plus de préciser leur nature :

```
>>who
     Your variables are:
     A         ans       r         x
>>whos
       Name        Size       Elements    Bytes    Density   Complex
         A        4 by 3         12         96        Full      No
        ans       1 by 1          1          8        Full      No
         r        1 by 3          3         24        Full      No
         x        1 by 5          5         40        Full      No
Grand total is 21 elements using 168 bytes
```

Pour créer un nombre complexe, on utilisera i ou j ($i2 = -1$ ou $j2 = -1$) :

```
>>z = 2+3i
>>z2 = 2-2j
```

Les opérateurs sont :
– `+` addition,
– `-` soustraction,
– `*` multiplication,
– `/` division à droite,
– `\` division à gauche,
– `'` transposée,
– `<opérateur>` opération membre à membre.
L'opérateur `*` permet de réaliser directement des multiplications de matrices.
L'opérateur `\` permet de résoudre l'équation `A*X=B` : la solution est `X=A\B`. Par exemple, en utilisant la notation matricielle cet opérateur permet de résoudre les systèmes d'équations :

$$\left\{ \begin{array}{l} x+y=2 \\ 2x+3y=5 \end{array} \right. \Leftrightarrow \left(\begin{array}{cc} 1 & 1 \\ 2 & 3 \end{array} \right) \left(\begin{array}{c} x \\ y \end{array} \right) = \left(\begin{array}{c} 2 \\ 5 \end{array} \right) \Leftrightarrow A \times X = B \Rightarrow X = A \backslash B$$

Le symbole : permet sous Matlab de créer des vecteurs, par exemple `x=1:5` crée un vecteur `x` contenant les chiffres de 1 à 5 avec un incrément de 1. Pour rentrer un incrément différent de 1 (ici 0.8), on tape :

```
>>x=1:0.8:5
```

Le symbole : permet aussi de sélectionner des indices dans les matrices.
Sachant que A est une matrice `10x10` :
– `A(1:5, 3)` sélectionnera le troisième élément des lignes 1, 2, 3, 4 et 5,
– `A(:,3)` sélectionnera la troisième colonne,
– `A(:,:)` sélectionnera la matrice entière.
Matlab permet aussi de réaliser des graphiques en 2 ou 3 dimensions. Voici un exemple en deux dimensions :

```
>>t=0:pi/10:2*pi;
>>y=sin(t);
>>plot(t,y)
>>title('Mon premier graphique sous Matlab')
>>xlabel('t en secondes')
>>ylabel('V en volts')
```

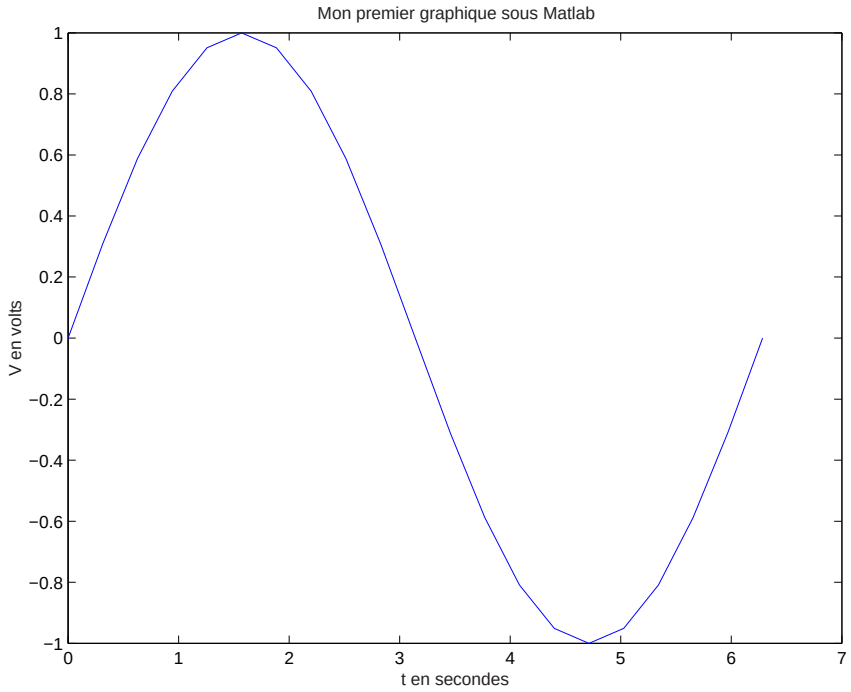


FIG. 1.2 – Premier graphique avec Matlab

Matlab permet d'utiliser des fichiers de commandes. Un fichier de commandes est fichier rassemblant des commandes Matlab.
Ce fichier est exécuté lorsqu'on tape le nom du fichier sous l'interpréteur. Le nom du fichier doit obligatoirement avoir une terminaison ".m". Voici un fichier de commandes fibo.m permettant de calculer la suite de Fibonacci.

```
% exemple de fichier de commandes, calcul de la suite de Fibonacci
% u(n+2)=u(n+1)+u(n)
f=[1 1];
i=1;
while f(i) + f(i+1) < 1000
    f(i+2)=f(i+1)+f(i);
    i=i+1;
end
plot(f)
```

On peut aussi écrire des fonctions sous Matlab. Une fonction sera un fichier Matlab qui commence par function (et non pas par fonction) et qui aura des arguments.
Par exemple voici la fonction mean qui calcule la moyenne d'un vecteur :

```
>>a=1:99;
>>y=mean(a);
```

```
function y = mean(x)
%MEAN Average or mean value.
% For vectors,  MEAN(X)  is the mean value of the elements in X.
% For matrices,  MEAN(X) is a row vector containing the mean value
```

```
% of each column.
%
% See also MEDIAN, STD, MIN, MAX.
% Copyright (c) 1984-94 by The MathWorks, Inc.

[m,n] = size(x);
if m == 1
    m = n;
end
y = sum(x) / m;
```

1.3 Excercices

Il est a noter qu'une aide en ligne est disponible à tout moment au niveau de l'interpréteur de commande. Par exemple, pour obtenir de l'aide sur la fonction `plot`, taper `help plot`. Veuillez écrire les fichiers de commandes pour les exercices suivants. A la fin de la séance vous rendrez un compte rendu ainsi que les listings de ces fichiers de commandes (qui devront être enregistrés sous votre répertoire).

1. Soit `x=[1 2 3]` et `y=[4 5 6]`. Que représente `x*y'` et `x'*y`?
2. Calculer le déterminant de $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ de deux manières (fonctions intégrée et calcul direct).
3. Afficher les courbes suivantes :
   - $x \in [0,1],\ f_1(x) = \cos(\tan(\pi x))$,
   - $x \in [-10,10],\ f_2(x) = \frac{\sin(x)}{x}$,
   - $x \in [-100,100],\ f_3(x) = x^5 + 2x^2 + x + 1$,
   - $x \in [-2,2],\ f_4(x) = e^{-t^2} \sin t$.
4. Résoudre le système suivant :

$$\begin{cases} x + y + z = 0 \\ 2x + y + z = 10 \\ 2x - y + z = 3 \end{cases}$$

5. Approximation d'une fonction par un polynôme.
   Soit $f(x)$, définie pour $x \in [a,b]$ (intervalle fini). Approximer $f(x)$ par un polynôme du premier degré revient à écrire :

$$y = p(x) \text{ où } p(x) = p_1(x) + p_0$$

   C'est-à-dire sous Matlab, il faut résoudre le système composé des n éléments de l'intervalle de définition de x :

$$\begin{cases} p_1(1) + p_0 = y(1) \\ \ldots \\ p_1(n) + p_0 = y(n) \end{cases} \text{ où } p_1, p_0 \text{ sont les inconnues.}$$

Application :

Approximer par un polynôme du 1^{er}, 2^{eme} et 3^{eme} degrés, la fonction $f(x) = \sin(x)$ sur l'intervalle $[0, \pi/2]$. À chaque fois répondre aux questions 5a, 5b et 5c.

(a) Ecrire les variables A, P et B permettant de résoudre le système.

(b) Résoudre le système

(c) Afficher le résultat

(d) A l'aide de la fonction subplot afficher les résultats sur une même fenêtre les trois courbes différentes.

1.4 Les signaux numériques

Nous allons voir les fonctions qui permettent de générer les signaux numériques usuels à l'aide de la fonction `stem`.

1.4.1 L'impulsion unité

Afin de générer l'impulsion unité on peut écrire le programme suivant :

```
%impulsion unité
t=-10:10;
x=[zeros(1,10),1,zeros(1,10)];
stem(t,x);
axis([-10 10 -0.5 1.5]);
title('Impulsion unité');
xlabel('n');
ylabel('Amplitude');
```

Ecrivez et testez le programme précédent. Expliquez à quoi servent les différentes fonctions.

1.4.2 L'échelon unité

Pour la cas de l'échelon unité, on se contentera d'un nombre fini d'échantillon. On peut écrire le programme suivant :

```
%echelon unité
t=-10:10;
x=[zeros(1,10),ones(1,11)];
stem(t,x);
axis([-10 10 -0.5 1.5]);
title('Echelon unité');
xlabel('n');
ylabel('Amplitude');
```

Ecrivez et testez le programme précédent.

1.4.3 Sinus et exponentielle décroissante

On recommence avec la fonction suivante $\sin(0,35 \times n)$. On écrira le programme suivant :

```
%sinus
t=-10:10;
x=sin(0.35*t);
stem(t,x);
axis([-10 10 -1.5 1.5]);
title('sinus');
xlabel('n');
ylabel('Amplitude');
```

Ensuite avec une exponentielle décroissante $e^{0,2\times n} \times u[n]$ avec u l'échelon unité. On a le programme suivant :

```
%exponentielle
t=-10:10;
u=[zeros(1,10),ones(1,11)];
x=exp(-0.2*t).*u;
stem(t,x);
axis([-10 10 -1.5 1.5]);
title('Exponentielle retardée');
xlabel('n');
ylabel('Amplitude');
```

On utilise ici l'opérateur de multiplication termes à termes `.*` qui permet de permet d'effectuer la multiplication terme à terme de deux matrices ($(A. * B)_{i,j} = A_{i,j} \times B_{i,j}$).

1.4.4 Scripts Matlab

Les morceaux de programmes ci dessus peuvent être recopiés dans des fichiers texte à l'aide de l'éditeur de matlab et enregistrés sous des noms représentatifs tels que `unit.m`, `echel.m`, `sinus.m` et `expo.m`. Il s'agit alors de fichiers de commandes scripts matlab qui penvent être lancés directement à partir de la console matlab. Les commandes contenues dans le fichier sont lancées comme si elles étaient tapées dans la console.

1.5 Opérations sur les signaux

1.5.1 Décalage et retournement temporelle

Les signaux numériques sont souvent exprimés comme des combinaisons d'autres signaux élémentaires décalés ou retournés dans le temps.Ainsi le signal $s[n-N]$ est égal au signal $a[n]$ décalé de N échantillons dans le temps (N entier). On effectue donc une translation de N échantillons vers la droite, si N est positif, et vers la gauche si N est négatif. On va utiliser l'échelon unité pour illustrer ce concept. Le signal $a[-n]$ est la version retournée dans le temps de $a[n]$. Cela signifie que le signal est retourné par rapport au point $n = 0$. On aura par exemple :

```
t=-10:10;
delta=[zeros(1,10),ones(1,11)];
subplot(3,1,1);
stem(t,delta);
axis([-10 10 -1.5 1.5]);
title('\delta[n]');
xlabel('n');
ylabel('Amplitude');
subplot(3,1,2);
deltam2=[zeros(1,2),delta(1:length(delta)-2)];
stem(t,deltam2);
axis([-10 10 -1.5 1.5]);
title('\delta[n-2]');
xlabel('n');
ylabel('Amplitude');
subplot(3,1,3);
deltap2=[delta(3:length(delta)),zeros(1,2)];
stem(t,deltap2);
axis([-10 10 -1.5 1.5]);
title('\delta[n+2]');
xlabel('n');
ylabel('Amplitude');
```

Ecrivez et testez le programme précédent. Expliquez à quoi servent les différentes fonctions.
Puis le programme suivant :

```
t=-10:10;
u=[zeros(1,10),ones(1,11)];
x=exp(-0.2*t).*u;
subplot(2,1,1);
stem(t,x);
axis([-10 10 -1.5 1.5]);
title('x[n]');
xlabel('n');
ylabel('Amplitude');
subplot(2,1,2);
x1=x(length(x):-1:1);
stem(t,x1);
axis([-10 10 -1.5 1.5]);
title('x[-n]');
xlabel('n');
ylabel('Amplitude');
```

Ecrivez et testez le programme précédent.

1.5.2 Fonctions matlab

Il est possible de mettre au point des programmes permettant de réaliser de manière automatique et général un décalage, un retournement sur un signal arbitraire d'entrée.

