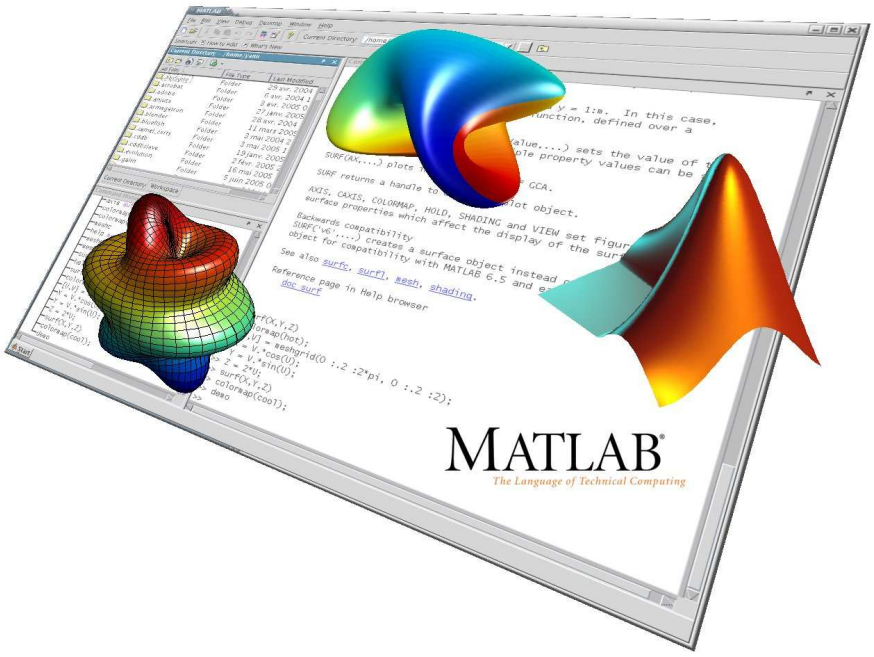
paul verlaine
université - metz
ufr sciences fondamentales
et appliquees

Initiation au progiciel Matlab

Licence Mécanique-Electronique

Année Universitaire 2005/2006



Y. Morère

Cette page est laissée blanche intentionnellement

Table des matières

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 :

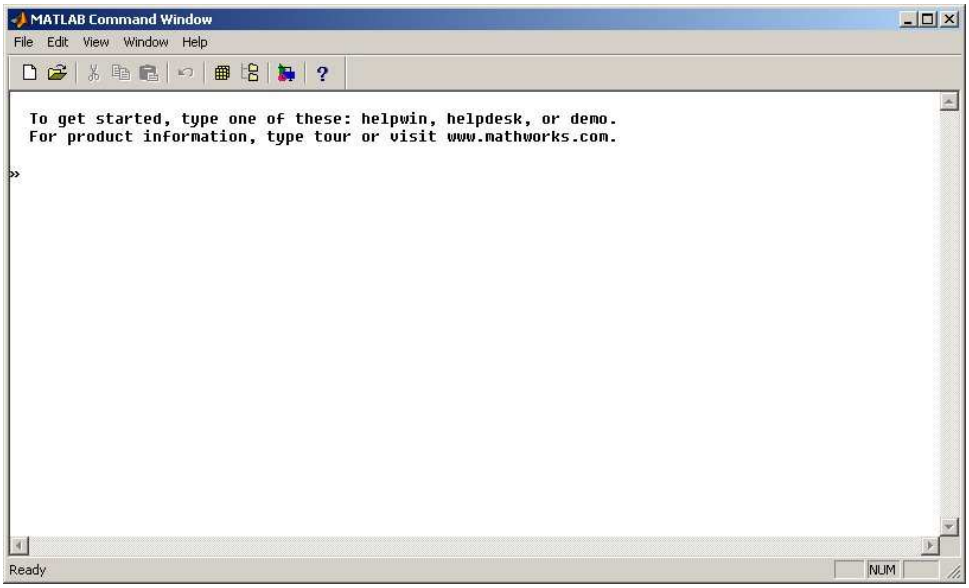


FIG. 1.1 – Fenêtre de commande Matlab

– 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 :

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

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')
```

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)
```

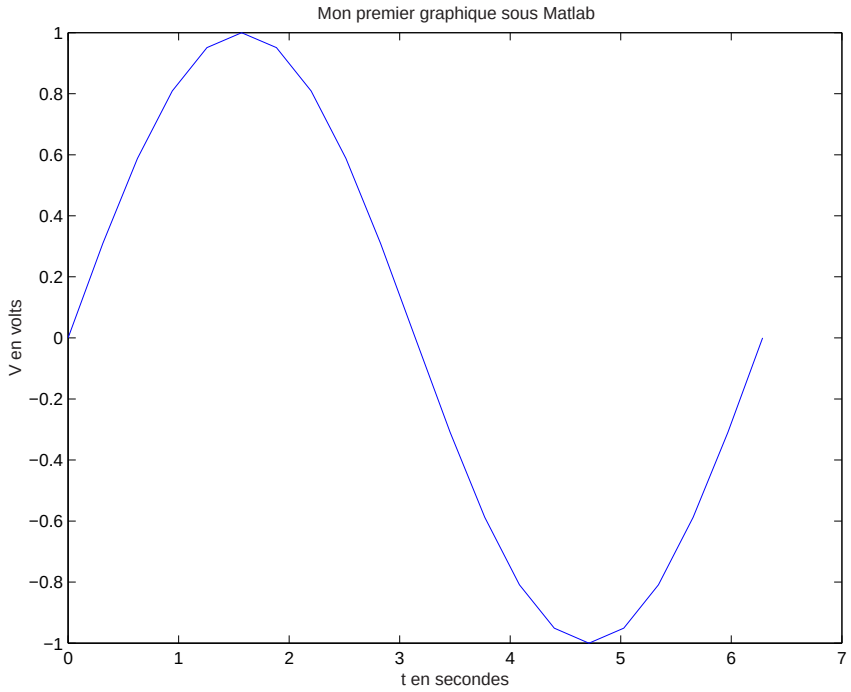


FIG. 1.2 – Premier graphique avec Matlab

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

Annexe A : Généralités sur Matlab

A.1 L'environnement Matlab

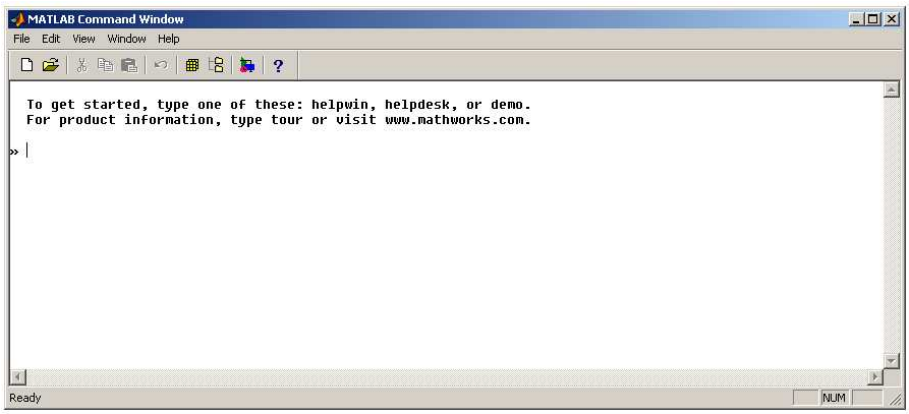


FIG. A.1 – Fenêtre de Matlab

Matlab (MatrixLaboratory) est un environnement de calcul numérique matriciel. Après le lancement de Matlab, une fenêtre de commande apparaît qui permet à l'utilisateur de taper une commande quelconque obéissant à la syntaxe de Matlab.
Le symbole `>>` apparaissant à gauche de la fenêtre de Matlab indique que l'interpréteur est prêt à recevoir une commande. Voici un exemple de session Matlab :

```
>> A = [ 1 3; 4 2 ]
A =
1 3
4 2
>> A*A
ans =
13 9
12 16
>> quit
 16 flops.
```

Dans cette session on a défini la matrice $A = \begin{pmatrix} 1 & 3 \\ 4 & 2 \end{pmatrix}$ et l'on a calculé son carré. Chaque ligne d'instructions doit se terminer par un retour chariot (touche Entrée). La commande pour quitter Matlab est `quit`.

A.1.1 Espace de travail

Comme tout langage de programmation, Matlab permet de définir des données variables. Les variables sont définies au fur et à mesure que l'on donne leurs noms (identificateur) et leurs

valeurs numériques ou leurs expressions mathématiques. Matlab ne nécessite pas de déclaration de type ou de dimension pour une variable. Les variables sont stockées dans l'espace de travail (ou workspace) et peuvent être utilisées dans les calculs subséquents. Pour obtenir la liste des variables actives de l'espace de travail on dispose des commandes who et whos. La commande who affiche le nom des variables actives. La commande whos donne plus d'informations : le nom, la taille du tableau (nombre de lignes et de colonnes) associé, l'espace mémoire utilisé (en Bytes) et la classe des données (principalement double array s'il s'agit d'un tableau de valeurs réelles ou complexes et char s'il s'agit d'un tableau de caractères). La commande clear permet de nettoyer l'espace de travail : toutes les variables sont détruites. Il est possible de ne détruire qu'une partie des variables en tapant clear nom-var où nom-var est le nom de la (ou des) variable(s) à détruire.

```
>> x=2*pi/3; y=sin(x); z=cos(x);
>> A = [ 1 3; 4 2 ]; B = A*A;
>> t = 'bonjour';
>> who
Your variables are :
A         B         t         x         y         z
>> whos
  Name      Size         Bytes  Class
  A         2x2             32  double array
  B         2x2             32  double array
  t         1x7             14  char array
  x         1x1              8  double array
  y         1x1              8  double array
  z         1x1              8  double array
Grand total is 18 elements using 102 bytes
>> clear x y t
>> whos
  Name      Size         Bytes  Class
  A         2x2             32  double array
  B         2x2             32  double array
  z         1x1              8  double array
Grand total is 9 elements using 72 bytes
>> clear
>> who
>>
```