Ce sont des fonctions matlab, repérées par un identificateur lié au nom de fichier qui les contient. Ainsi le corps de la fonction `toto` doit se trouver dans le fichier `toto.m`. De plus ce fichier doit être accessible par matlab au moment de son appel (Notion de chemin d'accès `PATH`).
Par exemple la fonction `sigshift` :

```
function [xs,ts]=sigshift(x,t,N)
%shifting a signal
%inputs :
%       x,t     input signal amplitude and instants
%       N       shift value
%outputs :
%       xs,ts   shifted signal amplitude and instants
xs=x;
ts=t-N;
```

Ecrivez et testez le programme précédent sur un exemple (l'echelon unité est une bon point de départ et noubliez pas d'afficher le résultat). Expliquez à quoi servent les différentes mots-clés.
Réalisez les mêmes opérations avec les fonctions suivantes :

```
function [xr,tr]=sigrev(x,t)
%Time reverting a signal
%inputs :
%       x,t     input signal amplitude and instants
%outputs :
%       xr,tr   time reversed signal amplitude and instants
lx=length(x);
tr=-t(lx:-1:1);
xr=x(lx:-1:1;

function [x,tx]=impseq(n0,n1,n2)
%Time shifted unit impulse
%inputs :
%       n0     shifting value
%       n1     timing begin value
%       n2     timing end value
%outputs :
%       x,tx   shifted unit impulse
tx=n1:n2;
x=[(tx-n0)==0];

function [x,tx]=stepseq(n0,n1,n2)
%Time shifted unit step
%inputs :
%       n0     shifting value
%       n1     timing begin value
%       n2     timing end value
%outputs :
%       x,tx   shifted unit step
```

```
tx=n1:n2;
x=[(tx-n0)>=0];
```

1.5.3 Addition, soustraction, multiplication et division sur les signaux

On peut maintenant aborder les opérations fondamentales sur les signaux. La fonction suivante permet d'ajouter deux signaux :

```
function [x,t]=sigadd(x1,t1,x2,t2)
%Adding two signals
%inputs :
%       x1,t1     first input signal amplitudes and instants
%       x2,t2     second input signal amplitudes and instants
%outputs :
%       x,t       sum signal amplitudes and instants
first=min(t1(1),t2(1));
l1=length(t1);
l2=length(t2);
last=max(t1(l1),t2(l2));
t=first:last;
l=length(t);
%search number of preamble ans postamble zeros
%needed to extend x1 and x2
preamb1=t1(1)-first;
preamb2=t2(1)-first;
postamb1=last-t1(l1);
postamb2=last-t2(l2);
%x1e=[zeros(1,preamb1),x1,zeros(1,postamb1)];
if preamb1==0
    x1e=[];
else
    x1e=[zeros(1,preamb1)];
end
%x2e=[zeros(1,preamb2),x2,zeros(1,postamb2)];
if preamb2==0
    x2e=[];
else
    x2e=[zeros(1,preamb2)];
end
x1e=[x1e,x1];
x2e=[x2e,x2];
if postamb1~=0
    x1e=[x1e,zeros(1,postamb1)];
end
if postamb2~=0
    x2e=[x2e,zeros(1,postamb2)];
end
```

```
x=x1e+x2e;
```

Ecrivez et testez la fonction précédente sur un exemple (ajouter un echelon et une sinusoïde par exemple). Expliquez à quoi servent les différentes mots-clés.
Réaliser simplement maintenant les fonctions `sigdiff`, `sigmul` et `sigdiv`.

1.5.4 Autres opérations

On peut encore envisager de calculer la somme ou le produit de tous les échantillons d'un signal ou encore la puissance d'un signal :

$$P_x = \frac{1}{N} \sum_{n=0}^{N-1} |x[n]|^2$$

En guise d'exercice réaliser les fonctions `sumsignal`, `prodsignal` et `puissancesignal`. Puis testez les sur un exemple.

1.6 Spectre des signaux

La représentation des signaux dans le domaine temporelle utilise la notion de spectre. Matlab fournit la fonction `fft` pour calculer la tranformée de Fourier complexe d'un vecteur. Le vecteur signal etant de dimension finie, c'est la transformée discrète qui est calculée. Si N est la longueur su signal, le spectre sera un vecteur complexe de même longueur qui pourra être représenté en coordonnées cartésiennes (partie réelle et imaginaire fonction`real` et `imag`), ou en coordonnées polaires (module et phase, fonction matlab `abs` et `angle` ou `unwrap`).
Prenons la cas d'une sinusoîde amortie. Les fréquences vont être graduées en Hz en supposant une fréquence d'échantillonnage f_e de 1Kz – les fréquences les fréquences supérieures à 500Hz correspondent au fréquence négatives du spectre $f_- = f - f_e$.

```
%signal sinusoïdal amorti
t=-10:1000;
u=[t>0];
s=sin(0.35*t);
x=exp(-0.2*t).*s.*u;
subplot(3,1,1);
stem(t,x);
axis([-10 30 -0.6 0.6]);
title('Sinusoïde amortie');
xlabel('n');
ylabel('Amplitude');
%spectre
X=fft(x);
rX=real(X);
iX=imag(X);
N=length(t);
f=(0:N-1)*1000/N; %réalisation de la plage de fréquence
subplot(3,2,3)
```

```
plot(f,rX);
title('Real(X)');
xlabel('f(Hz)');
ylabel('Amplitude');
grid;
subplot(3,2,4);
plot(f,iX);
title('Imag(X)');
xlabel('f(Hz)');
ylabel('Amplitude');
grid;
rho=abs(X);
theta=angle(X); %en radians
theta1=180*theta/pi; %en degres
subplot(3,2,5);
plot(f,rho);
title('Mag(X)');
xlabel('f(Hz)');
ylabel('Amplitude');
grid;
subplot(3,2,6);
plot(f,theta1);
title('Phase(X)');
xlabel('f(Hz)');
ylabel('deg.');
grid;

figure; %nouvelle figure
rho1=20*log10(rho);
theta2=180*unwrap(theta)/pi;
subplot(1,2,1);
plot(f,rho1);
title('Mag(X)');
xlabel('f(Hz)');
ylabel('dB.');
grid;
subplot(1,2,2);
plot(f,theta2);
title('Phase(X)');
xlabel('f(Hz)');
ylabel('deg.');
grid;
```

La commande `subplot` permet de réaliser la mise en page des figures. Il est couratn d'utiliser le module en dB et pour la phase de recourir à un algorithme de «déballage» pour la rendre continue.

Ecrivez et testez la fonction. Expliquez à quoi servent les différentes mots-clés.

1.7 Exemples de signaux

1.7.1 Exemple 1

On considère le signal :

```
x=[1 2 3 4 5 6 7 6 5 4 3 2 1];
t=-2:10;
```

Générer et tracer les séquences suivantes à l'aide des fonctions précédemment définies :

$$x_1[n] = 2x[n-5] - 3x[n+4]$$

$$x_2[n] = x[3-n] - x[n]x[n-2]$$

ainsi que leurs spectres.

1.7.2 Exemple 2

On considère le signal complexe :

```
t=-10:10;
alpha=-0.1+j*0.3;
x=exp(alpha*t);
```

Générer ce signal et en tracer les parties réelles et imaginiare ainsi que le module et la phase dans quatres vignettes indépendantes de la même figure. Représenter le spectre du signal.

Cette page est laissée blanche intentionnellement

TP 2 : Variables Aléatoires Continues et Discrètes

(2 séances)

2.1 Objectif du TP

Dans ce TP nous étudierons :
- Les propriétés des variables aléatoires discrètes et continues;
- Les variables aléatoires uniformes et Gaussiennes en terme de leurs densité de probabilité et fonction de répartition cumulée;
- Les moyennes, variances, écart types statistiques.

2.2 Préparation

1. Considérons un dé à 4 faces numérotées de 1 à 4.
   (a) Déterminer l'espace probabiliste quand le dé est lancé une seule fois
   (b) Dans l'expérience qui suit le dé est lancé 2000 fois. On observe les échantillons suivants :

Numéro de la face	1	2	3	4
Nombre d'occurences	112	784	865	239

TAB. 2.1 – Répartition des lancés de dé à 4 faces

   Observons une variable aléatoire X représentant le numéro de la face du dé quand celui-ci est lancé. Utilisez la définition de la fréquence relative en probabilité pour estimer $P(X = k),\ k = 1, ..., 4$.
   (c) En utilisant les résultats précédent, calculez et tracez la densité de probabilité et la fonction de répartition cumulée de X.
   (d) Déterminez $P(1 < X < 5)$ et $P(X \geq 5)$.
   (e) Déterminez $P(X = k), k = 1, ..., 4$, si le dé est non pipé.
2. Soit $U \sim U(a, b)$, une variable aléatoire uniformément distribué sur l'intervalle $[a, b]$
   (a) Déterminez et tracez la densité de probabilité de U.
   (b) Déterminez et tracez la fonction de répartition cumulée de U.
   (c) Déterminez $E[U]$, $E[U^2]$, et σ^2 en fonction de a et b.

(d) Pour $a = 2$ et $b = 6$, évaluer $P(-2 < U < 4)$ et $P(U = 5)$.

3. Considérons une variable aléatoire normale $X \sim N(\mu, \sigma^2)$. Soit $\Phi(z)$ la fonction de répartition cumulée de la variable aléatoire normale $Z \sim N(0, 1)$, dont les valeurs sont données sous forme de tableau.
   (a) Exprimez $P(a < X < b)$ et $P(X > b)$ en fonction de Φ,a,b,μ.
   (b) Pour $\mu = 4$, $\sigma^2 = 4/3$, déterminer $P(3 \leq X \leq 5)$ et $P(X \geq 5)$.

2.3 Pratique

Remarque :
– Lancez `start2` avant de démarrer la pratique.
– une fonction `help<nom de la fonction>` est disponible.

2.3.1 Variables Aléatoires Discrètes

1. En utilisant la fonction `de`, générez un échantillon aléatoire représentant les résultats de l'expérience quand un dé à 4 faces non-pipé est lancé 2000 fois.

   ```
   >> x = de(2000,4,'nonpipe');
   ```

2. Calculez et affichez la densité de probabilité (*dp*) et la fonction de répartition cumulée (*fr*) de la séquence x précédente.

   ```
   >>subplot(121), dp(x)
   >>subplot(122), fr(x)
   ```

   Les échantillons d'événements représentant les résultats d'expériences où le dé est lancé une fois, sont de la forme valeur de la face $= k, k = 1, ..., 4$. Observer la relation entre la *dp* et la *fr* pour différents cas.
   Question : Pourquoi les probabilités des échantillons précédents ne sont pas égales? Pouvez t-on s'attendre à un tel résultat? Quel élément donné en 2.3.1.1, peut justifier ce résultat.

3. Utilisez la fonction de qui génère une séquence représentant le résultat d'une expérience où un dé pipé est lancé 2000 fois.

   ```
   >> y = de(2000,4,'pipe');
   ```

   Effacez la fenêtre graphique et affichez la `fdp` de la séquence aléatoire y :

   ```
   >> clf
   >> dp(y)
   ```

   Tracez la *dp*. Calculez et tracez la *fr*. Indiquez tout les points caractéristiques à la *fr*. Vous pouvez vérifier votre réponse en utilisant la fonction *fr* de MATLAB.
   Question : La dp étant donnée pour un échantillon de 2000 expériences, estimez le nombre d'occurrences (résultats) pour chaque face.

2.3.2 Variables Aléatoires Continues

Variable aléatoire uniforme

1. En utilisant les fonctions `unif_dp` et `unif_fr` pour la variable aléatoire $U(2, 6)$, esquissez la *dp* et la *fr*.

```
>> subplot(121), unif_dp(2,6), axis([0, 8, -0.2, 1.2]);
>> subplot(122), unif_fr(2,6), axis([0, 8, -0.2, 1.2]);
```

2. Si $U \sim u(2,6)$, déterminez les probabilités suivantes en utilisant uniquement la *dp* ou la *fr* $P(0 < U \leq 3)$, $P(3 < U \leq 5)$, $P(U = 3)$.
   Question : Pourquoi $P(U = 3)$ est différente de $P(X = 3)$ en 2.3.1.3.
3. Jouons avec les statistiques. Générez 500 échantillons d'une distribution $U(2,6)$.

   ```
   >> u = uniform(2,6,500);
   ```

   Calculer la moyenne et la variance de la séquence aléatoire U.

   ```
   >> moy_u = mean(u) , var_u = var(u)
   ```

   Comparez ces résultats avec les réponses de la préparation en 2. Commentez toute différence.
   Pouvez-vous prédire les algorithmes des fonctions `mean` et `var` de MATLAB. Afficher leur contenu en utilisant la commande type de MATLAB.
   Utilisez `moy_u` et `var_u` pour déterminer $E[U^2]$. Vérifier votre résultat avec la fonction `moycar`.

Variable Aléatoire (normale) Gaussienne

1. En utilisant les fonctions `gaus_dp` et `gaus_fr`, afficher la *dp* et la *fr* de la variable aléatoire G. $G \sim N(\mu, \sigma^2)$ avec $\mu =$ `moy_u` et $\sigma^2 =$ `var_u`, voir point 2.3.2.3.Vérifier la *dp* et la *fr*.