Il est possible de sauvegarder une session Matlab dans un fichier pour une utilisation ultérieure. L'instruction save nom-fic enregistre toutes les variables de l'espace de travail dans le fichier `nom-fic.mat`. Si aucun nom de fichier n'est précisé, le fichier par défaut est `Matlab.mat`. Il est possible de ne sauver qu'une partie des variables (par exemple seulement la variable contenant le résultat d'un calcul) en utilisant l'instruction `save nom-fic nom-var` où `nom-var` est le nom de la (ou des) variable(s) à sauvegarder. Attention, seul le contenu des variables est sauvegardé et non pas l'ensemble des instructions effectuées durant la session. Pour ramener dans l'espace de travail les variables sauvegardées dans le fichier `nom-fic.mat`, taper `load nom-fic`. Dans l'exemple qui suit on calcule le sinus et le cosinus de *2 pi/3* dans les variables y et z. On sauve ces résultats dans un fichier `toto.mat`. Après avoir quitté Matlab on peut vérifier que le fichier `toto.mat` existe bien mais qu'il ne s'agit pas d'un fichier ascii. Si l'on relance Matlab et que l'on tape `load toto`, on peut vérifier que les variables y et z existent bien et ont les valeurs précédentes.

```
>> x=2*pi/3, y=sin(x), z=cos(x)
x =
    2.0944
y =
    0.8660
z =
   -0.5000
>> save toto y z
>> quit
 6 flops.
```

Lancer Matlab:

```
>> load toto
>> who
Your variables are :
 y         z
>> y
y =
    0.8660
>> z
z =
   -0.5000
>> x
??? Undefined function or variable 'x'.
>>
```

La commande diary permet de sauvegarder l'ensemble d'une session dans un fichier ascii pour un traitement ultérieur (insertion dans un document, impression, ...). Le fichier de sauvegarde par défaut a pour nom `diary`.
On provoque la sauvegarde dans le fichier `nom-fic`, par l'instruction `diarynom-fic`. Attention, c'est la commande `diary` qui déclenche le début de sauvegarde; il est impossible de sauvegarder la partie de la session précédent son appel. Pour arrêter la sauvegarde, taper `diary off`.

```
>>  x=2*pi/3; y=sin(x);
>> diary toto
>>  z=cos(x)
z =
   -0.5000
>> diary off
>> t = tan(x)
t =
   -1.7321
>>
```

Si vous éditez le fichier `toto`, voici ce que vous verrez :

```
>>  z=cos(x)
z =
   -0.5000
>> diary off
```

A.1.2 Obtenir de l'aide

Dans une session Matlab, il est possible d'obtenir une aide en ligne sur une commande en tapant `help nom-commande`. Par exemple,

```
>> help diary
 DIARY Save text of MATLAB session.
    DIARY file{\_}name  causes a copy of all subsequent terminal input
    and most of the resulting output to be written on the named
    file. DIARY OFF suspends it. DIARY ON turns it back on.
    DIARY, by itself, toggles the diary state.
    Use the functional form of DIARY, such as DIARY('file'),
    when the file name is stored in a string.
>>
```

Attention, les commandes Matlab doivent être tapées en *minuscules* pour être reconnues, même si elles figurent en *majuscules* dans l'aide en ligne. On peut également obtenir de l'aide par le biais de la commande `doc` qui donne accès à une documentation complète au format HTML. Pour quitter cette documentation, cliquer sur `Exit Program` dans le menu `File` du navigateur. La commande `lookfor` permet de rechercher un mot-clé parmi les lignes de commentaires en entête des fonctions Matlab (ces lignes sont celles affichées par la commande `doc`). L'instruction `lookfor` motclé recherche le mot-clé motclé dans la première ligne de commentaire de toutes les fonctions Matlab. L'instruction `lookfor motclé -all` recherche le `mot-clé` dans toutes les lignes de commentaires en entête des fonctions. Attention le `mot-clé` doit être en anglais, les commentaires des fonctions Matlab étant rédigés en anglais. On peut utiliser une phrase comme `mot-clé`. Il faut alors l'écrire entre guillemets (`' '`). La commande `lookfor` retourne le nom la fonction Matlab (ou des fonctions) où le mot-clé figure dans la première ligne de commentaires. Elle retourne également la ligne de commentaires où figure le mot-clé. Si le mot-clé n'a été trouvé dans aucune ligne de commentaires, Matlab rend la main sans rien afficher.

```
>> lookfor determinant
DET    Determinant.
>> lookfor 'tridiagonal matrix'
LESP   Tridiagonal matrix with real, sensitive eigenvalues.
POISSON Block tridiagonal matrix from Poisson's equation.
TRIDIAG Tridiagonal matrix (sparse).
TRIDIEIG  Find a few eigenvalues of a tridiagonal matrix.
TRIDISOLVE  Solve A*x = b where A is a square, symmetric tridiagonal
matrix.
>> lookfor papillon
>>
```

A.1.3 Syntaxe d'une ligne d'instructions

Si une instruction Matlab est suivie d'un point virgule, le résultat de cette instruction n'est pas affiché. Pour réafficher un résultat contenu dans une variable, il suffit de taper le nom de la variable. Le résultat de la dernière instruction exécutée peut être rappelé par la commande `ans` :

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

```
     8     1     6
     3     5     7
     4     9     2
>> A*A;
>> ans
ans =
    91    67    67
    67    91    67
    67    67    91
>>
```

Plusieurs instructions Matlab peuvent figurer sur une même ligne. Il faut alors les séparer par une virgule ou par un point virgule. D'autre part, si une commande est trop longue pour tenir sur une ligne, il est possible de poursuivre sur la ligne suivante en terminant la ligne par 3 points (...).

```
>> B = [ 1 3; 4 2 ]; B*B
ans =
    13     9
    12    16
>> x = 1 + 2 + 3 + 4 + 5 + 6 ...
+7 + 8 + 9 + 10
 =
    55
>>
```

Si la syntaxe de l'instruction soumise est erronée ou si vous demandez à Matlab d'exécuter une instruction illégale (qui n'a pas de sens mathématique par exemple), vous obtiendrez un message d'erreur. Ce message vous indique les sources d'erreurs possibles et doit vous permettre de corriger rapidement votre erreur.

```
>> A + B
??? Error using ==> +
Matrix dimensions must agree.
>> C = [ 1 2 3; 4 5]
??? Number of elements in each row must be the same.
>> whose
??? Undefined function or variable 'whose'.
>>
```

Dans la première instruction, on tente d'effectuer la somme de deux matrices aux dimensions incompatibles. Dans le second exemple on tente de définir une matrice dont le nombre d'éléments dans chaque ligne diffère. Enfin la troisième instruction est inconnue de Matlab : il ne s'agit n'y d'une fonction ni d'une variable incorporée ou utilisateur.

A.1.4 Gestion des fichiers du répertoire de travail

Un certain nombre de commandes permettent de gérer les fichiers du répertoire de travail. La commande `dir` donne la liste des fichiers du répertoire de travail. La commande `cd` permet de changer de répertoire de travail. La commande `type` permet d'afficher le contenu d'un fichier. La commande `delete` permet de détruire un fichier. Ces commandes s'utilisent de la même

manière que les commandes correspondantes du DOS. Enfin la commande edit permet d'ouvrir un éditeur de texte. Le choix de l'éditeur a été effectué au moment de l'installation de Matlab sur votre machine, il est possible de changer votre éditeur favori dans les Preferences de Matlab. Il est également possible d'exécuter des commandes DOS à partir de Matlab en faisant précéder la commande d'un point d'exclamation (!).
Si vous avez effectué les exemples du paragraphe précédent, deux fichiers `toto` et `toto.mat` existent dans votre répertoire de travail. Voici alors un exemple d'utilisation des commandes de gestion des fichiers.

```
>> dir
 .       ..      toto    toto.mat
>> edit toto
>> !notepad toto
>>
```

A.2 Types de données et variables

Comme tout langage de programmation, Matlab permet de définir des données variables. Une variable est désignée par un identificateur qui est formé d'une combinaison de lettres et de chiffres. Le premier caractère de l'identificateur doit nécessairement être une lettre. Attention, Matlab différencie majuscules et minuscules! Ainsi `X33` et `x33` désignent deux variables distinctes. Les variables sont définies au fur et à mesure que l'on donne leurs noms (identificateur) et leurs valeurs numériques ou leurs expressions mathématiques. L'utilisation de variables avec Matlab ne nécessite pas de déclaration de type ou de dimension. Le type et la dimension d'une variable sont déterminés de manière automatique à partir de l'expression mathématique ou de la valeur affectée à la variable. Une variable peut être de type réel, complexe, chaîne de caractères ou logique. Pour Matlab toute variable est considérée comme étant un tableau d'éléments d'un type donné. Matlab différencie trois formes particulières de tableaux. Les scalaires qui sont des tableaux à une ligne et une colonne. Les vecteurs qui sont des tableaux à une ligne ou à une colonne. Les matrices qui sont des tableaux ayant plusieurs lignes et colonnes. Une variable Matlab est donc toujours un tableau que l'on appelle variable scalaire, vecteur ou matrice suivant la forme du tableau.