   ```
   >> clg, subplot(121), gaus_dp(moy_u,var_u)
   >> subplot(122), gaus_fr(moy_u,var_u)
   ```

   Indiquez les valeurs sur l'axe horizontal pour la *dp* maximum et pour la *fr* égale à 0.5. Comparez ces valeurs avec la valeur moyenne de la distribution de Gauss.
2. Déterminez les probabilités suivantes : $P(0 < G \leq 3)$, $P(3 < G \leq 5)$, $P(G \geq 5)$. Comparez ces résultats avec ceux du point 2.3.2.2 . La distribution de Gauss utilisée pour générer ces résultats et la distribution uniforme au point 2.3.2.2 ont les mêmes moyenne et variance, pourtant les résultats précédents sont différents. Pouvez-vous l'expliquer?
3. Prenons $X \sim N(\mu, ^2)$. Supposons la valeur moyenne de X fixé $\mu = 1$ et $\sigma^2\{0.5, 1, 2, 5, 10\}$. Pour observer les effets de la variation de σ^2 sur la *dp* de la variable aléatoire Gaussienne, tapez la séquence suivante:

   ```
   >> clf
   >> m = 1; gaus_dp(m,0.5)
   >> axis( [-10 10 0 0.6]), hold on
   >> gaus_dp(m,1)
   :
   :
   >> gaus_dp(m,10)
   ```

   Question : Considérons l'événement $A = \{0 < X < 2\}$ où $X \sim N(1, \sigma^2)$ avec $\sigma^2\{0.5, 1, 2, 5, 10\}$. Déterminez la valeur de σ^2 pour laquelle $P(A)$ est maximum.
   Maintenant considérons les changements sur la *fr* de la variable aléatoire Gaussienne.

```
>> clf
>> m = 1; gaus_fr(m,10)
>> axis( [-10 10 0 1]), hold on
>> gaus_fr(m,5)
:
:
>> gaus_fr(m,0.5)
```

Pouvez-vous prédire la *fr* quand σ^2 devient très petit? Faire

```
>> gaus_fr(m,0.00001)
```

Que doit-on faire pour avoir une distribution de probabilité avec une très petite variance? La *dp* correspondante peut aider à illustrer ce point:

```
>> clf
>> gaus_dp(m,0.00001)
>> axis( [0 2 0 200])
```

Où dans ce TP avez-vous déjà observé une *dp* similaire? Si dans le dernier point nous avions une variable aléatoire uniformément distribué avec pour moyenne μ et pour variance 0.00001 à la place de la variable aléatoire gaussienne, la *dp* serait-elle différente?

4. Maintenant fixez la variance de la distribution de Gauss à σ^2 et changez la valeur moyenne μ tel que $\mu\{-4, -1, 2, 5\}$. Quel est l'effet de la variation de μ ? Prenons $X(\mu, \sigma^2)$ représentant une variable aléatoire tel que $X(\mu, \sigma^2) \sim N(\mu, \sigma^2)$. Comparez $P(-5 < X(-4, 1) < -3)$ et $P(4 < X(5, 1) < 6)$. Vous pouvez faire d'autres observations sur les effets de la variation de μ sur la *fr*. Tout d'abord effacer la fenêtre graphique et initialiser les axes $[-8, 8, 0, 1]$.

2.3.3 Moyenne, Variance et Puissance.

1. Tapez les séquences aléatoires suivantes avec différentes valeurs de moyenne :

```
>> x = gauss(-5,1,100);
>> y = gauss(0,1,100);
>> z = gauss(5,1,100);
>> clf
>> axis( [1 100 -10 10] ), grid on, hold on
>> plot(y)
>> plot(z)
```

En utilisant les terminologies techniques, vous pouvez savoir que la valeur moyenne change le niveau de la forme de l'onde.

2. Générez les séquences aléatoires de distribution Gaussienne avec différentes valeurs de variance.

```
>> a = gauss(0, 4, 100);
>> b = gauss(0, 1, 100);
>> c = gauss(0, 0.5, 100);
>> d = gauss(0, 0.01, 100);
>> clf
>> subplot(221), plot(a), axis( [1 100 -10 10] )
```

```
>> subplot(222), plot(b), axis( [1 100 -10 10] )
>> subplot(223), plot(c), axis( [1 100 -10 10] )
>> subplot(224), plot(d), axis( [1 100 -10 10] )
```

En utilisant les fonctions `mean` et `var` de MATLAB, déterminez la moyenne et la variance de chaque séquence et remplissez le tableau suivant avec vos réponses.

Séquence	Moyenne	Variance	Moyenne au Carré
a			
b			
c			
d			

TAB. 2.2 – Tableau de résultat

Déterminez la moyenne au carré de chaque forme d'onde en utilisant respectivement la moyenne et la variance. Vérifier vos résultats en utilisant la fonction `moycar`.

2.3.4 Jeu de fléchettes

Pour illustrer la signification de la moyenne et de la variance, utilisez la fonction flèche, qui simule une expérience où une fléchette est lancée dans une cible. L'utilisateur peut spécifier la moyenne, la variance de x et y qui déterminent le point d'impact sur la cible. Un bon point de départ de l'expérimentation est:

```
>> moy_x = 0.2; moy_y = 0.2;
>> var_x = 0.1; var_y = 0.1;
>> nombre_fleches = 20;
>> clf
>> fleche( [moy_x moy_y], [var_x var_y], nombre_fleches )
```

Si vous voulez avoir plus d'information à propos de cette simulation, utilisez l'aide en ligne en tapant help flèche.
Reprenez l'expérience en faisant varier les valeurs des moyennes et des variances. Il est intéressant de voir les résultats de la simulation pour une ou deux variances proches de zéro. Essayez de corréler vos observations avec celles des autres parties du TP.
Question : Considérez deux joueurs avec des statistiques de performance données par :
– Joueur 1 : $[\mu x, \mu y] = [0, 0], [\sigma^2 x, \sigma^2 y] = [0.5, 0.5]$;
– Joueur 2 : $[\mu x, \mu y] = [0.5, 0.5], [\sigma^2 x, \sigma^2 y] = [0.01, 0.01]$;
Pouvez-vous dire lequel de ces deux joueurs est le plus habile?

Cette page est laissée blanche intentionnellement

TP 3 : Mise en Œuvre d'un Filtre Analogique

3.1 Objectif du TP

Le but de ce TP est de mettre en œuvre des filtre analogiques passe-bas.

3.2 Travail à réaliser

Remarque : lancez `start` avant de démarrer la pratique.

1. Signal à étudier
   Dans Matlab, tapez `load tp3.mat`.
   Visualisez le signal $x(t)$.
   Sachant que ce signal est une somme de sinus, déterminez leur nombre, ainsi que les fréquences correspondantes. Pour cela, on utilisera la commande `fft` qui calcule la transformée de Fourier d'un signal; ainsi on pourra afficher son spectre.
2. Filtre passe-bas du premier ordre
   Réalisez un filtre analogique du premier ordre supprimant la plus haute fréquence.
   On utilisera la commande `lsim` pour simuler le système.
   Réaliser maintenant un filtre du même type supprimant les deux plus hautes fréquences.
   Les résultats sont-ils acceptables ?
   Pourquoi ?
3. Blackbox
   On entre maintenant le signal de départ dans une boite noire.
   Pour cela, on tape `y=blackbox(x)`, si le signal de départ est x.
   Que remarque-t-on ?
   Comparez les résultats avec ceux obtenus en 2.
   Pour avoir des précisions sur le contenu de la boite noire, éditez `blackbox.m`.
   De quel type de filtre s'agit-il ?
   Donnez ses caractéristiques.

Cette page est laissée blanche intentionnellement

TP 4 : Traitement d'image

4.1 But du TP

Le but de ce TP est d'écrire quelques routines simples de traitement d'image.

4.2 Introduction

L'affichage des images sur Matlab se fait en trois étapes :
- le chargement de l'image à partir d'un fichier dont on connaît la structure.
  Généralement on importe une image que l'on a numérisée et ensuite stockée dans un format donné. Dans ce TP, le format des images est une matrice de 256 x 256 en 256 niveaux de gris. La fonction de lecture est la suivante :

```
function y=lit_img(nom_de_fichier)
% fonction permettant de lire une image test pour les TP.
%

fid=fopen(nom_de_fichier);
% on charge le fichier
if fid==-1
msg=['Erreur de chargement du fichier : ',nom_de_fichier];
disp(msg)
else
y=zeros(256,256);
y=fread(fid, 256*256,'uchar');
y=reshape(y,256,256)';
% on lit le fichier en une seule passe la matrice de 256x256 pixels
% représentés par un caractère non signés (8 bits) et on recrée la matrice de
% 256x256 et on transpose.
end

fclose(fid)
% on ferme le fichier
```

- la sélection de la palette de couleurs propre à l'image à l'aide de l'instruction colormap.
  Par exemple `colormap('gray(256')` permet de selectionner la palette de couleur avec 256 niveaux de gris.
- l'affichage proprement dit à l'aide de l'instruction image.

Par exemple, le fichier gatlin.mat est un fichier Matlab contenant une image et sa palette de couleurs. Pour l'afficher, on tapera les instructions suivantes :

```
>>load gatlin
```

```
>>colormap(map)
>>image(X).
```

On pourra aussi afficher l'image contenue dans `clown.mat`. La fonction `imageext` permet d'afficher les images de démonstration de Matlab.

1. Ecrire le programme qui permet d'afficher l'histogramme d'une image.
2. Ecrire la fonction qui permet de faire une égalisation d'histogramme d'une image. L'algorithme est le suivant :
   - on calcule l'histogramme `hist_in` de l'image d'entrée `img_in`
   - on calcule la somme cumulée `somme_histo` d'`hist_in` (avec la fonction `cumsum`)
   - on calcule l'image égalisée `img_eq` telle que :

   $$\text{img\_eq}(i,j) = \frac{\text{nombre de niveaux de gris}}{\text{taille de l'image}} \times \text{somme\_histo}(\text{img\_in}(i,j))$$

   Justifier cette formule et appliquer votre fonction aux images de test.
3. En utilisant l'instruction `conv2` qui permet de réaliser un produit de convolution de deux matrices, appliquer les masques de Kirsh, Prewitt et Sobel aux images de test. Quel est le résultat obtenu ?

   De l'image `damier.img`, extraire les lignes horizontales et ensuite les lignes verticales.
4. Ecrire la fonction qui permet d'effectuer le seuillage d'une image.

Annexe A : Le progiciel Matlab

A.1 Introduction

Le logiciel Matlab (MATrix LABoratory) est un progiciel de calcul constitué d'un noyau de base extentsible à l'aide de nombreuses boîte à outils officielles et des boîtes à outils personnelles. Il utilise un langage de programmatin proche du C, semi compilé.
Il présente comme avantage la disponibilité d'un grand nombre d'algorithme de calcul numérique "tout programmés", de nombreuses fonctions de tracé de graphes 2-D et 3-D, la possibilité de calcul direct sur les nombres complexes et l'existence d'une aide en ligne avec la commande `help`

A.1.1 Fenêtres de Matlab (ou liées à Matlab)

Lorsqu'on travaille sous Matlab, on rencontre plusieurs types de fenêtres :
- Fenêtre commande : cette fenêtre est utilisée pour l'entrée de commandes pour exécution immédiate (caractère de sollicitation ou prompt `>>`)et l'on efface la command tapée par la touche `backspace` ou encore `home` ou encore la commande `clc`;
- Fenêtre graphique : cette fenêtre sert à l'affichage de graphiques 2-D ou 3-D et s'efface par la commande `clg`;
- Fenêtre de texte : cette fenêtre peut être intern (éditeur intégré) ou externe. Elle est utilisée pour l'édition de fichiers de commandes ou de fonctions.

A.2 Matrices

Matlab comporte six *types* (ou classes) fondamentaux qui se présentent tous sous forme d'un tableau multidimensionnel. Ces six classes sont `double` (nombre flottants en double précision), `char` (tableau ou chaines de caractères)n `sparse` (matrices creuses), `uint8` (nombres entiers non signés codés sur 8 bits), `cell` (tableau de cellules) et `struct` (tableaux de structures ou enregistrement).
Une matrice comportant une seule ligne ou une seule colonne et nommée un *vecteur*. Une matrice ne comportant qu'une seule et qu'une seule colonne est nommée un *scalaire*.

A.2.1 Nombres complexes

Le plupart des opérations Matlab connaissent les nombres complexes. Voici deux manières de saisir les nombres complexes dasns une matrice :

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

Lors de l'écriture des nombres complexes, il faut faire attention de ne pas mettre d'espace entre le nombre et l'unité imaginaire. i et j désigne indifféremment l'unité imaginaire.

A.2.2 Variables

A priori, toutes le variables de Matlab sont des matrices réelles ou complexes,avec comme cas particuliers les vecteurs ligne et colonne et les scalaires. La déclaration spécifique des variables n'est pas nécessaires en Matlab : il se produit une augmentation de taille automatique en cours d'exécution en fonctions des besoins.
Matlab n'effectue pas la distinction entre les entiers et les réels. La syntaxe pour les noms de variables suit les règles standards Fortran, C ou Pascal. Matlab effectue la distinction majuscules/minuscules. Des informations sur les variables sont visualisables par les commandes `who` et `whos`. L'effacement de variables de la zone de travail est possible avec la commande `clear` suivi du nom de la ou les variables séparées par des espaces. La commande `clear all` efface toutes les variables de la zone de travail.

A.2.3 Entrées des matrices

Dans un programme ou la fenêtre de commande :

```
>> a=[1 2 3; 4 5 6; 7 8 9] %séparation par des espaces

a =

     1     2     3
     4     5     6
     7     8     9

>> a=[1,2,3; 4,5,6; 7,8,9] %séparation par des virgules

a =

     1     2     3
     4     5     6
     7     8     9

>> b=[1,2,3; 4,5,6; 7,8,9]; %point virgule de fin pour éviter l'affichage
>> b

b =

     1     2     3
     4     5     6
     7     8     9

>> c=[a b] %concaténation en colonnes

c =
```

```
     1     2     3     1     2     3
     4     5     6     4     5     6
     7     8     9     7     8     9

>> d=[1 2 3 4 5 6]

d =

     1     2     3     4     5     6

>> e=[c ;d] %concaténation en lignes

e =

     1     2     3     1     2     3
     4     5     6     4     5     6
     7     8     9     7     8     9
     1     2     3     4     5     6
```

A.2.4 Indexation - Extraction de sous-matrices

```
>> f=e(2:3,3:5) %extraction de sous-matrices

f =

     6     4     5
     9     7     8

>>
```

A.2.5 Initialisations de matrices

- Mise à zéro : commande `zeros` : `zeros(m,n)` est une matrice de zéros comportant `m` lignes et `n` colonnes, alors que `zeros(A)` est une matrice de zéros de taille identique à `A`;
- Remplissage de 1 : commande `ones` (syntaxes identiques à celles de `zeros`);
- Matrice identité (unité) : commande `eyes` : `eyes(n)` est la matrice unité de rang `n`.

A.2.6 Taille

- `size` avec la syntaxe `[m,n]=size(x)` fournit le nombre de lignes `m` et de colonnes `n` de la matrice `x`.

```
>> size(a)

ans =

     3     3
```

```
>>
```

Remarque : `ans` est une variable implicite contenant le résultat de l'évaluation d'expressions sans affectation à une variable.
- `legnth` fournit la longueur d'un vecteur.

A.2.7 Opérations matricielles

- Addition : `X=A+B`
- Soustraction : `X=A-B`
- Multiplication : `X=A*B` (Attention le nombre de colonnes de `A` doit être égal au nombre de lignes de `B`).
- Division :
  - division à droite : `X=A/B` soit `A=X*B`
  - division à gauche : `X=A\B` soit `A=B*X`

Remarque : les divisions matricielles sont en général plus précises que la fonction inversion de matrice `inv`.
- Élevation à la puisance : `X=A^B`
- Transposition : `A'` est la transposée de `A` (Attention si `A` est complexe on obtient en fait la transposée conjuguée). En fait il s'agit de transposition-conjugaison (matrice adjointe) Ceci se réduit à la transposition dans la cas d'une matrice réelle.
- Transposition : `.'` opère la transposition simple (matrice réelle ou complexe);
- Déterminant : `det(A)`
- Inverse : `inv(A)`
- Valeurs et vecteurs propres : `eig(A)` avec la syntaxe `[V,D]=eig(A)`, `V` est la matrice des vecteurs propres rangés par colonnes et `D` la matrice diagonale des valeurs propres.

A.2.8 Autres types

Les chaînes de caractères (*string*) sont représentées efficacement par le type `char`. On forme une chaîne de caractère en insérant du texte entre 2 apostrophes (*single quotes*) :

```
texte = 'Matlab est puissant';
```

Les matrices creuses sont des matrices dont la plupart des coefficients sont nuls. Matlab dispose de fonctions pour travailler efficacement avec les matrices creuses. Se reporter à l'aide en ligne de `sparse` et `full`.
Les images sont stockées à l'aide du type `uint8`.
Les tableaux de cellules sont des collections de tableaux qui s'obtiennent en plaçant la liste de ces tableaux entre accolades :

```
c = {1:5, 'Matlab est puissant', rand(1,5)};
```

Il est possible de fabriquer d'autres types de données en utilisant la surcharge (aide sir `class`).

A.2.9 Vecteurs et polynômes

– Génération d'éléments régulièrement espacés : il y a plusieurs formes possibles :

```
>> D=1:4

D =

     1     2     3     4

>> E=0:0.1:0.5

E =

          0    0.1000    0.2000    0.3000    0.4000    0.5000

>>
```

– Boubles implicites avec vecteurs : si `V` est un vecteur, `U=sin(V)` est un vecteur dont les éléments sont les sinus de ceux de `V`;
– Cas spécial : polynômes
  – La fonction `poly` (`P=poly(A)`) fournit le polynôme caractéristique, représenté selon l'ordre des puissances décroissantes.
  – La fonction`roots` fournit les racines d'un polynôme. dans le cas ci-dessus `roots(P)` donne les valeurs propres de `A`.
  – La fonction `conv` effectue la multiplication de polynômes. Par exemple `C=conv(A,B)`.

A.3 Opérations matricielles et élémént par élément

Attention les 4 opérations élémentaires, plus l'élévation à la puissance (opérateur `^`, sont des opérations matricielles, c'est à dire portant sur les matrices dans leur ensemble.
`C=A*B` effectue le produit matriciel. Par contre, si `A` et `B` sont des matrices de taille identique, on peut effectuer leur multiplication élément par élément (que l'on notera `C=A.*B`). Dans le premier cas : $c_{ij} = a_{ik} \times b_{kj}$ alors que dans le second : $c_{ij} = a_{ij} \times b_{ij}$.
L'opérateur matriciel est donc simplement précédé par un point, qu'il faudra ne faudra pas confondre avec le séparateur décimal (toujours écrire `1.0` et non `1.`.
Matlab définit également des fonctions matricielles générales en parallèles des fonctions classiques qui opèrent sur les éléments : `expm` exponentielle matricielle, `logm` logarithme matriciel, `sqrtm` racine caréée matricielle ou `funm` fonction matricielles générale.

A.4 Déclaration, expressions et variables

Matlab présente un langage de programation interprété. Cela signifie que les expression saisie sont immédiatement interprétées et exécutées.
Les déclarations Matlab sont de la forme *variable = expression* ou simplement *expression*.
Les expressions sont généralement composées d'opérateurs, de fonctions et de noms de variables. L'évaluation d'une expression conduit à une matrice qui est affichée à l'écran ou affectée à une nouvelle variable en vue d'une utilisation future. Si l'on omet de donner un nom de variable suivi du signe `=`, le résultat est automatiquement affecté à une variable

par défaut nommée `ans` (pour *answer*).
Une déclaration est normalement terminée par la saisie d'un retour-chariot (touche *entrée*),. Toutefois une déclaration peut être répartie sur deux ou plusieurs lignes à condition de faire précéder le retour-chariot séparant deux lignes consécutives de la déclaration par au moins trois points (`...`). On peut aussi placer plusieurs déclarations sur une même ligne à condition de les séparer par des virgules ou des points virgules.

A.4.1 Suppression de l'affichage des résultats

Si le dernier caractère d'une déclaration est un point virgule, l'affichage des résultats est supprimé (mais bien entendu, la déclaration est exécutée). On peut anisi se débarrasser des résultats intermédiaires fastidieux.

A.4.2 Majuscules et Minuscules

Matlab tient compte de la casse des lettres, c'est à dire qu'il fait la distinction entre majuscules et minuscules. Ainsi le variable `SolveUT` est différente de `Solveut`.

A.4.3 Liste de variables et de fichiers `M`

La commande `who` ou `whos` permet d'obtenir la liste des variables définies dans l'espace mémoire de la session courante. De même le commande `inmem` permet de connaître la liste de fichiers `M` compilés couramment chargés en mémoire.
Une variable ou une fonction peut être effacée de l'espace mémoire avec la commande `clear nom_fonction` (ou `clear nom_variable`). Si la commande `clear` est utilisée sans être suivie de nom, on efface de l'apces mémoire toutes les variables non permanentes. De la même façon, `clear functions` efface de l'espace mémoire tous les fichiers M compilés.

A.4.4 Interruption d'un calcul

Sur la plupart des machines un calcul, ou, plus généralement, l'exécution d'un programme, peut être interrompu ssns quitter Matlab en pressant simultanément les touches *Ctrl* et *c*.

A.5 Structure de Contrôles

Dans leur forme élémentaire, les structures de contrôle de Matlab fonctionnent de la même manière que dans la plupart des autres langages de programmation.

A.5.1 Boucles inconditionnelles `for`

La forme générales d'une boucle `for` est :

```
for i=1:n instructions end
```

Les *instructions* vont eêtre exécutées n fois pour des valeurs de i qui sont succéssivement 1,2,...n. Par exemple, les instructions :

```
x=[];
n=3;
for i=1:n
x=[x,i^2]
end
```

produit `[1 4 9]` et :

```
x=[];
n=3;
for i=n:-1:1
x=[x,i^2]
end
```

produit `[9 4 1]`. On remarque qu'une matrice peut être vide `[]`.
L'intervalle `1:n` qui définit l'évolution du compteur d'une boucle `for` peut être remplacer par n'importe quelle matrice. Dans ce cas le compteur parcourt les colonnes successives de la matrice en question. Par exemple :

```
s=0;
for c=A
s= s + sum(c);
end
```

permet de calculer la somme des coefficient de la matrice `A` (toutefois `sum(sum(A))` est un moyen plus efficace).

A.5.2 Boucles conditionnelles `while`

La forme générale d'une boucle `while` est :

```
while condition instructions end
```

Les *instructions* vont être exécutées de façon répétée tant que la *condition* est satisfaite. Par exemple, étant donnée un nombre a, la boucle suivante détermine le plus petit entier positif tel que $2^n > a$:

```
n=0;
while 2^n < a
n=n+1;
end
n
```

A.5.3 Branchements conditionnels `if`

La forme générale d'une instruction de brachement conditionnel est la suivante :

```
if condition instruction1 else instruction2 end
```

Si la *condition* est satisfaite, le jeu d'*instructions 1* va être exécuté. Dans le cas contraire, c'est le jeu d'*instructions 2* qui va être exécuté. On peut aussi recourir à un branchement multiple à l'aide du mot réservé `elseif` comme le montre l'exemple suivant:

```
if n < 0
parité = 0;
elseif rem(n,2) == 0
parité = 2;
else
parité = 1;
end
```

A.5.4 Opérateurs relationnels et opérateurs logiques

On dispose des opérateurs relationnels suivants :

`<`	strictement inférieur
`>`	strictement supérieur
`<=`	inférieur ou égal
`>=`	supérieur ou égal
`==`	égal
`~=`	différent

Il convient de bien distinguer l'opérateur d'affectation `=` du test d'égalité `==`.
Les expressions peuvent être reliées par les opérateurs logiques suivants :

`&`	Et
`|`	Ou
`~`	Non

Le résultat d'un test dépend de la nature des objets auxquels il est appliqué. Si l'on effectue un test sur des scalaires, le résultat est 1 ou 0 selon que la relation est vérifiée ou non.
Lorsqu'un test est effectué sur des matrices de mêmes dimensions, la réponse est une matrice, de mêmes dimensions que les matrices intervenant dans le test, dont les coefficients sont des 0 ou des 1 selon le résultat de la comparaison entre les coefficients des matrices concernées.
Un test faisant intervenir des matrice dans la partie condition d'une instruction `while` ou d'une instruction `if` est considéré comme satisfait lorsque chaque coefficient de la matrice résultat est non nul. De la sorte, si l'on veut exécuter un jeu d'instructions à la condition que le matrices `A` et `B` soient égales, on peut écrire :

```
if A == B
<instructions>
end
```

Alors que si l'on veut exécuter les instructions seulement si `A` et`B` sont distinctes, on peut écrire :

```
if any(any(A ~= B))
<instructions>
end
```

ou encore plus simplement

```
if A == B else
<instructions>
end
```

Il convient de faire attention à l'instruction`if A ~= B, <instruction>, end` qui n'est pas aussi évidente qu'il y paraît : les instructions seront exécutées lorsque chaque coefficient de `A` est distinct du coefficient correspondant de `B`. Les fonction `all` et `any` peuvent être utilisées astucieusement pour ramener un test sur des matrices à un test sur des scalaires.