A.2.1 Les types de données

A.2.1.1 Les 4 types de données Matlab

Les trois principaux types de variables utilisés par Matlab sont les types réel, complexe et chaîne de caractères. Il n'y a pas de type entier à proprement parler. Le type logique est associé au résultat de certaines fonctions. Signalons qu'il est inutile (impossible) de déclarer le type d'une variable. Ce type est établi automatiquement à partir des valeurs affectées à la variable.
Par exemple les instructions `x = 2; z = 2+i; rep = 'oui';` définissent une variable x de type réel, une variable z de type complexe et une variable rep de type chaîne de caractères.

```
>> clear
>> x = 2; z = 2+i; rep = 'oui';
>> whos
  Name      Size         Bytes  Class
  rep       1x3              6  char array
  x         1x1              8  double array
  z         1x1             16  double array (complex)
```

```
Grand total is 5 elements using 30 bytes
>>
```

Comme on ne définit pas de manière explicite le type d'une variable, il est parfois utile de pouvoir le déterminer. Cela est possible grâce aux commandes `ischar`, `islogical` et `isreal`. `ischar(x)` retourne 1 si x est de type chaîne de caractères et 0 sinon. `islogical(x)` retourne 1 si x est de type logique et 0 sinon. La commande `isreal(x)` est à utiliser avec discernement : elle retourne 1 si x est réel ou de type chaîne de caractères et 0 sinon (x est complexe à partie imaginaire non nulle ou n'est pas un tableau de valeurs réelles ou de caractères).

```
>> ischar(rep)
ans =
     1
>> ischar(x)
ans =
    0
>> isreal(z)
ans =
     0
>> isreal(x)
ans =
     1
>> isreal(rep)
ans =
     1
>>
```

A.2.1.2 Le type complexe

L'unité imaginaire est désignée par i ou j. Les nombres complexes peuvent être écrits sous forme cartésienne $a + ib$ ou sous forme polaire $r \times e^{it}$. Les différentes écritures possibles sont `a+ib`, `a+i*b`, `a+b*i`, `a+bi` et `r*exp(it)` ou `r*exp(i*t)` avec a, b, r et t des variables de type réel. Les commandes `imag`, `real`, `abs`, `angle` permettent de passer aisément de la forme polaire à la forme cartésienne et réciproquement. Si z est de type complexe, les instructions `imag(z)` et `real(z)` retournent la partie imaginaire et la partie réelle de z. Les instructions `abs(z)` et `angle(z)` retournent le module et l'argument de z. On fera attention au fait que les identificateurs i et j ne sont pas réservés. Aussi il est possible que des variables de noms i et j aient été redéfinies au cours d'un calcul antérieur et soient toujours actives. Si c'est la cas, on peut soit détruire ces deux variables (`clear i j`), i et j redeviennent alors l'unité imaginaire, soit réaffecter à i ou à j la valeur unité imaginaire par l'instruction `i = sart(-1)`. On se méfiera donc des boucles d'indices i et j dans lesquelles on manipule des variables de type complexe. On fera également attention à ne pas laisser d'espace autour de l'unité imaginaire afin d'éviter de mauvaises interprétations des données dans certains cas. Comparez par exemple,

```
>> z = [1+i, 2, 3i]
z =
   1.0000 + 1.0000i   2.0000                  0 + 3.0000i
>> y = [1+i, 2, 3 i]
y =
   1.0000 + 1.0000i   2.0000             3.0000               0 + 1.0000i
>>
```

A.2.1.3 Le type chaîne de caractères

Une chaîne de caractères est un tableau de caractères. Une donnée de type chaîne de caractères (`char`) est représentée sous la forme d'une suite de caractères encadrée d'apostrophes simples (’). Une variable de type chaîne de caractères étant interprétée comme un tableau de caractères, il est possible de manipuler chaque lettre de la chaîne en faisant référence à sa position dans la chaîne. La concaténation de chaînes de caractères s'effectue selon les règles de manipulation des tableaux. L'exemple suivant présente différentes manipulations d'une chaîne de caractères.

```
>> ch1 = 'bon'
ch1 =
bon
>> ch2 = 'jour'
ch2 =
jour
>> whos
  Name      Size         Bytes  Class
  ch1       1x3              6  char array
  ch2       1x4              8  char array
Grand total is 7 elements using 14 bytes
>> ch = [ch1,ch2]
ans =
bonjour
>> ch(1), ch(7), ch(1 :3)
ans =
b
ans =
r
ans =
bon
>> ch3 = 'soi';
>> ch = [ch(1 :3), ch3, ch(7)]
ans =
bonsoir
>>
```

Si une chaîne de caractères doit contenir le caractère apostrophe (’) celui-ci doit être doublé dans la chaîne (ainsi pour affecter le caractère apostrophe (’) à une variable on devra écrire ’’’’, soit quatre apostrophes.

```
>> rep = 'aujourd'hui'
??? rep = 'aujourd'hui

Missing operator, comma, or semi-colon.
>>  rep = 'aujourd''hui'
rep =
aujourd'hui
>>  apos = ''''
apos =
'
>>
```

La chaîne de caractères vide s'obtient par deux apostrophes `''`. La commande `isempty` permet de tester si une variable de type chaîne de caractères est vide ou non. La commande `strcmp` permet de tester si deux chaînes de caractères sont égales ou non.

A.2.1.4 Le type logique

Le type logique (`logical`) possède deux formes : 0 pour faux et 1 pour vrai. Un résultat de type logique est retourné par certaines fonctions ou dans le cas de certains tests. Dans l'exemple qui suit on considère une variable x contenant la valeur 123 et une variable y définie par l'expression mathématique $y = exp(log(x))$. On teste si les variables x et y contiennent les mêmes valeurs. La variable tst est une variable de type logique qui vaut 1 (vrai) les valeurs sont égales et 0 (faux) sinon. Suivant la valeur de tst, on affiche la phrase x est égal a y ou la phrase x est différent de y. Dans l'exemple proposé, compte tenu des erreurs d'arrondis lors du calcul de $exp(log(123))$, la valeur de la variable y ne vaut pas exactement 123. On pourra également considérer le cas où $x = 12$.

```
>> x = 123; y = exp(log(x));
>> tst = ( x==y );
>> if tst, disp('x est egal a y '), else disp('x est different de y '), end
x est different de y
>> whos
  Name      Size         Bytes  Class
  tst       1x1              8  double array (logical)
  x         1x1              8  double array
  y         1x1              8  double array
Grand total is 3 elements using 24 bytes
>> format long
>> x, y, tst
x =
   123
y =
     1.229999999999999e+02
tst =
     0
>> format, clear
>>
```

A.2.2 Les vecteurs

A.2.2.1 Définir un vecteur

On définit un vecteur ligne en donnant la liste de ses éléments entre crochets (`[ ]`). Les éléments sont séparés au choix par des espaces ou par des virgules. On définit un vecteur colonne en donnant la liste de ses éléments séparés au choix par des points virgules (`;`) ou par des retours chariots. On peut transformer un vecteur ligne x en un vecteur colonne et réciproquement en tapant `x'` (`'` est le symbole de transposition). Il est inutile de définir la longueur d'un vecteur au préalable. Cette longueur sera établie automatiquement à partir de l'expression mathématique définissant le vecteur ou à partir des données. On peut obtenir la longueur d'un vecteur donné grâce à la commande length. Un vecteur peut également être défini par blocs selon la même syntaxe. Si par exemple x_1, x_2 et x_3 sont trois vecteurs (on note `x1`, `x2` et `x3` les variables Matlab correspondantes), ondéfinit le vecteur bloc (x_1 | x_2 | x_3) par l'instruction `X = [x1 x2 x3]`.

```
>> x1 = [1 2 3], x2 = [4,5,6,7], x3 = [8; 9; 10]
x1 =
     1     2     3
x2 =
     4     5     6     7
x3 =
     8
     9
    10
>> length(x2), length(x3)
ans =
     4
ans =
     3
>> whos
  Name      Size         Bytes  Class
  x1        1x3             24  double array
  x2        1x4             32  double array
  x3        3x1             24  double array
Grand total is 10 elements using 80 bytes
>> x3'
ans =
     8     9    10
>> X = [x1 x2 x3']
X =
     1     2     3     4     5     6     7     8     9    10
>>
```

Les éléments d'un vecteur peuvent être manipulés grâce à leur indice dans le tableau. Le k-ieme élément du vecteur x est désignée par $x(k)$. Le premier élément d'un vecteur a obligatoirement pour indice 1. En pratique ceci impose de faire des translations d'indices si par exemple on souhaite définir une suite $(x_0\ ,\ x_1\ ,\ ...\ ,\ x_n)$. Le terme x_0 de la suite correspondra à l'élément $x(1)$ du vecteur et le terme x_N à l'élément $x(N+1)$. Il est possible de manipuler plusieurs éléments d'un vecteur simultanément. Ainsi les éléments k à l du vecteur x sont désignés par $x(k : l)$. On peut également manipuler facilement les éléments d'un vecteur dont les indices sont en progression arithmétique. Ainsi si l'on souhaite extraire les éléments $k, k+p, k+2p, ..., k+Np = l$, on écrira `x(k :p :l)`. Plus généralement, si K est un vecteur de valeurs entières, $X(K)$ retourne les éléments du vecteur X dont les indices sont les éléments du vecteur K. Reprenons l'exemple précédent.

```
>> X(5)
ans =
     5
>> X(4 :10)
ans =
     4     5     6     7     8     9    10
>> X(2 :2 :10)
ans =
     2     4     6     8    10
>> K = [1 3 4 6]; X(K)
ans =
```

```
     1     3     4     6
>>
```

Il est très facile de définir un vecteur dont les composantes forment une suite arithmétique. Pour définir un vecteur x dont les composantes forment une suite arithmétique de raison h, de premier terme a et de dernier terme b, on écrira `x = a :h :b`. Si $a-b$ n'est pas un multiple de h, le dernier élément du vecteur x sera $a+Ent((a-b)/h)h$ où Ent est la fonction partie entière. La commande linspace permet de définir un vecteur x de longueur N dont les composantes forment une suite arithmétique de premier terme a et de dernier terme b (donc de raison $(a-b)/N$). Les composantes du vecteur sont donc linéairement espacées. La syntaxe est `x = linspace(a,b,N)`.

```
>> x = 1.1 :0.1 :1.9
x =
  Columns 1 through 7
    1.1000    1.2000    1.3000    1.4000    1.5000    1.6000    1.7000
  Columns 8 through 9
    1.8000    1.9000
>> x = 1.1 :0.2 :2
x =
    1.1000    1.3000    1.5000    1.7000    1.9000
>> x = linspace(1.1,1.9,9)
ans =
  Columns 1 through 7
    1.1000    1.2000    1.3000    1.4000    1.5000    1.6000    1.7000
  Columns 8 through 9
    1.8000    1.9000
>>
```

A.2.2.2 Vecteurs spéciaux

Les commandes `ones`, `zeros` et `rand` permettent de définir des vecteurs dont les éléments ont respectivement pour valeurs 0, 1 et des nombres générés de manière aléatoire.

`ones(1,n)`	:	vecteur ligne de longueur n dont tous les éléments valent 1
`ones(m,1)`	:	vecteur colonne de longueur m dont tous les éléments valent 1
`zeros(1,n)`	:	vecteur ligne de longueur n dont tous les éléments valent 0
`zeros(m,1)`	:	vecteur colonne de longueur m dont tous les éléments valent 0
`rand(1,n)`	:	vecteur ligne de longueur n dont les éléments sont générées de manière aléatoire entre 0 et 1
`rand(m,1)`	:	vecteur colonne de longueur m dont les éléments sont générés de manière aléatoire entre 0 et 1

Tab. A.1 – Les vecteurs spéciaux

A.2.3 Les matrices

A.2.3.1 Définir une matrice

On a déjà vu que l'on définissait la matrice $A = \begin{pmatrix} 1 & 3 \\ 4 & 2 \end{pmatrix}$ en tapant `A = [ 1 3; 4 2 ]`. D'une façon générale, on définit une matrice en donnant la liste de ses éléments entre crochets.

Signalons que Matlab admet d'autres façons d'écrire les matrices. Les éléments d'une ligne de la matrice peuvent être séparés au choix par un blanc ou bien par une virgule (`,`). Les lignes quant à elles peuvent être séparées au choix par le point-virgule (`;`) ou par un retour chariot. Par exemple, on peut aussi écrire la matrice A de la manière suivante :

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

Un élément d'une matrice est référencé par ses numéros de ligne et de colonne. `A(i,j)` désigne le i-ième élément de la j-ième ligne de la matrice A. Ainsi `A(2,1)` désigne le premier élément de la deuxième ligne de A,

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

La commande size permet d'obtenir les dimensions d'une matrice A donnée. On peut soit obtenir de manière séparée le nombre de lignes et de colonnes par les instructions `size(A,1)` et `size(A,2)` respectivement, soit obtenir le nombre m de lignes et le nombre n de colonnes par l'instruction `[m,n] = size(A)`.
On peut construire très simplement une matrice par blocs. Si A, B, C, D désignent quatre matrices (aux dimensions compatibles), on définit la matrice bloc, $A = \left(\begin{array}{c|c} 1 & 3 \\ \hline 4 & 2 \end{array}\right)$ par l'instruction `K = [A B ; C D]`. Voici un exemple de construction par blocs de la matrice

$$A = \left(\begin{array}{ccc|cc|c} 35 & 1 & 6 & 26 & 19 & 24 \\ 3 & 32 & 7 & 21 & 23 & 25 \\ 31 & 9 & 2 & 22 & 27 & 20 \\ \hline 8 & 28 & 33 & 17 & 10 & 15 \\ 30 & 5 & 34 & 12 & 14 & 16 \\ \hline 4 & 36 & 29 & 13 & 18 & 11 \end{array}\right)$$

```
>> A11 = [35 1 6; 3 32 7; 31 9 2];
>> A12 = [26 19; 21 23; 22 27];
>> A21 = [ 8 28 33; 30 5 34];
>> A22 = [17 10; 12 14];
>> B11 = [ A11 A12; A21 A22 ]
B11 =
```

```
    35     1     6    26    19
     3    32     7    21    23
    31     9     2    22    27
     8    28    33    17    10
    30     5    34    12    14
>> B12 = [ 24 25 20 15 16]';
>> B = [ B11 B12];
>> B21 = [ 4 36 29 13 18 11];
>> B = [ B ; B21]
B =
    35     1     6    26    19    24
     3    32     7    21    23    25
    31     9     2    22    27    20
     8    28    33    17    10    15
    30     5    34    12    14    16
     4    36    29    13    18    11
>>
```

A.2.3.2 Matrices spéciales

Certaines matrices se construisent très simplement grâce à des commandes dédiées. Citons les plus utilisées :

`eye(n)`	:	la matrice identité de dimension n
`ones(m,n)`	:	la matrice à m lignes et n colonnes dont tous les éléments valent 1
`zeros(m,n)`	:	la matrice à m lignes et n colonnes dont tous les éléments valent 0
`rand(m,n)`	:	une matrice à m lignes et n colonnes dont les éléments sont générés de manière aléatoire entre 0 et 1

Signalons que si les entiers m et n sont égaux on peut se contenter de ne spécifier qu'une seule valeur de dimension : `ones(n)` est la matrice carrée de dimension n dont tous les éléments valent 1. Mentionnons enfin la commande `magic(n)` qui permet d'obtenir une matrice magique de dimension n.

```
>> eye(3)
ans =
     1     0     0
     0     1     0
     0     0     1
>> ones(3,2)
ans =
     1     1
     1     1
     1     1
>> zeros(2)
ans =
     0     0
     0     0
>> rand(2,3)
ans =
    0.4565    0.8214    0.6154
```

```
    0.0185    0.4447    0.7919
>> magic(6)
ans =
    35     1     6    26    19    24
     3    32     7    21    23    25
    31     9     2    22    27    20
     8    28    33    17    10    15
    30     5    34    12    14    16
     4    36    29    13    18    11
>>
```

A.2.3.3 Manipuler des matrices

Le symbole deux-points (:) permet d'extraire simplement des lignes ou des colonnes d'une matrice. Le j-ieme vecteur colonne de la matrice A est désigné par `A( :,j)`. C'est simple, il suffit de traduire le symbole deux-points (:) par «tout». Ainsi `A( :,j)` désigne toutes les lignes et la j-ieme colonne de la matrice A. Bien entendu, la i-ieme ligne de la matrice A est désignée par $A(i,:)$.

```
>> A = magic(5)
A =
    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
>>  A(1, :)
ans =
    17    24     1     8    15
>>  A( :,2)
ans =
    24
     5
     6
    12
    18
>>
```

Si l'on souhaite échanger les colonnes 2 et 3 de la matrice A une première possibilité consiste à exécuter :

```
>> v = A( :,2); A( :,2) = A( :,3); A( :,3) = v;
A =
    17     1    24     8    15
    23     7     5    14    16
     4    13     6    20    22
    10    19    12    21     3
    11    25    18     2     9
>>
```

On peut également extraire plusieurs lignes ou colonnes simultanément. Si J est un vecteur d'entiers, `A( :,J)` est la matrice issue de A dont les colonnes sont les colonnes de la matrice A

d'indices contenus dans le vecteur J. De même `A(J, :)` est la matrice issue de A dont les lignes sont les lignes de la matrice A d'indices contenus dans le vecteur J. D'une façon plus générale, il est possible de n'extraire qu'une partie des éléments des lignes et colonnes d'une matrice. Si L et C sont deux vecteurs d'indices, `A(L,C)` désigne la matrice issue de la matrice A dont les éléments sont les `A(i,j)` tels que i soit dans L et j soit dans C.

```
>> A = magic(5)
A =
    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
>> L = [1 3 5]; C = [3 4];
>> A(L,C)
ans =
     1     8
    13    20
    25     2
>> A(1 :2 :5,3 :4)
ans =
     1     8
    13    20
    25     2
>>
```

Dans la dernière instruction, on a utilisé la forme spéciale permettant de définir un vecteur dont les composantes sont en progression arithmétique. Une seconde possibilité pour échanger les lignes 2 et 3 de la matrice A consiste à exécuter :

```
>> J = [1 3 2 4]; A = A( :,J)
A =
    17     1    24     8    15
    23     7     5    14    16
     4    13     6    20    22
    10    19    12    21     3
    11    25    18     2     9
>>
```

Il existe des commandes Matlab permettant de manipuler globalement des matrices. La commande `diag` permet d'extraire la diagonale d'une matrice : si A est une matrice, `v=diag(A)` est le vecteur composé des éléments diagonaux de A. Elle permet aussi de créer une matrice de diagonale fixée : si v est un vecteur de dimension n, `A=diag(v)` est la matrice diagonale dont la diagonale est v.

```
>> A=eye(3); diag(A)
ans =
     1
     1
     1
>> v=[1 :3]
```

```
v =
     1 2 3
>> diag(v)
 ans =
     1 0 0
     0 2 0
     0 0 3
```

On n'est pas obligé de se limiter à la diagonale principale. La commande `diag` admet un second paramètre k pour désigner la k sur-diagonale (si $k > 0$) ou la k sous-diagonale (si $k < 0$).

```
>> A = [4 5 6 7 ; 3 4 5 6
        2 3 4 5; 1 2 3 4]
A =
     4     5     6     7
     3     4     5     6
     2     3     4     5
     1     2     3     4
>> diag(A,1)
ans =
     5
     5
     5
>> diag(A,-2)
ans =
     2
     2
>>
```

On construit à l'aide de la commande `diag` très simplement des matrices tridiagonales. Par exemple la matrice correspondant à la discrétisation par différences finies du problème de Dirichlet en dimension 1 s'obtient ainsi

```
>> N=5;
>> A=diag(2*ones(N,1)) - diag(ones(N-1,1),1) - diag(ones(N-1,1),-1)
A =
     2    -1     0     0     0
    -1     2    -1     0     0
     0    -1     2    -1     0
     0     0    -1     2    -1
     0     0     0    -1     2
>>
```

On dispose également de la commande `tril` permet d'obtenir la partie triangulaire inférieure (l pour lower) d'une matrice. La commande `triu` permet d'obtenir la partie triangulaire supérieure (u pour upper) d'une matrice.

```
>> A = [ 2 1 1 ; -1 2 1 ; -1 -1 2]
A =
     2     1     1
    -1     2     1
```

```
    -1    -1     2
>> triu(A)
ans =
     2     1     1
     0     2     1
     0     0     2
>> tril(A)
ans =
     2     0     0
    -1     2     0
    -1    -1     2
>>
```

Comme pour la commande `diag`, les commandes `triu` et `tril` admettent un second paramètre k. On peut ainsi obtenir la partie triangulaire supérieure (ou inférieure) à partir du k diagonal. Ainsi,

```
>> tril(A,-1)
ans =
     0     0     0
    -1     0     0
    -1    -1     0
>> tril(A,1)
ans =
     2     1     0
    -1     2     1
    -1    -1     2
>>
```

On obtient la transposée de la matrice A à coefficients réels en tapant `A'`. Si la matrice est à coefficients complexes, `A'` retourne la matrice adjointe de A.

```
>> A = [0 1 2; -1 0 1; -2 -1 0]
A =
     0     1     2
    -1     0     1
    -2    -1     0
>> A'
ans =
     0    -1    -2
     1     0    -1
     2     1     0
>>
```