A.6 Fonctions Matlab prédéfinies

A.6.1 Fonctions scalaires

Certaines fonctions Matlab sont conçues pour s'appliquer à des scalaires maisvont être distribuées sur tous les coefficients si elles sont appliquées à une matrice. On peut citer les fonctions les plus courantes :

`sin`,`cos`, `tan`	fonctions trigonométriques circulaires directes
`asin`,`acos`, `atan`	fonctions trigonométriques circulaires reciproques
`abs`	valeur absolue
`sqrt`	racine carrée
`floor`	partie entière (plus grand entier inférieur ou égal)
`round`	plus proche entier
`ceil`	plus petit entier supérieur ou égal
`sign`	signe
`exp`	exponentielle népérienne
`log`	logarithme népérien
`log10`	logarithme en base 10
`rem`	reste d'une division euclidienne

A.6.2 Fonctions vectorielles

D'autres fonctions Matlab sont conçues pour s'appliquer à des vecteurs (lignes ou colonnes) mais vont être sitribuées sur les vecteurs colonnes si elles sont appliquées à une matrice et produire un résultat sous la forme d'un vecteur ligne. On obtient une distribution de la fonction sur les vecteurs lignes en ayant recours à la transposition (par exemple, `mean(A')'`) ou en spécifiant la dimension sur laquelle doit s'appliquer la fonction (par exemple `mean(A,2)`). Voici quelques unes de ces fonctions (qui s'appliquent toutes aux coefficients du vecteur passé en argument) :

`max`	maximum
`min`	minimum
`sum`	somme
`prod`	produit
`median`	médiane
`mean`	moyenne
`std`	écart type
`sort`	tri
`any`	teste si l'un des éléments de la matrice considérée est non nul
`all`	teste si tous les éléments de la matrice considérée sont non nuls

A.6.3 Fonctions matricielles

La puissance de Matlab vient des nombreuses fonctions matricielles implémentées par le logiciel. Certaines de ces fonctions peuvent fournir une ou plusieurs valeurs en sortie. Par exemple, `y=eig(A)` ou simplement `eig(A)` produit un vecteur colonne constitué des valeurs propres de `A` tandis que `[U,D]=eig(A)` produit la matrice `U` dont les colonnes sont constituées de vecteurs propres de `A` et la matrice diagonale `D` comportant les valeurs propres correspondantes sur sa diagonale. Parmi d'autres, on peut citer :

`eig`	valeurs et vecteurs propres
`chol`	factorisation de Cholesky
`svd`	décomposition en valeurs singulières
`inv`	inverse
`lu`	factorisation LU
`qr`	factorisation QR
`hess`	mise sous forme de Hessenberg
`schur`	décomposition de Schur
`rref`	réduite echelonnée par lignes (obtenue par la méthode de Gauss avec pivot partiel)
`expm`	exponentiation matricielle
`sqrtm`	racine carrée matricielle
`poly`	polynôme caractéristique
`det`	déterminant
`size`	dimensions
`norm`	norme-1, norme-2, norme-∞, norme de Frobenius
`rank`	rang
`cond`	conditionnement

A.7 Édition de ligne

La ligne de commande peut être facilement traitée sous Matlab. Le curseur peut être positionné à l'aide des touches de déplacement (droite et gauche).
Les touches de déplacement haut et bas permettent de circuler dans la pile des instructions précédemment exécutée.

A.8 Sous-matrices

Les vecteurs et les sous-matrices sont fréquemment utilisés avec Matlab pour accomplir des manipulations relativement complexes sur les données. Le symboles : et l'indexation par des vecteurs d'entiers sont la clé d'une programmation efficace. Une bonne utilisation de ces techniques permet d'éviter le recours à des boucles et rend le code simple et lisible.

A.8.1 Génération de vecteurs

L'expression `1:5` représente le vecteur ligne `[1 2 3 4 5]`. Il n'est pas nécessaire de prendre des valeurs entières pour les coefficients du vecteur ni pour l'incrément. C'est ainsi que `0.2:0.2:1.2` donne `[0.2,0.4,0.6,0.8,1.0,1.2]` et `5:-1:1` donne `[5 4 3 2 1]`.
Ainsi pour obtenir une tables des sinus on peut faire :

```
x = [0.0:0.1:2.0];
y=sin(x);
[x y]
```

A.8.2 Accès aux sous-matrices

La notation : peut être utilisée pour désigner des sous-matrices d'une matrice. Par exemple `A(1:4,3)` désigne le vecteur colonne constitué des quatres premiers élément de la troisième colonne de `A`.
Le symbole : tout seul signifie une colonne ou une ligne toute entière : `A(:,3)` désigne la troisième colonne de `A`, `A(1:4,:)` désigne les quatre premières lignes de `A`.
On peut aussi utiliser des vecteurs d'entiers pour définir des colonnes. Par exemple `A(:,[2 4])` contient la deuxième et quatrième colonne de `A`. Une telle manière d'indexé les lignes ou les colonnes peut être encore utilisée de la manière suivante :

```
A(:,[2 4 5]) = B(:,1:3);
```

remplace les colonnes 2, 4 et 5 de `A` par les trois preimères colonnes de `B`.
De même on peut multiplier (à droite) les deuxième et quatrième colonnes de `A` par la matrice d'ordre 2 `[1 2;3 4]` en écrivant :

```
A(:,[2,4])=A(:,[2,4])*[1 2;3 4];
```

Il existe un indice particulier permettant de désigner le dernier élément d'un vecteur, il s'appelle `end`.

A.9 Fichier `M`

Matlab peut exécuter une suite d'instructions stockées dans un fichier. Un tel fichier est appelé un «fichier M» (pour *M file*) parce que son nom doit présenter l'extension «`.m`».
Il existe deux types de fichiers `M` : les *fichiers de commandes* ou scripts et les *fichiers de fonctions*.

A.9.1 Fichiers de commandes (scripts)

Un *fichier de commandes* consiste en une suite de déclarations Matlab normales. Si par exemple, le fichier est nommé `toto.m`, alors l'instruction `toto` provoque l'exécution des instructions contenues dans ce fichier. Les variables définies dans un fichier de commandes sont globales et si dans la session, des variables de même nom existaient préalablement au chargement du fichier, ces dernières seront modifiées conformément aux instructions du fichier.
Un fichier de commande peut être utilisé pour entrer les coefficients d'une matrice de grande dimension. En procédant de la sorte, on peut facilement vérifier la saisie des coefficients.
Enfin un fichier `M` peut appeller un autre fichier `M` et peut aussi s'appeler lui-même de manière récursive.

A.9.2 Fichiers de fonctions

Les *fichiers de fonctions* permettent d'étrebdre Matlab. On peut créer de nouvelles fonctions spécifiques à un domaine particulier, et attribué à ces fonctions un statut analogue à celui des fonctions prédéfinies. Les variables définies dans un fichier de fonction sont; par défaut, considérées comme locales. Toutefois si on le souhaite, une variable peut être déclarée comme globale (se rapporter à l'aide de `global`.
Voici un exemple simple de fichier de fonction :

```
function y =y randint(m,n)
% Génération aléatoire d'une matrice
% à coefficients entiers
% randint(m,n) renvoie une matrice
% de dimension m x n à coefficients compris entre 0 et 9
y = floor(10*rand(m,n));
```

dont une version plus générale est :

```
function y =y randint(m,n,a,b)
% Génération aléatoire d'une matrice
% à coefficients entiers
% randint(m,n) renvoie une matrice
% de dimension m x n à coefficients compris entre 0 et 9
% randint(m,n,a,b) renvoie une matrice
% de dimension m x n à coefficients compris entre a et b
if nargin < 3, a=0; b = 9; end
y = floor((b-a+1)*rand(m,n))+a;
```

Ces instructions doivent être placées dans un fichier appelé `randint.m` (nom qui correspond au nom de la fonction).
La première ligne du fichier consiste en la déclaration du nom de la fonction, de ses arguments d'entrée et de sortie. En l'absence de cette ligne, le fichier sera considéré comme un simple fichier de commaned. Dès lors, l'instruction Matlab `z = randint(4,5)` provoque l'affectation du résultat de l'application de la fonction au couple *(m,n)=(4,5)* à la variable `z`.
Signalons l'existence de la variable `nargin` qui, au moment de l'appel à une fonction, contient le nombre d'arguments effectivement transmis à la fonction, ce qui permet de

donner une valeur par défaut aux paramètres omis, comme cela est fait pour `a` et`b` dans la deuxième version.

A.9.3 Sorties multiples

Une fonction peut fournir plusieurs variables en sortie. Voici un exemple de cette situation :

```
function [moy,et]=stat(x)
% Calcul de la moyenneet de l'écart type
% x désignat un vecteur, stat(x)
% renvoie la moyenne des coefficients de x alors que
% [moy,et] = stat(x) renvoie simultanément
% le moyenne et l'écart-type des coefficients de x.
% Si x est une matrice, stat(x) s'applique à chaque
% colonne de x
[m n]=size(x);
if m ==1
m=n; %traite le cas d'un vecteur ligne
end
moy = sum(x)/m;
et = sqrt(sum(x.^2)/m-moy.^2);
```

Lorsque cette fonction a été sauvegardée dans la fichier `stat.m`, l'instruction Matalab `[xm, xe] = stat(x)` affecte respectivement à xm et xe la moyenne et l'écart type des données contenues dans le vecteur x. On peut aussi ne garder qu'une sortie d'une fonction qui en produit plusieurs. Par exemple `xm = stat(x)` affecte à xm la moyenne des données contenues dans x.

A.9.4 Commentaires et aide en ligne

Le symbole `%` annonce une ligne de commantaire : tout ce qui suit ce signe est considéré comme un commentaire et n'est pas pris en compte par l'interpréteur. En outre, le premier bloc de lignes de commentaires contiguës constitue de la fonction : si l'on prend le fichier `stat.m` donné en exemple, les lignes en question seront affichées lors d'un appel à `help stat`. Il est clair que l'on devrait toujours inclure une telle documentation lors de la rédaction d'un fichier `M`.

A.10 Chaînes, messages d'erreur et entrées

Une chaînes de caractères (*string*) doit être entourée d'apostrophes (*singles quotes*) en Matlab. L'instruction :

```
s= 'Ceci est une chaîne de caratères'
```

affecte le texte mentionné à la variable `s`. Les chaînes de caractères (comme les matrices à coefficients numériques) peuvent être affichées avec la commande `disp`. C'est ainsi que `disp(s)` permettra d'afficher le contenu de `s`.

A.10.1 Message d'erreur

On peut provoquer l'affichage d'un message d'erreur avec la commande `error` comme, par exemple :

```
error('Désolé, la matrice doit être symétrique');
```

Lorsque Matlab rencontre une instruction de ce genre dans un fichier `M`, l'exécution de ce fichier est interrompue.

A.10.2 Entrées

Dans un fichier `M`, on peut demander à l'utilisateur de saisir interactivement une valeur en ayant recours à la commande `input`. Par exemple, l'instruction :

```
iter = input ('Indiquer le nombre de lignes :');
```

va provoquer l'affichage du message et interrompre l'exécution de la fonction de manière à ce que l'utilisateur puisse saisir la valeur demandée. Lorsque l'utilisateur presse la touche *entrée* (signifiant qu'il a terminé sa saisie), la valeur saisie est affectée à la variable `iter` et l'exécution de la fonction reprend.

A.11 Gestion des fichiers `M`

Au cours d'une session Matlab, il peut arriver que l'on ait besoin de créer ou modifier un fichier `M` à l'aide d'un éditeur de texte avant de revenir à Matlab alors qu'on veut éviter de quitter Matlab pour ne pas perdre les variables déjà calculées.

A.11.1 Exécution de commandes systèmes

On peut lancer l'exécution d'une commande système,sans quitter Matlab, l'aide de la commande spéciale `!` : il suffit de faire précéder la commande système en question par le symbole `!`.

A.11.2 Gestion des répertoires et des fichiers

Sous Matlab, la commande ·`pwd` permet d'obtenir le nom du répertoire courant et la commande `cd` permet de changer de répertoire. La commade `dir` (ou `ls`) fournit le catalogue des fichiers contenus dans le répertoire courant tandis que la commande `what` ne liste que les fichiers du répertoire en rapport avec Matlab en les regroupant par type de fichiers. La commande `delete` permet d'effecer un fichier et `type` permet d'afficher à l'écran un fichier `M`. Bien entendu toutes ces commandes font double emploi avec une commande système accessible grâce à `!`.

A.11.3 Matlab et chemins d'accès

Les fichiers invoqués doivent se trouver dans un répertoire accessible à Matlab. Tous les fichiers `M` se trouvant dans le répertoire courant sont toujours accessibles. La plupart des installation de Matlab permettent de définir un répertoire nommé `matlab` auquel Matlab peut toujours accéder pour ouvrir les fichiers. La liste des chemins d'accès couramment

connus s'obtient à l'aide de la commande `path`. Elle permet aussi d'ajouter ou d'enlever des répertoires à la liste (`help path`). La commande `which` permet de trouver les fichiers dans le chemin d'accès.

A.12 Mesure de l'efficacité d'un programme

Le nombre d'opérations effectuées (*flops* pour *floating pont operations*) et le temps de calcul constituent deux moyens pour apprécier l'efficacité d'un programme.

A.12.1 Fonction `flops`

La fonction Matlab `flops` totalise le nombre d'opérations effectuées au cours d'un calcul. L'instruction `flops(0)` réinitialise ce total à 0. Ainsi en lançant l'instruction `flops(0)` avant l'exécution d'un algorithme, puis l'instruction `flops` immédiatement après cette exécution, on obtient le nombre d'opérations mises en œuvre au cours de l'exécution. Par exemple :

```
flops(0), x=A/B; flops;
```

Permet de savoir combien d'opérations a necessisté la résolution du système linéaire $Ax = b$ par la méthode de Gauss.

A.12.2 Temps de Calcul

Le temps (en secondes) requis par un clacul peut être mesuré à l'aide des commandes `tic` et`toc`. `tic` permet de déclencher le chronomètre tandis que `toc` permet de connaître le temps écoulé depuis la dernière exécution de `tic`. C'est ainsi que :

```
tix, x=A/b; toc
```

permet de connaître le temps nécessaire à la résolution du système linéaire $Ax = b$.

A.12.3 Profileur

Matlab met à disposition de l'utilisateur un profileur qui permet de connaître le temps de calcul requis par chaque ligne d'un fichier `M` donné. Se reporter à l'aide en ligne `help profile` détaillant la commande `profile`.

A.13 Formats de sortie

Bien que tous les calculs soient effectués en double précision, le format d'affichage des sorties peut être contrôlé à l'aide des commandes suivantes (entre parenthèses le nombres de chiffres significatifs du format) :

`format short`	flottant court (4)
`format long`	flottant long (14)
`format short e`	flottant court en notation scientifique (4)
`format long e`	flottant long en notation scientifique (15)
`format hex`	nombre hexadécimal
`format+`	nombre signé (affichage d'un signe + ou d'un signe -)

Le format `short` est retenu par défaut. Une fois invoqué, le format choisi reste actif jusqu'à ce qu'il soit modifié par un nouvel appel à `format`. Notons que cette commande affecte l'affichage du nombre considéré mais n'a aucun effet sur sa précision.
L'instruction `format compact` conduit à la suppression de la plupart des lignes vides dans l'affichage, permettant l'affichage de davantage d'informations à l'écran. Inversement l'instruction `format loose` permet de revenir à un affichage moins dense.

A.14 Représentations graphiques

Matlab permet la représentation de courbes planes, de courbes gauches, de nappes à l'aides de maillages ou surfaces. On prénsetera les principales commandes `plot`, `plot3`, `mesh`, `surf` et `light`. On peut lancer la commande `demo` afin d'avoir une idée des possibilités de ces commandes.

A.14.1 Graphiques en dimension 2

La commande `plot` sert à tracer des courbes planes. Plus précisément, si x et y désignent des vecteurs de même longueur, l'appel à `plot(x,y)` ouvre une fenête graphique et trace une ligne brisée joignant les points dont les coordonnées sont définies par les listes x et y. Par exemple, une manière d'obtenir la représentation graphique de la fonction *sin* sur l'intervalle $[-4, 4]$ consiste à écrire :

```
x=-4.0:0.01:4; y=sin(x); plot(x,y);
```

La commande `zoom` permet d'agrandir ou de diminuer l'echalle d'un graphique.

A.14.2 Graphiques multiples

Matlab peut gérer plusieurs représentations graphiques simultanément. L'une d'elles constitue la figure courante. C'est elle qui reçoit les représentations graphiques résultant de l'exécution des commandes. Si l'on dispose déjà d'une première fenêtre graphique avec une figure, alors l'instructin `figure(2)` (ou plus simplement `figure`) va créer une deuxième fenêtre graphique qui va devanir la figure courante. L'instruction `figure(1)` à pour effet d'afficher à nouveau le première figure et d'en faire la figure courante. La commande `gcf` permet de connaître le numéro de figure courante.

A.14.3 Graphe d'une fonction

La commande `fplot` sert à obtenir simplement et efficacement la représentation graphique d'une fonction. Par exemple, on peut représenter le graphe de $y = e^{-x^2}$ en créant un fichier `M` appelé `expnormal.m` contenant les lignes :

```
function y = expnormal(x)
y = exp(-x.^2);
```

et en exécutant l'instruction :

```
fplot('expnormal',[-1.5,1.5]);
```

A.14.4 Courbes paramétrées

Matlab permet aussi de représenter des courbes paramétrées planes. Voici un exemple montrant comment procéder :

```
t=0:.001:2*pi;
x=cos(3*t);
y=sin(2*t);
plot(x,y);
```

A.14.5 Titres, légendes, textes

Les graphiques peuvent être agrémentés de titres, de légendes ou de textes. On utilise dans ce but les fonctions suivantes, qui prennent toutes une chaîne de caractères en argument.

`title`	titre du graphique
`xlabel`	légende associée à l'axe des abscisses
`ylabel`	légende associée à l'axe des ordonnées
`gtext`	positionne du texte sur le graphique à l'aide de la souris
`text`	positionne du texte sur le graphique en un point spécifié parses coordonnées

Par exemple l'instruction :

```
title('La fonction exponentielle');
```

donne un titre à la figure courante. L'instruction `gtext('un point')` permet de positionner interactivement le texte considéré en cliquant sur l'emplacement choisi.
La commande `grid` permet de supersposer un quadrillage au grahique.

A.14.6 Axes et échelles

Par défaut, les échelles des axes sont ajustées automatiquement. L'utilisateur peut aussi fixer les échelles à l'aide de la commande `axis`. Voici certaines des options disponibles :

`axis`	(appliqué à $[x_{min}, x_{max}, y_{min}, y_{max}]$) établit une échelle en fonction des bornes spécifiées
`axis(axis)`	gèle l'échelle pour les figures suivantes
`axis auto`	revient à une échelle déterminée automatiquement par Matlab
`v = axis`	place dans le vecteur v la définition de l'échelle courante
`axis square`	force les deux axes à la même longueur (mais leurs échelles ne sont pas nécessairement les mêmes)
`axis equal`	force la même échelle et la même graduation sur les deux axes
`axis on`	restaure les axes
`axis off`	supprime les axes

La commande `axis` doit être exécutée après (et non avant) la commande `plot` qui a créé le graphique.

A.14.7 Graphiques multiples

Il existe plusieurs façons de placer plusieurs représentations graphiques sur la même figure. La première est illustrée par l'exemple suivant :

```
x = 0:0.01:2*pi;
y1=sin(x); y2=sin(2*x); y3=sin(4*x);
plot(x,y1,x,y2,x,y3);
```

La deuxième consiste à former la matrice `Y` contenant les valeurs fonctionnelles en colonnes :

```
x = 0:0.01:2*pi;
Y=[sin(x)',sin(2*x)',sin(4*x)'];
plot(x,Y);
```

La troisième façon consiste à utiliser la commande `hold` qui permet de geler la fenêtre graphique courante de sorte que les représentation s suivantes se superposent sur cette même figure. Dans ce cas, il peut arriver que les échelles soient réajustées. On annule l'effet de `hold` par l'instruction`hold off`.
La fonction `legend` permet d'associer une légende à chaque courbe de la figure (`help legend`).

A.14.8 Types de tracés, types de marqueurs, couleurs

Matlab choisit par défaut les types de tracés, les types de marqueurs et les couleurs mais l'utilisateur peut imposer ses propores choix. Par exemple :

```
x = 0:0.01:2*pi;
y1=sin(x); y2=sin(2*x); y3=sin(4*x);
plot(x,y1,'|',x,y2,':',x,y3,'+');
```

produit un tracé en tiretés, un autre en pointillés, et un troisième avec des symboles `+`.
Voici les différents types de tracés et de marqueurs :
- Types de lignes : `solid (-)`, `dashed (|)`, `dotted (:)`, `dashdot (-.)`
- Types de marqueurs : `point (.)`, `plus (+)`, `star (*)`, `circle (o)`, `x-mark (x)`, `square (s)`, `diamond (d)`, `triangle-down (v)`, `triangle-up (^)`, `triangle-left (<)`, `triangle-right (>)`, `pentagram (p)`, `hexagram (h)`

Des couleurs peuvent être précisées pour les lignes ou les marqueurs :
- `yellow (y)`, `magenta (m)`, `cyan (c)`, `red (r)`, `green (g)`, `blue (b)`, `white (w)`, `black (k)`

Par exemple, `plot(x,y,'r|')` effectue un tracé en tiretés de couleur rouge.

A.14.9 Autres fonctions spécialisées

La commande `subplot` permet de partitionner une figure de manière à placer plusierus petit graphiques sur la même figure (`help subplot`). D'autres fonctions interressantes sont `polar`, `bar`, `hist`, `quiver`, `compass`, `feather`, `rose`, `stairs` ou encore `fill`.

A.14.10 Impression des graphiques

Une impression de la figure graphique courante s'obtient facilement avec la commande `print`. Utilisée sans option, cette commande envoie une copie haute résolution du graphique courant à l'imprimante par défaut.
Le fichier `M printopt` peut être utilisé pour modifier les options par défaut utilisées par la commande `print` (`help printopt`).
L'instruction `print nom_fichier` peut être utilisée pour sauvegarder le graphique courant dans le fichier désigné au format spécifié par défaut. L'instruction `print figures` crée un fichier Postscript appelé `figure.ps` contenant la description de la figure considérée.
Les paramètres par défaut peuvent être modifiés au moment de l'appel. Par exemple :

```
print -deps -f3 graphique
```

place dans un fichier Postscript encapsulé de nom `graphique.eps` une description de la fenêtre graphique 3.

A.14.11 Représentation des courbes gauches

La commande `plot3` est l'analogue de la commande `plot` pour la dimension 3. Si x, y et z désignent des vecteurs de mêmes dimensions, l'instruction `plot(x,y,z)` produit une vue en perspective de la courbe obtenue en reliant les points dont les coordonnées sont respectivement éléments de x, y et z. Les vecteurs sont souvent définis paramétriquement comme dans l'exemple suivant :

```
t=0:0.01:20*pi;
x=cos(t); y=sin(t);z=t.^3;
plot3(x,y,z);
```

qui conduit à la représentation graphique d'une hélice.
Comme dans le cas des courbes en dimension 2, on peut ajouter un titre et des légendes aux axes du graphiques.
`Représentation de nappes` L'instruction `mesh(z)` permet de réaliser une vue en perspective du maillage défini par la matrice z. Plus précisément, le maillage est défini par les cotes `z` de points répartis sur un rectangle du plan `x-y`. Le lecteur peut essayer `mesh(eye(20))`.
Ainsi pour obtenir la représentation de la nappe définie par l'équation $z = f(x, y)$, on commence par déterminer xx et yy, vecteurs constituant une subdivision de chaquue côté du rectangle sur lequel on souhaite effectuer la représentation. On vrée ensuite une matrice x (respectivement y) comportant autant de lignes (respectivement de colonnes) qu'il y a d'éléments dans yy (respectivement xx) et dont toutes les lignes (respectivement colonnes) sont égales à xx (repsectivement yy), avec l'instruction :

```
[x,y]=meshgrid(xx,yy);
```

On clacule alors une matrice z à laquelle on puisse appliquer `mesh` ou `surf`. Á cet effet, on distribue l'application de f sur les matrices x et y.
Ainsi pour représenter la nappe $z = e^{-x^2-y^2}$ sur le carré $[-2, 2] \times [-2, 2]$, on procède de la manière suivante :

```
xx = -2:0.2:2;
yy=xx;
[x,y]=meshgrid(xx,yy);
z = exp(-x.^2-y.^2);
mesh(z);
```

On peut d'ailleurs remplacer les trois premières instructions par :

```
[x,y]=meshgrid(-2:0.2:2,-2:0.2:2);
```

Il est instructif de comparer le résultat obtenu par l'utilisation de `mesh` avec celui obtenu par l'utilisation de `surf`.

A.14.12 Couleurs et ombres portées

Les effets de couleur et d'ombre portée peuvent être définis à l'aide de la commande `shading`. Les trois options possibles, `faceted` (option par défaut), `interpolated` et `flat`, prennent effet après exécution de l'une des instructions `shading faceted`, `shading interp` ou `shading flat`.
Sur une représentation prosuite par `surf`, les options `interpolated` et `flat` font disparaître les lignes définissant le maillage.
La commande `shading` doit être appelée après la commande `surf` ayant produit le graphique. Ceci est aussi valable pour les command e`colormap` et `view`.
Le coloriage d'une figure est défini par la commande `colormap`. Un certain nombre d'options de coloriage prédéfinies existent; on peut citer `hsv` (option par défaut), `hot`, `cool`, `jet`, `pink`, `copper`, `flag`, `gray`, `bone`, `prism` et `white`. Par exemple l'instruction `colormap(cool)` définit un certain profil pour le coloriage de la figure courante. On pourra se reporter à la commande `help colobar` pour plus d'informations.

A.14.13 Perspective d'une vue

La commande `view` permet de définir en coordonnées cartésiennes ou en coordonnées sphériques le point de vue sur un objet graphique (voir `help view`).
La commande `rotate3d` permet de définir ce point de vue interactivement à l'aide de la souris.
Il est aussi possible une caméra et des sources de lumières pour visualiser et éclairer une figure.