A.2.4 La structure Sparse

On appelle matrice creuse (le terme anglais est sparse matrix) une matrice comportant une forte proportion de coefficients nuls. De nombreux problèmes issus de la physique conduisent à l'analyse de systèmes linéaires à matrice creuse. L'intérêt de telles matrices résulte non seulement de la réduction de la place mémoire (on ne stocke pas les zéros) mais aussi de la réduction du nombre d'opérations (on n'effectuera pas les opérations portant sur les zéros). Par défaut dans

Matlab une matrice est considérée comme pleine (ou *full* en anglais), c'est-à-dire que tous ses coefficients sont mémorisés. Si M est une matrice, la commande `sparse(M)` permet d'obtenir la même matrice mais stockée sous la forme `sparse`. Si l'on a une matrice stockée sous la forme sparse, on peut obtenir la même matrice stockée sous la forme ordinaire par la commande `full`. Il est possible de visualiser graphiquement la structure d'une matrice grâce à la commande `spy`. Si M est une matrice, la commande `spy(M)` ouvre une fenêtre graphique et affiche sous forme de croix les éléments non-nuls de la matrice. Le numéro des lignes est porté sur l'axe des ordonnées, celui des colonnes en abscisse. On obtient également le nombre d'éléments non-nuls de la matrice. La commande `nnz` permet d'obtenir le nombre d'éléments non-nuls d'une matrice. Reprenons l'exemple de la matrice tridiagonale issue de la discrétisation par différences finies du problème de Dirichlet en dimension 1.

```
>> N=5;
>> A=diag(2*ones(N,1)) - diag(ones(N-1,1),1) - diag(ones(N-1,1),-1)
A =
     2    -1     0     0     0
    -1     2    -1     0     0
     0    -1     2    -1     0
     0     0    -1     2    -1
     0     0     0    -1     2
>> nnz(A)
ans =
    13
>> B = sparse(A)
B =
   (1,1)        2
   (2,1)       -1
   (1,2)       -1
   (2,2)        2
   (3,2)       -1
   (2,3)       -1
   (3,3)        2
   (4,3)       -1
   (3,4)       -1
   (4,4)        2
   (5,4)       -1
   (4,5)       -1
   (5,5)        2
>> whos
  Name      Size         Bytes  Class
  A         5x5            200  double array
  B         5x5            180  sparse array
  N         1x1              8  double array
Grand total is 39 elements using 388 bytes
>> spy(A)
```

Pour les très grosses matrices creuses, il n'est bien entendu pas souhaitable d'utiliser une structure pleine (full) pour définir la matrice avant de passer en stockage sparse. La commande `sparse` permet de définir une matrice creuse directement sous la forme sparse. On l'utilise de la

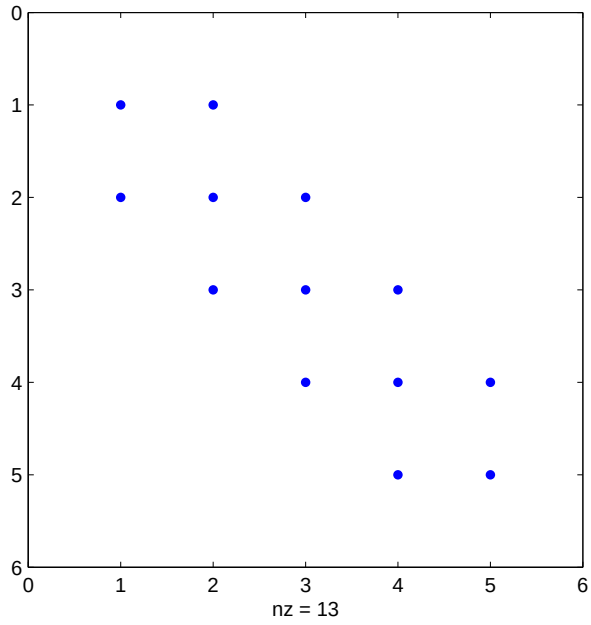


FIG. A.2 – Résultat de la commande `spy(B)`.

façon suivante : `A = sparse(is,js,s,n,m)` pour définir une matrice A à n lignes et m colonnes dont les coefficients sont nuls (et donc non mémorisés) sauf les éléments $a_{is(l),js(l)}$ qui valent $s(l)$ pour l variant de 1 à L longueur du tableau s (`L = length(s)`).

```
>> N=5;
>> s =[2*ones(1,N), -ones(1,N-1), -ones(1,N-1)]
s =
  Columns 1 through 12
     2     2     2     2     2    -1    -1    -1    -1    -1    -1    -1
  Column 13
    -1
>> is = [1 :N,1 :N-1,2 :N]
is =
  Columns 1 through 12
     1     2     3     4     5     1     2     3     4     2     3     4
  Column 13
     5
>> js = [1 :N,2 :N,1 :N-1]
js =
  Columns 1 through 12
     1     2     3     4     5     2     3     4     5     1     2     3
  Column 13
     4
>> B = sparse(is,js,s,N,N)
B =
   (1,1)        2
   (2,1)       -1
   (1,2)       -1
   (2,2)        2
   (3,2)       -1
```

```
   (2,3)       -1
   (3,3)        2
   (4,3)       -1
   (3,4)       -1
   (4,4)        2
   (5,4)       -1
   (4,5)       -1
   (5,5)        2
>> A = full(B)
A =
     2    -1     0     0     0
    -1     2    -1     0     0
     0    -1     2    -1     0
     0     0    -1     2    -1
     0     0     0    -1     2
>>
```

Les commandes `spdiags`, `speye` et `sprand` permettent de construire des matrices diagonales, identités et des matrices dont les éléments sont des nombres aléatoires avec un stockage sparse. Elles s'utilisent de la même manière que leur équivalent `diag`, `eye` et `rand` pour les matrices pleines.

A.3 Calculer avec Matlab

A.3.1 Les constantes

Les principales constantes sont :

`pi`	:	3.1415926535897
`I`	:	$i^2 = -1$
`J`	:	$j^2 = -1$
`eps`	:	précision numérique relative
`realmin`	:	plus petit nombre à virgule flottante manipulable
`realmax`	:	plus grand nombre à virgule flottante manipulable
`inf`	:	infini. Est obtenu quand on essaie d'évaluer une expression dont le résultat excède realmax
`NaN`	:	not-a-number. Est obtenu quand on essaie d'effectuer une opération non-définie comme 0/0

Les valeurs des constantes `eps`, `realmin` et `realmax` dépendent de la machine sur laquelle Matlab est installé. Par exemple sur un PC avec Windows XP on a `eps = 2.2204e-16`, `realmin = 2.2251e-308` et `realmax = 1.7977e+308`. Les noms des constantes ne sont pas réservés, c'est-à-dire qu'il est possible de définir des variables de même nom. Dans ce cas, l'identificateur fera référence à la variable définie par l'utilisateur et non plus à la constante Matlab. On fera attention par exemple, si l'on utilise le type complexe, à ne pas écrire de boucles ayant i ou j comme indices. Pour que l'identificateur fasse à nouvelle référence à la constante Matlab, il suffit de supprimer la variable de même nom de la mémoire par la commande `clear`.

```
>> pi = 0; cos(pi)
ans =
```

```
1
>> clear pi
>> cos(pi)
ans =
-1
>>
```

A.3.2 Opérations et fonctions portant sur les scalaires

Il est bien entendu possible d'utiliser Matlab pour faire de simples additions (Si x et y sont deux variables scalaires de type réel, `x+y`, `x-y`, `x*y` et `x/y` désignent les quatre opérations usuelles entre les valeurs de x et y dans $\mathbb{R}$. Si x et y sont deux variables scalaires de type complexe, $x+y$, $x-y$, $x \times y$ et x/y désignent les quatre opérations usuelles entre les valeurs de x et y dans $\mathbb{C}$. L'exponentiation s'obtient grâce au symbole `^` (la syntaxe est `x^y`).
La commande `rem` donne le reste (*remainder*) de la division entière de deux entiers (la syntaxe est `rem(m,n)`). Les commandes `lcm(m,n)` et `gcd(m,n)` retournent respectivement le plus petit multiple commun et le plus grand commun diviseur à deux entiers m et n. La commande `factor(n)` permet d'obtenir les termes de la décomposition en facteurs premiers de l'entier n.
Les fonctions mathématiques incorporées sont :

`log(x)`	:	logarithme néperien de x,
`log10(x)`	:	logarithme en base 10 de x,
`exp(x)`	:	exponentielle de x,
`sqrt(x)`	:	racine carrée de x (s'obtient aussi par x. 0.5),
`abs(x)`	:	valeur absolue de x,
`sign(x)`	:	fonction valant 1 si x est positif ou nul et 0 sinon.

Lorsque la fonction est définie sur le corps des nombres complexes l'argument peut être de type complexe. On dispose également de fonctions spécifiques aux complexes :

`conj(z)`	:	le conjugué de z,
`abs(z)`	:	le module de z,
`angle(z)`	:	argument de z,
`real(z)`	:	partie réelle de z,
`imag(z)`	:	partie imaginaire de z.

Les fonctions d'arrondis sont :

`round(x)`	:	entier le plus proche de x,
`floor(x)`	:	arrondi par défaut,
`ceil(x)`	:	arrondi par excès,
`fix(x)`	:	arrondi par défaut un réel positif et par excès un réel négatif.

Les fonctions trigonométriques et hyperboliques sont :

`cos`	:	cosinus,
`acos`	:	cosinus inverse (arccos),
`sin`	:	sinus,
`asin`	:	sinus inverse (arcsin),
`tan`	:	tangente,
`atan`	:	tangente inverse (arctan),
`cosh`	:	cosinus hyperbolique (ch),
`acosh`	:	cosinus hyperbolique inverse (argch),
`sinh`	:	sinus hyperbolique (sh),
`asinh`	:	sinus hyperbolique inverse (argsh),
`tanh`	:	tangente hyperbolique (th),
`atanh`	:	tangente hyperbolique inverse (argth).

A.3.3 Opérations et fonctions portant sur les vecteurs

Une particularité de Matlab est de permettre d'effectuer des opérations de manière globale sur les éléments d'un vecteur de type réel ou complexe sans avoir à manipuler directement ses éléments. Si k est une variable scalaire et x un vecteur, l'instruction `k*x` multiplie tous les éléments de x par k. Si x et y sont deux vecteurs de longueur identique, l'instruction `z = x+y` (respectivement `x-y`) définit le vecteur z dont les éléments sont $z(i) = x(i) + y(i)$ (respectivement $z(i) = x(i) - y(i)$). On obtient un vecteur z dont la i-eme composante est le produit (respectivement le quotient) de la i-eme composante du vecteur x par la i-eme composante du vecteur y en effectuant l'instruction `z = x.*y` (respectivement `z = x./y`).
Attention à ne pas oublier le point!
La commande `cross(x,y)` permet de calculer le produit vectoriel des deux vecteurs x et y. Il n'y a pas dans la version 5.1 de commande dédiée pour calculer le produit scalaire de deux vecteurs. Il s'obtient grâce à l'instruction `sum(x.*y)`. Dans les versions suivantes, la commande `dot(x,y)` permet de calculer le produit scalaire des deux vecteurs x et y.
Les fonctions mathématiques incorporées décrites au paragraphe précédent peuvent être utilisées avec un argument qui est un vecteur. La fonction est alors appliquée à tous les éléments du vecteur en même temps.

```
>> x = [1 :10 :100]; y=sqrt(x);
y =
Columns 1 through 7
1.0000 3.3166 4.5826 5.5678 6.4031 7.1414 7.8102
Columns 8 through 10
8.4261 9.0000 9.5394
>>
```

Il existe également quelques fonctions spécifiques aux vecteurs :

```
>> x=[3 1 2];
>> sum(x)
ans =
     6
>> prod(x)
ans =
     6
```

`sum(x)`	:	somme des éléments du vecteur x,
`prod(x)`	:	produit des éléments du vecteur x,
`max(x)`	:	plus grand élément du vecteur x,
`min(x)`	:	plus petit élément du vecteur x,
`mean(x)`	:	moyenne des éléments du vecteur x,
`sort(x)`	:	ordonne les éléments du vecteur x par ordre croissant,
`fliplr(x)`	:	échange la position des éléments du vecteur x.

```
>> max(x)
ans =
     3
>> min(x)
ans =
     1
>> sort(x)
ans =
     1     2     3
>> fliplr(x)
ans =
     2     1     3
>>
```

Citons enfin quelques fonctions logiques. Les commandes `all` et `any` servent à tester si les éléments d'un vecteur sont nuls ou non. Si v est un vecteur de valeurs numériques, `all(v)` retourne vrai (1) si tous les éléments du vecteur sont différents de 0 et faux (0) si au moins un élément vaut 0. `any(v)` retourne vrai (1) si au moins un élément du vecteur est différent de 0 et faux (0) si le vecteur est composé exclusivement de 0.

A.3.4 Opérations et fonctions portant sur les matrices

Si les opérandes sont des matrices, les opérations `+` (addition), `-` (soustraction), `*` (multiplication), `^` (exponentiation), sont alors les opérations matricielles usuelles. Ainsi `A*B` désigne le produit de la matrice A par la matrice B, `A+B` désigne la somme de ces deux matrices et `A^2` le carré de la matrice A.

```
>> A=[1 2 3; 4 5 6]
A =
    1 2 3
    4 5 6
>> B = [1 1; 2 2; 3 3]
B =
     1 1
     2 2
     3 3
>> C = A*B
C =
     14 14
     32 32
>> C^2
```

```
ans =
     644  644
    1472 1472
>>
```

Si les dimensions des matrices A et B sont incompatibles avec l'opération matricielle, Matlab renvoi un message d'erreur :

```
>> A+B
??? Error using ==> + Matrix dimensions must agree.
>>
```

En plus des opérations matricielles usuelles, il est possible d'effectuer des opérations entre matrices «élément par élément». Pour cela, il faut faire précéder l'opérateur d'un point (.). Ainsi si A et B sont deux matrices de même dimension, on obtient la matrice dont le terme d'indices (i, j) est le produit des deux termes d'indices (i, j) des matrices A et B par la commande `A.*B`. De même la commande `A.^2` fournit la matrice dont les termes sont les carrés des termes de la matrice A. Bien entendu les commandes `A.+B` et `A+B` donnent le même résultat.

```
>> A=[1 2 3; 4 5 6]
A =
    1 2 3
    4 5 6
>> B=[1 2 3; 1 2 3]
B =
     1 2 3
     1 2 3
>> A.*B
ans =
     1 4 9
     4 10 18
>> A.^2
ans =
     1     4     9
    16    25    36
>>
```

Les fonctions matricielles les plus courantes sont :
On peut obtenir les différentes normes d'une matrice A grâce à la commande `norm`.
Ces fonctions matricielles incorporées de Matlab peuvent être utilisées avec un argument qui est une matrice sparse. Les exceptions sont les fonctions `rank`, `expm` et `norm` qui nécessitent de passer en stockage full (on exécutera donc `rank(full(B))` par exemple).

A.3.5 Résolution de systèmes linéaires

La commande Matlab \ (*backslash*) est la commande générique pour résoudre un système linéaire. L'algorithme mis en oeuvre dépend de la structure de la matrice A du système. Matlab utilise dans l'ordre les méthodes suivantes :
– Si A est une matrice triangulaire, le système est résolu par simple substitution.
– Si la matrice A est symétrique ou hermitienne, définie positive, la résolution est effectuée par la méthode de Choleski.

`det(A)`	:	renvoie le déterminant de la matrice carrée A.
`eig(A)`	:	renvoie les valeurs propres (*eigenvalues*) de la matrice carrée A. Si l'on souhaite également les vecteurs propres on exécutera `[V,D] = eig(A)` qui renvoie une matrice diagonale D formée des valeurs propres de A et une matrice V dont les vecteurs colonnes sont les vecteurs propres correspondant.
`poly(A)`	:	renvoie les coefficients du polynôme caractéristique associé à la matrice carrée A. On sera vigilant à l'ordre dans lequel sont rangés les coefficients : le premier élément du vecteur est le coefficient du monôme de plus haut degré. Ainsi dans l'exemple suivant il faut lire $p(x) = x^3 - 6x^2 - 72x - 27$, `>> A = [1 2 3 ; 4 5 6 ; 7 8 0]; p = poly(A)` `p = 1 -6 -72 -27`
`inv(A)`	:	renvoie l'inverse de la matrice carrée A.
`rank(A)`	:	renvoie le rang de la matrice carrée A.
`trace(A)`	:	renvoie la trace de la matrice A.
`expm(A)`	:	renvoie l'exponentielle matricielle de A.

`norm(A)`	:	renvoie la norme 2 de la matrice A.
`norm(A,2)`	:	même chose que `norm(A)`.
`norm(A,1)`	:	norme 1 de la matrice A, $\|\|A\|\|_1 = \max_{1\leq j\leq n} \sum_{1\leq i\leq n} \|a_{ij}\|$.
`norm(A,inf)`	:	norme infini de la matrice A, $\|\|A\|\|_\infty = \max_{1\leq i\leq n} \sum_{1\leq j\leq n} \|a_{ij}\|$.
`norm(A,'fro')`	:	norme de Frobenius de la matrice A, $\|\|A\|\|_{fro} = \sqrt{\sum_{1\leq i,j\leq n} \|a_{ij}\|^2}$.

– Si A est une matrice carrée mais n'entrant pas dans les deux cas précédents, une factorisation LU est réalisée en utilisant la méthode d'élimination de Gauss avec stratégie de pivot partiel.
– Si A n'est pas une matrice carrée, la méthode QR est utilisée.

Dans le cas des matrices stockées sous forme sparse, des algorithmes particuliers sont mis en oeuvre. Chacune des méthodes précédentes peut être utilisée de manière spécifique grâce aux commandes `chol`, `lu`, `qr`. Il est également possible d'utiliser des méthodes itératives. Les commandes `cgs`, `bicg`, `bicgstab` mettent par exemple en oeuvre des méthodes de type gradient conjugué.

```
>> A=[1 2 ;3 4]; b=[1 1]';
>>  x= A$\backslash $b
x =
    -1
     1
>> A*x
ans =
     1
     1
>>
```

A.3.6 Les polynômes

Sous Matlab le polynôme de degré n, $p(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$ est défini par un vecteur p de dimension $n+1$ contenant les coefficients $\{a_i\}_{\{i=0,\ldots,n\}}$ rangés dans l'ordre décroissant des indices. C'est-à-dire que l'on a $p(1) = a_n, \ldots, p(n+1) = a_0$. La commande

`polyval` permet d'évaluer le polynôme p (la fonction polynomiale) en des points donnés. La syntaxe est `polyval(p,x)` où x est une valeur numérique ou un vecteur. Dans le second cas on obtient un vecteur contenant les valeurs de la fonction polynomiale aux différents points spécifiés dans le vecteur x. Utilisée avec la commande `fplot`, la commande `polyval` permet de tracer le graphe de la fonction polynomiale sur un intervalle $[x_{min}, x_{max}]$ donné. La syntaxe de l'instruction est : `fplot('polyval([ a_n, ..., a_0] , x)' , [x_min , x_max])`. Voici par exemple comment définir le polynôme $p(x) = x^2 - 1$. Le graphe de la fonction polynomiale est présenté à la figure A.3.

```
>> p = [ 1, 0, -1];
>> polyval(p,0)
ans =
    -1
>> polyval(p,[-2,-1,0,1,2])
ans =
     3     0    -1     0     3
>> fplot('polyval([ 1, 0, -1] , x)' , [-3,3]), grid
```

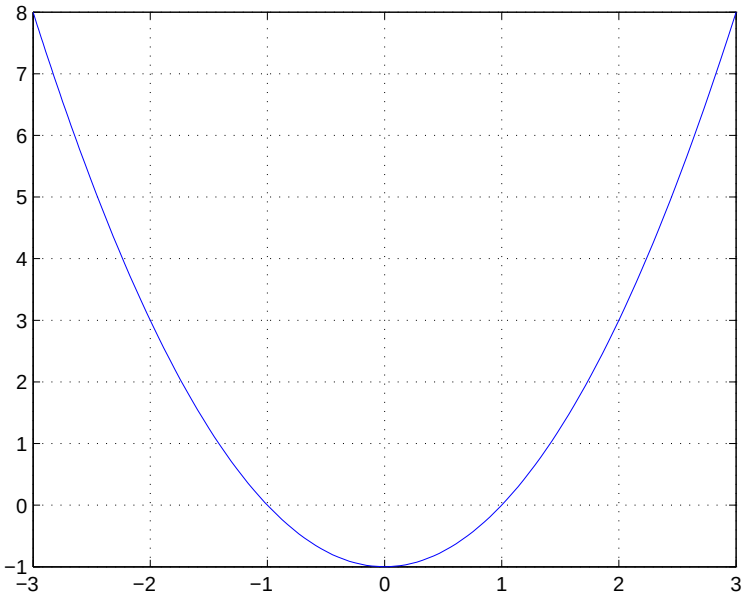


FIG. A.3 – Graphe de la fonction polynomiale $p(x) = x^2 - 1$.