Annexe B : Les fonctions usuelles de Matlab

B.1 Commandes générales

Lancement et arrêt de Matlab	
`matlab.rc`	fichier `M` de démarrage principal
`quit`	permet de quitter Matlab
`startup.m`	fichier `M` de démarrage secondaire

Gestion de l'environnement	
`addpath`	ajoute un répertoire à la liste des chemins de recherche de Matlab
`doc`	charge la documention hypertexte
`help`	lance l'aide en ligne
`lasterr`	fournit le dernier message d'erreur
`lookfor`	lance une recherche par mot-clé
`path`	affiche la liste des chemins de recherche
`profile`	mesure le temps d'exécution d'un fichier `M`
`rmpath`	enlève un répertoire de la liste des chemins de recherche Matlab
`type`	liste le contenu d'un fichier
`version`	affiche la version de Matlab en cours d'exécution
`what`	donne la liste des fichiers `M`, `MEX` et `MAT` présents dans le répertoire courant
`whatsnew`	affiche les fichiers `readme` pour Matlab et ses boîtes à outils
`which`	permet de localiser les fichiers et les fonctions

Gestion des variables et de l'espace mémoire	
`clear`	efface les variables choisies de la mémoire
`disp`	permet d'afficher du texte ou des tableaux
`length`	fournit la dimension (longueur) d'un vecteur
`load`	charge des variables depuis un fichier
`pack`	lance la défragmentation de l'espace mémoire de travail
`save`	sauvegarde des variables dans un fichier
`size`	fournit les dimensions d'une matrice
`who`, `whos`	liste toutes les variables connues de l'espace de travail

Gestion de la fenêtre de commande	
`echo`	affiche les commandes du fichier `M` couramment exécuté
`format`	définit le format d'affichage des nombres
`more`	active et désactive le mode d'affichage par page

Gestion des fichiers. Accès au système d'exploitation	
`cd`	change le répertoire de travail
`delete`	efface des fichiers ou des objets graphiques
`diary`	permet la sauvegarde d'une session sur fichier
`dir`	fournit le catalogue d'un répertoire
`edit`	édite un fichier `M`
`inmem`	liste les fonctions en mémoire
`matlabroot`	donne le nom du répertoire temporaire du système d'exploitation
`tempdir`	fournit le nom du répertoire temporaire du système d'exploitation
`tempname`	produit un nom unique pour un fichier temporaire
`!`	lance l'exécution de la commande système dont le nom suit

B.2 Opérateurs et caractères spéciaux

Opérateurs et caractères spéciaux	
+	addition de réels ou de matrices
-	soustraction de réels ou de matrices
*	produit de réels ou de matrices
.*	produit élément par élément de matrices
\	division à gauche de réels ou de matrices
/	division à droite de réels ou de matrices
.\	division à gauche élément par élément de matrices
./	division à droite élément par élément de matrices
.	point décimal
..	désigne le répertoire parent du répertoire courant
...	symbole de continuation (pour terminer une instruction ou une expression à la ligne suivante)
%	commentaire
'	matrice adjointe (transposée + conjugée) et délimiteur de chaîne de caractère
.'	matrice transposée
!	lance une commande système dont le nom suit, tout en restant sous Matlab
=	affectation
==	test d'égalité
~=	test de non égalité
<, <=, >, >=	opérateurs de comparaison
&	ET logique
|	OU logique
~	NON logique
xor	OU EXCLUSIF logique

Fonctions logiques	
all	détermine si tous les éléments de la matrice considérée sont non nuls
any	détermine si l'un des éléments de la matrice considérée est non nul
exist	détermine si la variable, la fonction ou le fichier considéré existe
find	détermine les indices et la valeur des éléments non nuls d'un tableau ou d'une matrice
is*	détermine un état
isa	détermine un objet d'une classe donnée
logical	convertit une valeur numérique en valeur logique

Fonctions binaires	
`bitand`	ET logique bit à bit
`bitcmd`	complément bit à bit
`bitor`	OU logique bit à bit
`bitmax`	plus grand entier non signé représentable sur la machine
`bitset`	mise à 1 d'un bit à un emplacement donné
`bitshift`	décalage (à droite ou à gauche)
`bitget`	valeur d'un bit à une position donnée
`bitxor`	OU EXCLUSIF bit à bit

B.3 Langage de programmation

Utilisation de fonctions	
`builtin`	lance l'exécution d'une commande du noyau (fonction *built-in*)
`eval`	interprète et évalue la chaîne de caractères (contenant des commandes, des opérations et des noms de variables) passée en argument
`feval`	lance l'évaluation d'une fonction (dont le nom et les arguments sont passés en arguments)
`function`	définit une fonction dans un fichier `M`
`global`	permet de déclarer une variable globale
`nargchk`	vérifie le nombre d'arguments passés à une fonction

Programmation	
`break`	interrompt l'exécution de la structure de contrôle courante
`case`	séléction multiple (se présente après un `switch`; voir aussi `otherwise`)
`else`	aiguillage (se présente après un `if`)
`elseif`	aiguillage multiple (se présente après un `if`)
`end`	sert à délimiter la fin d'une instruction `for`, `if`, `switch` ou `while`
`error`	provoque l'affichage du message d'erreur passé en argument et l'interruption du programme
`for`	boucle inconditionnelle
`if`	branchement conditionnel (voir aussi `else` et `elseif`)
`otherwise`	introduit le traitement par défaut d'un `switch` (voir aussi `case`)
`return`	retour à la fonction appelante
`switch`	sélection multiple (voir aussi `case` et `otherwise`
`warning`	provoque l'affichage du message d'alerte passé en argument
`while`	boucle conditionnelle

Saisie interactive	
`input`	interrompt l'exécution en attente d'une saisie de l'utilisateur
`keyboard`	interrompt l'exécution et donne le contrôle au clavier
`menu`	génère un menu de choix à destination de l'utilisateur
`pause`	interrompt temporairement l'exécution

Mise au point (débogage)	
`dbclear`	supprime les points d'arrêts
`dbcont`	relance l'exécution du fichier `M` jusqu'au prochain point d'arrêt
`dbdown`	change le contexte de la fonction en cours d'exécution en celui de l'espace de travail
`dbmex`	active le debogage des fichiers `MEX`
`dbquit`	termine le débogage et rend le contrôle à l'interpréteur
`dbstack`	affiche le nom de la fonction en cours d'exécution, le numéro de la ligne pù est fixé le point d'arrêt courant, le nom du fichier `M` appelant cette fonction, le numéro de ligne de l'appel, etc.
`dbstatus`	donne la liste des numéros de lignes où sont placés les points d'arrêt
`dbstep`	exécute une ou plusieurs lignes de programme en mode mise au point
`dbstop`	place un point d'arrêt dans un fichier `M`
`dbtype`	liste un fichier `M` avec des numéros de ligne
`dbup`	change le contexte de l'espace de travail de base en celui de la fonction en cours d'exécution

Constantes et variables spéciales	
`ans`	dernier résultat calculé
`computer`	identifie le type d'ordinateur sur lequel est exécuté Matlab
`eps`	précision relative des calculs menés en virgule flottante
`flops`	compte le nombre d'opérations élémentaires
`i`	unité imaginaire
`Inf`	infini
`inputname`	nom de l'argument en entrée
`j`	unité imaginaire
`NaN`	désigne un résultat non numérique (*not a number*)
`nargin`	nombre d'arguments passés en entrée à une fonction
`nargout`	nombre d'arguments fournis en sortie par une fonction
`pi`	designe la constante trigonométrique π
`realmax`	plus grand nombre flottant positif pouvant être représenté en machine
`realmin`	plus petit nombre flottant négatif pouvant être représenté en machine
`varargin`	liste d'arguments d'entrée contenant les arguments optionnels d'appel d'une fonction
`varargout`	liste d'arguments de retour contenant les arguments optionnels de retour d'une fonction

Date et Heure	
`calendar`	retourne une matrice représentatn le calendrier du mois spécifié
`clock`	retourne la date et l'heure courantes sous forme d'un vecteur `[année mois jour heures minutes secondes]`
`cputime`	calcule le temps écoulé entre deux dates
`date`	retourne la date courante
`datenum`	convertit la date sous forme de chaîne en un nombre
`datestr`	convertit la date sous forme d'un nombre en une chaîne
`datevec`	sépare les composantes d'une date, donnée sous forme d'une chaîne ou d'un nombre, en un vecteur `[année mois jour heures minutes secondes]`
`eomday`	détermine le jour de fin d'un mois
`etime`	calcule le temps (en secondes) écoulé entre deux dates données sous forme de vecteur `[année mois jour heures minutes secondes]`
`now`	retourne la date et l'heure courantes sous forme d'un nombre
`tic`	déclenche la fonction de chronométrage
`toc`	arrête la fonction de chronométrage
`weekday`	retourne le jour de la semaine, sous forme d'un nombre ou d'une chaîne, d'une date donnée

B.4 Matrices particulières et opérations sur les matrices

Matrices et vecteurs remarquables	
`eye`	matrice identité
`linspace`	génère un vecteur ligne de valeurs également espacées
`logspace`	génère un vecteur ligne de valeurs logarithmiquement espacées
`ones`	matrices composées de 1
`rand`	matrice (ou nombre) généré aléatoirement selon une distribution uniforme
`randn`	matrice (ou nombre) généré aléatoirement selon une distribution normale
`zeros`	matrice nulle

Manipulations sur les matrices	
`cat`	permet la concaténation de tableaux
`diag`	permet la définition de matrices diagonales ou l'extraction de la diagonale d'une matrice
`fliplr`	permutation circulaire des colonnes
`flipud`	permutation circulaire des lignes
`repmat`	permet la réplication d'une valeur dans une matrice
`reshape`	permet de transformer et redimenssionner une matrice
`rot90`	transformation équivalente à l'application successive de la transposition et de `flipud`
`tril`	extrait le triangle inférireur d'une matrice
`triu`	extrait le triangle supérieur d'une matrice

Matrices spécialisée	
`compan`	matrice compagnon
`gallery`	matrice de tests
`hadamard`	matrice de Hadamard
`hankel`	matrice de Hankel
`hilb`	matrice de Hilbert
`invhilb`	inverse d'une matrice de Hilbert
`magic`	carré magique
`pascal`	matrice de Pascal
`toeplitz`	matrice de Toeplitz
`wilkinson`	matrice de Wilkinson

B.5 Fonctions mathématiques usuelles

Fonctions mathématiques usuelles	
`abs`	valeur absolue d'un réel ou module d'un complexe
`acos`, `asin`, `atan`, `acot`	fonctions circulaires réciproques
`acosh`, `asinh`, `atanh`, `acoth`	fonctions hyperboliques réciproques
`acsc`, `asec`	fonctions réciproques de la cosécante et de la séquante circulaires
`acsch`, `asech`	fonctions réciproques de la cosécante et de la séquante hyperboliques
`angle`	argumant d'un nombre complexe
`ceil`	plus petit entier supérieur au nombre flottant considéré
`cos`, `sin`, `tan`, `cot`	fonctions circulaires directes
`cosh`, `sinh`, `tanh`, `coth`	fonctions hyperboliques directes
`csc`, `sec`	fonctions directes de la cosécante et de la séquante circulaires
`csch`, `sech`	fonctions directes de la cosécante et de la séquante hyperboliques
`exp`	fonction exponentielle néperienne
`fix`	arrondi entier vers 0
`floor`	plus grand entier inférieur zau nombre flottant considéré (partie entière)
`gcd`	plus grand commun diviseur
`imag`	partie imaginaire d'un nombre complexe
`lcm`	plus petit commun multiple
`log`	fonction logarithme népérien
`log2`	fonction logarithme en base 2
`log10`	fonction logarithme en base 10
`mod`	reste (signé) d'une division euclidienne (fonction mudulo)
`real`	partie réelle d'un nombre complexe
`rem`	reste d'une division euclidienne
`round`	arrondi au plus proche entier
`sign`	signe
`sqrt`	racine carrée

B.6 Fonctions mathématiques spécialisées

Fonctions mathématiques spécialisées	
airy	fonctions d'Airy
besselh	fonctions de Bessel de troisième espèce (fonction de Hankel)
besseli, besselk	fonctions de Bessel de première et deuxième espèces modifiées
besselj, bessely	fonctions de Bessel de première et deuxième espèces
beta, betainc, betaln	fonctions Bêta
ellipj	fonction elliptiques (dites de Jacobi)
ellipke	intégrales elliptiques complètes de première et deuxième espèces
erf, erfc, erfcx, erfinv	fonctions d'erreur
expint	fonction exponentielle intégrale
gamma, gammainc, gammaln	fonctions gamma
legendre	fonctions de Legendre associées
pow2	puissances entières de 2
rat, rats	approximation rationnelle

Conversion s entre systèmes de coordonnées	
cart2pol	transforme des coordonnées cartésiennes en coordonnées polaires
cart2sph	transforme des coordonnées cartésiennes en coordonnées sphériques
pol2cart	transforme des coordonnées polaires en coordonnées cartésiennes
sph2cart	transforme des coordonnées sphériques en coordonnées cartésiennes

B.7 Manipulation de matrices - Algèbre linéaire

Analyse matricielle	
cond	Conditionnement (pour le calcul de l'inverse)
condeig	Conditionnement (pour le calcul des valeurs propres)
det	déterminant
norm	normes matricielles et vectorielles
null	noyau de l'application linéaire associée
orth	image de l'application linéaire associée
rank	rang
rcond	estimation de l'inverse du conditionnement
rref, rrefmovie	réduite échelonnée par ligne
subspace	angle entre deux sous espaces vectoriels
trace	trace (some des éléments diagonaux

Systèmes linéaires	
`chol`	factorisation de choleski
`inv`	inversion matricielles
`lscov`	résolution au sens des moindres carrés (la covariance étant connue)
`lu`	factorisation LU
`nnls`	méthodes des moindres carrés
`pinv`	pseudo-inverse de Moore-Penrose
`qr`	factorisation QR