On obtient les racines du polynôme p grâce à l'instruction `roots(p)`.
L'instruction poly permet d'obtenir la représentation canonique $p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$ d'un polynôme de degré n dont on connaît les $n+1$ racines $\{x_i\}_{\{i=0,\ldots,n\}}$. Les coefficients $\{a_i\}_{\{i=0,\ldots,n\}}$ sont obtenus sous forme d'un vecteur p et sont rangés dans l'ordre décroissant des indices. C'est-à-dire que $p(1) = a_n, \ldots, p(n+1) = a_0$.

```
>> r = roots(p)
r =
   -1.0000
    1.0000
>> poly(r)
```

```
ans =
    1.0000    0.0000   -1.0000
>>
```

A.4 Les entrées-sorties

A.4.1 Les formes d'affichage des réels

Matlab dispose de plusieurs formats d'affichage des réels. Par défaut le format est le format court à cinq chiffres. Les autres principaux formats sont :

`format long`	:	format long à 15 chiffres.
`format short e`	:	format court à 5 chiffres avec notation en virgule flottante.
`format long e`	:	format long à 15 chiffres avec notation en virgule flottante.

Matlab dispose également des formats format `short g` et `format long g` qui utilise la «meilleure» des deux écritures à virgule fixe ou à virgule flottante. On obtiendra tous les formats d'affichage possibles en tapant `help format`. On impose un format d'affichage en tapant l'instruction de format correspondante dans la fenêtre de contrôle, par exemple `format long`. Pour revenir au format par défaut on utilise la commande `format` ou `format short`.

```
>> pi
ans =
    3.1416
>> format long
>> pi
ans =
   3.14159265358979
>>  format short e
>> pi\ 3
ans =
   3.1006e+01
>> format short g
>> pi\ 3
ans =
       31.006
>> format short
>>
```

A.4.2 Affichage simple, la commande `disp`

La commande `disp` permet d'afficher un tableau de valeurs numériques ou de caractères. L'autre façon d'afficher un tableau est de taper son nom. La commande `disp` se contente d'afficher le tableau sans écrire le nom de la variable ce qui peut améliorer certaines présentations.

```
>> A = magic(4);
>>  disp(A)
    16     2     3    13
```

```
     5    11    10     8
     9     7     6    12
     4    14    15     1
>> A
A =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
>>
```

On utilise fréquemment la commande `disp` avec un tableau qui est une chaîne de caractères pour afficher un message. Par exemple

```
disp('Calcul du déterminant de la matrice A')
```

On utilise également la commande `disp` pour afficher un résultat. Par exemple :

```
disp(['Le  déterminant de la matrice A vaut ', num2str(det(A))])
```

On remarque que l'usage de la commande `disp` est alors un peu particulier. En effet un tableau doit être d'un type donné, les éléments d'un même tableau ne peuvent donc être des chaînes de caractères et des valeurs numériques. On a donc recours à la commande `num2str` (*number to string*) pour convertir une valeur numérique en une chaîne de caractères. Par défaut la commande `num2str` affiche quatre décimales mais il est possible de lui spécifier le nombre de décimales souhaité en second paramètre. De même il est possible de lui spécifier un format d'affichage particulier de la valeur numérique; on consultera la documentation Matlab pour plus de détails. Attention, si la chaîne de caractères contient une apostrophe il est impératif de doubler l'apostrophe.

A.4.3 Lecture

La commande `input` permet de demander à l'utilisateur d'un programme de fournir des données. La syntaxe est `var = input(' une phrase ')`. La phrase une phrase est affichée et Matlab attend que l'utilisateur saisisse une donnée au clavier. Cette donnée peut être une valeur numérique ou une instruction Matlab. Un retour chariot provoque la fin de la saisie. Une valeur numérique est directement affectée à la variable var tandis qu'une instruction Matlab est évaluée et le résultat est affecté à la variable var. Il est possible de provoquer des sauts de ligne pour aérer le présentation en utilisant le symbole`\n` de la manière suivante : `var = input('\n une phrase :\n ')`. Pensez à mettre un point virgule (`;`) à la fin de l'instruction si vous ne souhaitez pas voir s'afficher `var =`.
Sous cette forme il est impossible d'avoir une donnée de type chaîne de caractères dans la mesure où Matlab essaie d'interpréter cette chaîne de caractères comme une instruction. Si l'on souhaite saisir une réponse de type chaîne de caractères on utilise la syntaxe `var = input(' une phrase ','s')`. Signalons qu'un retour chariot (sans autre chose) initialise la variable `var` au tableau vide `[]`. Voici un exemple d'utilisation de la commande `input` (on suppose que la variable res contient une valeur numérique).

```
rep = input(' Affichage du resultat ? o/n [o] ','s');
if isempty(rep), rep = 'o'; end
if rep == 'o' |  rep == 'y'
  disp(['Le resultat vaut ', num2str(res)])
end
```

A.4.4 Impressions dirigées par format

La commande `sprintf` permet l'impression de variables selon un modèle donné. Un modèle d'édition se présente sous la forme du symbole pourcent (`%`) suivi d'indications permettant de composer le contenu du champ à imprimer, en particulier sa longueur en nombre de caractères. Le modèle d'édition utilisé par Matlab est le modèle d'édition du langage C. La syntaxe de la commande `sprintf` est :

```
sprintf(format, variables)
```

où
- `variables` est le nom des variables à imprimer suivant le modèle d'édition spécifié dans format;
- `format` est le format d'édition. Il s'agit d'une chaîne de caractères contenant les modèles d'éditions des variables à imprimer.

A.4.4.1 Modèle d'édition de caractères

Un modèle d'édition de caractères est de la forme `%Ls` où `%` est le symbole de début de format et `s` le symbole précisant que la donnée est de type chaîne de caractères. `L` est un entier donnant la longueur total du champ (en nombre de caractères). Par défaut le champ est justifié à droite (si la longueur de la chaîne de caractères est plus petite que la longueur `L` du champ, des espaces seront insérés après la chaîne de caractères). Le symbole `-` (moins) juste après le symbole `%` permet de justifier à gauche. En l'absence de l'entier L la longueur totale du champ est égale au nombre de caractères de la chaîne.

```
>> sprintf('%s', 'il fera beau a Metz')
ans =
il fera beau a Metz
>> temps = 'il fera beau a Metz'; sprintf('%s',temps)
ans =
il fera beau a Metz
>> sprintf('%30s', temps)
ans =
          il fera beau a Metz
>> sprintf('%-30s', temps)
ans =
il fera beau a Metz
>> sprintf('meteo : %s', temps)
ans =
meteo : il fera beau a Metz
>>
```

A.4.4.2 Modèle d'édition des réels

Un modèle d'édition de réel est de la forme `%+- L.D t`, où `%` est le symbole de début de format, `L` est un entier donnant la longueur total du champ (en nombre de caractères, point virgule compris), `D` est le nombre de décimales à afficher et `t` spécifie le type de notation utilisée. Par défaut le champ est justifié à droite (si la longueur de la variable est plus petite que la longueur du champ `L`, des espaces sont insérés à gauche). Le symbole `-` (moins) permet de justifier à gauche. Le symbole `+` (plus) provoque l'affichage systématique d'un signe `+` devant les réels positifs. Les principales valeurs possibles pour `t` sont les suivantes :

d	:	pour les entiers
e	:	pour une notation à virgule flottante où la partie exposant est délimitée par un e minuscule (ex : 3.1415e+00)
E	:	même notation mais E remplace e (ex : 3.1415E+00)
f	:	pour une notation à virgule fixe (ex : 3.1415)
g	:	la notation la plus compacte entre la notation à virgule flottante et la notation à virgule fixe est utilisée

```
>> x = pi/3; y = sin(x);
>> sprintf('sin(%8.6f) = %4.2f', x,y)
ans =
sin(1.047198) = 0.87
>> sprintf('sin(%8.6f)  = %4.2E', x,y)
ans =
exp(1.047198)  =  8.66E-01
>>
```

A.4.4.3 Utilisations particulières

La commande `sprintf` est vectorielle : si la variable n'est pas scalaire le format d'impression est réutilisé pour tous les éléments du tableau, colonne par colonne.

```
>> x = [1 :10];
>> sprintf(' %d ,',x)
ans =
 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10 ,
>>
```

Il est possible d'utiliser les symboles suivant dans les chaînes de caractères

\n	:	provoque le passage à une nouvelle ligne
\t	:	insère une tabulation horizontale
\b	:	décale l'impression du champ suivant d'un caractère vers la gauche
\r	:	saut horizontal

```
>> z =[]; x = [1 :10]; for i=1 :length(x), z = [z ,x(i), log(x(i))]; end;
>> s = sprintf('%4.1f | %8.6E \n ', z )
ans =
  1.0 | 0.000000E+00
  2.0 | 6.931472E-01
  3.0 | 1.098612E+00
  4.0 | 1.386294E+00
  5.0 | 1.609438E+00
  6.0 | 1.791759E+00
  7.0 | 1.945910E+00
  8.0 | 2.079442E+00
  9.0 | 2.197225E+00
 10.0 | 2.302585E+00
>>
```

Si l'on a besoin d'afficher le caractère `%` on le doublera `%%` pour qu'il ne soit pas interprété comme le début d'un format. La commande fprintf est l'analogue de sprintf pour imprimer de variables selon un modèle donné dans un fichier.