Valeurs propres et valeurs singulières	
`balance`	améliore la précision pour le calcul des valeurs propres
`cdf2rdf`	conversion d'une matrice diagonle complexe en la matrice réelle digonale par blocs correspondante
`eig`	calcul des valeurs propres
`hess`	forme de Hessenberg associée à une matrice
`poly`	polynôme caractéristique
`qz`	factorisation QZ
`rsf2csf`	conversion d'une matrice réelle diagonale par blocs en la matrice diagonale complexe correspondante
`schur`	décomposition de Schur
`svd`	décomposition en valeurs singulières

Fonctions matricielles	
`expm`	exponentielle matricielle
`funm`	évaluation de fonctions matricielles
`logm`	logarithme matriciel
`sqrtm`	racine carrée matricielle

Fonctions matricielles de bas niveau	
`qrdelete`	efface une colonne d'une factorisation QR
`qrinsert`	insert une colonne dans une factorisation QR

Fonctions vectorielles	
`cross`	produit vectoriel
`intersect`	détermine les éléments communs à deux vecteurs
`ismember`	détermine si un nombre apparaît dans un vecteur
`setdiff`	effectue la différence ensembliste entre les éléments de deux vecteurs
`setxor`	effectue l'opération OU-EXCLUSIF sur les éléments de deux vecteurs
`union`	établit l'union ensembliste des éléments de deux vecteurs
`unique`	détermine les éléments uniques d'un vecteur

B.8 Analyse de données

Fonctions élémentaires	
`convhull`	enveloppe convexe
`cumprod`	produit cumulé
`cumsum`	somme cumulée
`cumtrapz`	somme partielles au cours d'une intégration numérique par la méthode des trapèzes
`delaunay`	Triangulation de Delaunay
`dsearch`	recherche du point le plus proche
`factor`	décomposition en produit de facteurs premiers
`max`	élément maximal d'un tableau
`mean`	moyenne des valeurs d'un tableau
`median`	valeur médiane d'un tableau
`min`	élément minimal d'un tableau
`perms`	liste toutes les permutations possibles
`polyarea`	Aire d'un polygone
`primes`	génère la liste des nombres premiers
`prod`	effectue le produit des éléments d'un tableau
`sort`	trie les éléments d'un tableau
`sortrows`	trie les lignes d'une matrice par ordre croissant
`std`	ecart type
`sum`	effectue la somme des éléments d'un tableau
`trapz`	intégration numérique selon la méthode des trapèzes
`tsearch`	recherche d'un triangle de Delaunay englobant le point considéré
`voronoi`	Diagramme de Voronoi

Différences finies	
`del2`	Laplacien discret
`diff`	dérivation discrète (différences finies)
`gradient`	gradiant numérique

Corrélation	
`corrcoef`	coefficient de corrélation
`cov`	matrice de covariance

Filtrage et convolution	
`conv`	convolution et produit de polynômes
`conv2`	convolution bidimensionnelle
`deconv`	déconvolution et division de polynôme
`filter`	filtrage numérique avec filtre de type RIF (réponse impulsionnelle finie) et avec filtres de type RII (réponse impulsionnelle infinie)
`filter2`	filtrage numérique bidimensionnel

Transformation de Fourier	
`abs`	module d'un nombre complexe
`angle`	argument d'un nombre complexe
`cplxpair`	associe des nombres complexes par paires de nombres deux à deux conjugués
`fft`	transformée de Fourier rapide monodimensionnelle
`fft2`	transformée de Fourier rapide bidimensionnelle
`fftshift`	permet le centrage du spectre d'une transformée de Fourier rapide
`ifft`	transformée de Fourier inverse monodimensionnelle
`ifft2`	transformée de Fourier inverse bidimensionnelle
`nextpow2`	détermine la puisance de 2 immédiatement supérieure au nombre passé en argument
`unwrap`	corrige les angles des phases

B.9 Polynômes et interpolation

Polynômes	
`conv`	convolution et produit de polynômes
`deconv`	déconvolution et division de polynôme
`poly`	détermine un polynôme à partir de ses racines
`polyder`	dérivation polynomiale
`polyeig`	problème polynomial de valeurs propres
`polyfit`	ajustement polynomial au sens des moindres carrés
`polyval`	évaluation d'un polynôme en un point
`polyvalm`	évaluation d'un polynôme de matrice
`residue`	décomposition en élément simples
`roots`	calcul des racines d'un polynômes

Interpolation	
`interp1`	interpolation de données monodimensionnelles
`interp2`	interpolation de données bidimensionnelles
`interp3`	interpolation de données tridimensionnelles
`interpft`	interpolation de données dans le domaine fréquentiel fondée sur une transformée de Fourier rapide
`interpn`	interpolation de données multidimensionnelles
`meshgrid`	génère des matrices pour l'établissements d'un graphique en 3 dimensions
`ndgrid`	génère les tableaux pour une interpolation multidimensionnelle
`spline`	interpolation par des splines cubiques

B.10 Intégration numérique

Inrégartion numérique	
`dblquad`	intégration numérique d'une intégrale double
`fmin`	minimisation d'une fonction d'une variable
`fmins`	minimisation d'une fonction de plusieurs variables
`fzero`	recherche numérique des zéros d'une fonction d'une variable
`ode45`, `ode23`, `ode113`, `ode15s`, `ode23s`	solveurs (schémas d'intégration numérique de Runge-Kutta, Adams-Bashforth-Moulton, ...) d'équation différentielle
`odefile`	permet de définir une équation différentielle en vue d'un résolution numérique
`odeget`	permet de connaître les paramètres d'une résolution numérique d'équation différentielles
`odeset`	permet de modifier les paramètres d'une résolution numérique d'équation différentielles
`quad`, `quad8`	intégration numérique d'une intégrale
`vectorize`	vectorisation d'une expression

B.11 Fonctions permettant de traiter le son

Fonctions permettant de traiter le son	
`auread`	lecture de fichiers son au format Next/SUN (fichier `.au`)
`auwrite`	écriture de fichiers son au format Next/SUN (fichier `.au`)
`readsnd`	lecture de fichiers et de ressources son
`recordsound`	enregistrement du son
`sound`	conversion des vecteurs en son
`soundcap`	détermine les capacités de traitement du son
`speak`	prononce une chaine de caractères
`wavread`	lecture de fichiers son au format Microsoft WAVE (fichier `.wav`)
`wavwrite`	écriture de fichiers son au format Microsoft WAVE (fichier `.wav`)
`writesnd`	écriture de fichiers et de ressources son

B.12 Représentations graphiques

Représentation graphique en dimension 2	
`plot`	graphique en coordonnées linéaires
`loglog`	graphique en coordonnées logarithmiques
`polar`	graphique en coordonnées polaires
`semilogx`, `semilogy`	graphique en coordonnées semilogarithmique

Représentation graphique en dimension 3	
`fill`	représentation à base de polygones
`mesh`	représentatino d'une nappe par maillage
`plot3`	représentation de segments et de points en dimension 3
`surf`	représentation d'une nappe par une surface

Gestion des axes	
`axes`	création d'axes en positions arbitraires
`axis`	contrôle de l'apparence et de l'échelle des axes
`box`	encadre le graphique avec des axes
`grid`	crée un quadrillage superposé au graphique
`hold`	gèle la figure graphique courante
`subplot`	juxtapose des graphiques dans la même figure
`zoom`	agrandit ou diminue une figure

Annotation d'une figure	
`gtext`	permet de placer du texte sur une figure à l'aide de la souris
`legend`	ajoute une légende à la figure
`text`	place du texte sur la figure
`title`	place un titre sur la figure
`xlabel`, `ylabel`, `zlabel`	place une étiquette sur l'axe correspondant

Impression	
`print`	impression ou copie dans un fichier `M` d'un graphique
`printopt`	options d'impression
`orient`	orientation du papier

B.13 Traitement des chaînes de caractères

Manipulation de chaînes de caractères	
`deblank`	enlève tous les caractères blancs susceptibles de se trouver en fin d'une chaîne
`eval`	provoque l'interprétation et l'évaluation du contenu d'une chaîne
`findstr`	recherche une chaîne dans une chaîne
`lower`	convertit en minuscule tous les caractères d'une chaîne
`strcat`	concatène des chaînes
`strcmp`	compare deux chaînes
`strjust`	justifie un tableau de caractères
`strmach`	filtre une chaîne
`strncmp`	compare le n-ième caractère de deux chaînes
`strrep`	cherche et remplace un motif dans une chaîne
`strtok`	recherche une clé dans une chaîne
`strvcat`	concatète des chaînes verticalement
`upper`	convertit en majuscules tous les caractères d'une chaîne

Conversion entre chaînes et nombres	
`char`	crée un tableau de caractères (chaîne)
`int2str`	convertit un nombre entier en chaîne de caractères
`mat2str`	convertit une matrice en chaîne de caractères
`sprintf`	écrit des données formatées dans une chaîne
`sscanf`	lit une chaîne formatée
`str2num`	convertit un chaîne de caractères en une matrice de nombres

Autres fonctions de conversion	
`bin2dec`	convertit un nombre binaire en nombre entier
`dec2bin`	convertit un nombre entier en nombre binaire
`dec2hex`	convertit un nombre entier en un nombre hexadécimal (écrit sous forme de chaîne de caractère)
`hex2dec`	convertit un nombre hexadécimal (écrit sous forme de chaîne de caractère) en un nombre entier
`hex2num`	convertit un nombre hexadécimal (écrit sous forme de chaîne de caractère) en un nombre flottant double précision

B.14 Fonction d'entrées/sortie

Ouverture et fermeture de fichiers	
`fclose`	ferme un ou plusierus fichiers
`fopen`	ouvre un fichier; permet d'obtenir des renseignements sur les fichiers ouverts

Entrées/sortie non formatées	
`fread`	permet la lecture dans un fichier binaire
`fwrite`	permet l'écriture dans un fichier binaire

Entrées/sortie formatées	
`fgetl`	retourne la prochaine ligne du fichier lu sous forme d'une chaîne de caractères, le caractère de retour à la ligne n'étant pas inclus dans la chaîne
`fgets`	retourne la prochaine ligne du fichier lu sous forme d'une chaîne de caractères, le caractère de retour à la ligne étant inclus dans la chaîne
`fprintf`	permet l'écriture formatée dans un fichier
`fscanf`	permet la lecture formatée dans un fichier
`feof`	permet la détection de la fin d'un fichier
`ferror`	retourne le message et le numéro d'erreur de la dernière opération d'entrées/sorties
`frewind`	positionne le pointeur du fichier donné sur le premier octet
`fseek`	deplace le pointeur du fichier donné pour le positionner à l'endroit indiqué
`ftell`	indique la position du pointeur du fichier donné

Conversion de chaînes	
`sscanf`	permet l'extraction d'informations (sous un format particulier) d'une chaîne de caractères
`sprintf`	permet l'envoi de sorties formatées dans une chaîne de caractères

Fonction d'entrées/sortie spécialisées	
`dlmread`	lecture d'un fichier ASCII délimité dans une matrice
`dlmwrite`	écriture d'une matrice dans un fichier ASCII délimité
`imfinfo`	information sur l'image stockée dans un fichier graphique
`imread`	lecture d'une image dans un fichier graphique
`imfwrite`	écriture d'une image dans un fichier graphique
`qtmwrite`	écriture d'un fichier QuickTime
`wk1read`	lecture d'une feuille de calcul au format Lotus dans une matrice
`wk1write`	écriture d'une matrice dans une feuille de calcul au format Lotus
`xlgetrange`	récupère le contenu de cellules d'une feuille de calcul Excel
`xlsetrange`	établit le contenu de cellules d'une feuille de calcul Excel

B.15 Types et structures de données

Conversion de chaînes	
`double`	convertit en un nombre flottant double précision
`sparse`	crée une matrice creuse
`char`	crée un tableau de caractères (chaîne)
`cell`	crée un tableau de cellules
`struct`	crée (ou convertit) en tableau de structures
`uint8`	convertit en un entier non signé codé sur 8 bits

Fonctions sur les tableaux	
`cat`	concatène deux tableaux
`flipdim`	effectue une permutation circulaire sur un tableau selon une dimension spécifiée
`ipermute`	inverse la fonction `permute`
`ndgrid`	génère des tableaux pour des représentations graphiques
`ndims`	nombre de dimensions d'un tableau
`permute`	échange des dimensions d'un tableau
`shiftdim`	décale les dimensions d'un tableau
`squeeze`	compacte un tableau en supprimant les dimensions réduites à un élément

Fonctions sur les tableaux de cellules	
`cell`	crée un tableau de cellules
`cellstr`	crée un tableau de cellules de chaînes à partir d'un tableau de caractères
`cellstruct`	convertit un tableau de cellules en un tableau de structures
`celldisp`	affiche le contenu des cellules d'un tableau
`num2cell`	convertit un tableau de nombres en un tableau de cellules
`cellplot`	représente graphiquement la structure d'un tableau de cellules

Fonctions sur les structures (enregistrements)	
`fieldnames`	noms des champs d'une structures
`getfield`	accède au contenu d'un champ d'une structure
`rmfield`	supprime des champs d'une structure
`setfield`	établit la valeur d'un champ d'une structure
`struct`	crée une structure (enregistrements)
`struct2cell`	transforme un tableau de structures en un tableau de cellules