A.5 Programmer sous Matlab

A.5.1 Scripts et fonctions

Il est possible d'enregistrer une séquence d'instructions dans un fichier (appelé un *M-file*) et de les faire exécuter par Matlab. Un tel fichier doit obligatoirement avoir une extension de la forme `.m` (d'où le nom *M-file*) pour être considéré par Matlab comme un fichier d'instructions. On distingue deux types de *M-file*, les fichiers de scripts et lesfichiers de fonctions. Un script est un ensemble d'instructions Matlab qui joue le rôle de programme principal. Si le script est écrit dans le fichier de nom `nom.m` on l'exécute dans la fenêtre Matlab en tapant `nom`. Même si l'on ne souhaite pas à proprement parler écrire de programme, utiliser un script est très utile. Il est en effet beaucoup plus simple de modifier des instructions dans un fichier à l'aide d'un éditeur de texte que de retaper un ensemble d'instructions Matlab dans la fenêtre de commande.
Les fichiers de fonctions ont deux rôles. Ils permettent à l'utilisateur de définir des fonctions qui ne figurent pas parmi les fonctions incorporées de Matlab (built-in functions) et de les utiliser de la même manière que ces dernières (ces fonctions sont nommées fonctions utilisateur). Ils sont également un élément important dans la programmation d'applications où les fonctions jouent le rôle des fonctions et procédures des langages de programmation usuels.
On définit la fonction fonc de la manière suivante:

```
function [vars_1, ..., vars_m] = fonc(vare_1, ..., vare_n)
séquence d'instructions
```

où `vars_1, ..., vars_m` sont les variables de sortie de la fonction;
`vare_1, ..., vare_n` sont les variables d'entrée de la fonction;
`séquence d'instructions` est le corps de la fonction.
Le fichier doit impérativement commencer par le mot-clé `function`. Suit entre crochets les variables de sortie de la fonction, le symbole `=`, le nom de la fonction et enfin les variables d'entrée entre parenthèses. Si la fonction ne possède qu'une seule variable de sortie, les crochets sont inutiles. Il est impératif que la fonction ayant pour nom `fonc` soit enregistrée dans un fichier de nom `fonc.m` sans quoi cette fonction ne sera pas visible par Matlab.
Dans l'exemple qui suit, on définit une fonction modulo qui calcule la valeur de a modulo n en prenant pour système de résidus $\{1, 2, \ldots, n\}$ au lieu de $\{0, 1, \ldots, n-1\}$ (système de résidus considéré par la fonction incorporée mod). Les lignes qui suivent doivent être enregistrées dans un fichier de nom `modulo.m`.

```
function [r,q] = modulo(a,n)

% Calcule la valeur de a modulo n en prenant pour systeme de residus

% 1, ... , n au lieu de 0, ... , n-1.
%
% appel : [r,q] = modulo(a,n)
%
% Arguments de sortie :
%  r : le residu
```

```
%  q : le quotient

q = floor(a./n);
r = a - n*q;

% si le reste de la division entiere vaut 0, le residu vaut par
convention n
if r == 0, r = n; end
```

Les lignes précédées du symbole % sont des lignes de commentaire. Les lignes de commentaire situées entre la ligne function et la 1ere ligne d'instructions sont affichées si l'on demande de l'aide sur la fonction modulo.

```
>> help modulo

 Calcule la valeur de a modulo n en prenant pour systeme de residus
 1, ... , n au lieu de 0, ... , n-1.

 appel : [r,q] = modulo(a,n)

 Arguments de sortie :
  r : le residu
  q : le quotient

>>
```

L'appel d'une fonction utilisateur s'effectue de la même façon que l'appel de n'importe quelle fonction Matlab :

```
>> b = 10 ; m = 4;
>> [r,q] = modulo(b,m)
r =
     2
q =
     2
>> modulo(10,5)
ans =
     5
>>
```

Remarques : Il n'y a pas de mot-clé (par exemple end) pour indiquer la fin de la fonction. La fonction est supposée se terminer à la fin du fichier. Il est toutefois possible de provoquer un retour au programme appelant dans le corps de la fonction grâce à la commande `return`.
On ne peut écrire qu'une seule fonction par fichier (qui doit porter le nom de cette fonction). Toutefois dans la version 5 de Matlab existe la notion de «sous-fonction». Une sous-fonction est une fonction écrite dans le même fichier qu'une autre fonction (dite principale) et qui ne sera utilisable que par cette fonction principale (une sous-fonction ne peut pas être appelée par un autre sous-programme que la fonction principale).
Si le fichier ne commence pas par le mot-clé `function` on a tout simplement écrit un script!
La gestion des variables d'entrée et de sortie est très souple sous Matlab. Si l'on n'est intéressé que par le résidu et pas par le quotient, on peut se contenter de ne mettre qu'une seule variable

de sortie, `v = modulo(10,4)`. Dans cet appel la variable v contiendra le résidu (la première variable de sortie). Par contre, même si l'on ne souhaite recueillir que le quotient, on est obligé d'effectuer un appel de la forme `[r,q] = modulo(10,4)` et donc de définir une variable inutile. Aussi, d'une manière générale, il est bon de ranger les variables de sortie par ordre d'importance. Il est également possible d'appeler une fonction donnée avec moins de variables d'entrée que le nombre indiqué pour la définition de la fonction (il faut bien entendu que le corps de la fonction soit programmé de sorte de prévoir cette éventualité). Il existe deux fonctions Matlab utiles pour gérer cette situation : `nargin` qui retourne le nombre de variables d'entrée utilisés lors de l'appel et `nargout` qui retourne le nombre de variables de sortie prévues lors de l'appel. Voici un petit exemple venant illustrer ces possibilités.

```
function [A,r] = matale(T,m,n)

% Construit une matrice A de m lignes et n colonnes ayant des elements

% entiers generes de maniere aleatoire entre 0 et T.
% Calcule le rang de la matrice si l'appel est effectue avec 2 arguments

% de sortie.
% Si la matrice est carree, le parametre n peut etre omis.
%
% Appels :
%       [A,r] = Matale(T,m,n)
%       [A,r] = Matale(T,m)
%           A = Matale(T,m,n)
%           A = Matale(T,m)

if nargin == 2
  A = fix(T*rand(m));
else
  A = fix(T*rand(m,n));
end

if nargout == 2
 r = rank(A);
end
```

Dans cet exemple, on gère les variables d'entrée de la fonction de sorte de ne pas avoir besoin de donner lors de l'appel le nombre de lignes et de colonnes si la matrice est carrée. On gère aussi les variables de sortie afin de ne pas calculer le rang de la matrice si aucune variable de sortie pour le résultat n'est prévue lors de l'appel.

```
>> [A,r] = matale(20,3,4)
A =
    16    13    13    10
    10    16     7    14
     4     0    16     8
r =
     3
>> [A,r] = matale(20,3)
A =
```

```
    12     0    18
     5    14     9
     3     8     8
r =
     3
>> A = matale(20,3)
A =
     8     7     2
    17    16     4
     1     0     3
>>
```

Un point important concerne la gestion des variables entre le programme principal (ou le workspace) et les fonctions de l'utilisateur. Toutes les variables définies à l'intérieur d'une fonction sont des variables locales à cette fonction. La communication avec des variables du programme principal (ou du workspace) ou avec des variables d'autres fonctions se fait uniquement par les variables d'entrée et sortie de la fonction. Une alternative existe toutefois : il est possible de déclarer certaines variables comme des variables globales. Une variable globale peut être partagée entre un programme principal et plusieurs fonctions sans qu'il soit besoin de la spécifier parmi les variables d'entrée-sortie des différentes fonctions. On déclare une variable globale grâce au mot clé `global`. Par exemple pour déclarer la variable `numex` globale on écrit `global numex`. Attention, la déclaration `global numex` doit être reprise dans chaque fonction utilisant *numex* comme variable.

A.5.2 Opérateurs de comparaison et opérateurs logiques

Les opérateurs de comparaison sont :

`==`	:	égal à ($x == y$)
`>`	:	strictement plus grand que ($x > y$)
`<`	:	strictement plus petit que ($x < y$)
`>=`	:	plus grand ou égal à ($x >= y$)
`<=`	:	plus petit ou égal à ($x <= y$)
`~=`	:	différent de ($x \sim= y$)

Les opérateurs logiques sont :

`&`	:	et ($x \& y$)
`|`	:	ou ($x \| y$)
`~`	:	`non` ($\sim x$)

Les opérateurs de comparaison et les opérateurs logiques sont utilisés essentiellement dans les instructions de contrôle.

A.5.3 Instructions de contrôle

Les instructions de contrôle sous Matlab sont très proches de celles existant dans d'autres langages de programmation.

A.5.3.1 Boucle FOR : parcours d'un intervalle

Une première possibilité pour exécuter une séquence d'instructions de manière répétée consiste à effectuer une boucle pour les valeurs d'un indice, incrémenté à chaque itération, variant entre deux bornes données. Ce processus est mis en oeuvre par la boucle `for`.
Syntaxe :

```
for indice=borne_inf :borne_sup

séquence d'instructions

end
```

où
– `indice` est une variable appelée l'indice de la boucle;
– `borne_inf` et `borne_sup` sont deux constantes réelles (appelées paramètres de la boucle);
– `séquence d'instructions` est le traitement à effectuer pour les valeurs d'indices variant entre `borne_inf` et `borne_sup` avec un incrément de 1. On parle du corps de la boucle.

Interprétation :
Si `borne_inf` est plus petit ou égal à `borne_sup`, le traitement séquence d'instructions est exécuté `borne_sup - borne_inf` fois, pour les valeurs de la variable indice égales à `borne_inf`, `borne_inf+1`, ..., `borne_sup`. Si borne_inf est strictement plus grand que `borne_sup`, on passe à l'instruction qui suit immédiatement l'instruction de fin de boucle `end`.
Remarque :
L'indice de boucle ne prend pas nécessairement des valeurs entières. D'autre part il n'est pas nécessaire que l'indice de la boucle apparaisse dans le corps de la boucle; par contre il est interdit de modifier sa valeur s'il apparaît. Il est possible d'imbriquer des boucles mais elles ne doivent pas se recouvrir. On peut utiliser un incrément (pas) autre que 1 (valeur par défaut). La syntaxe est alors `borne_inf : pas : borne_sup`. Le pas peut être négatif. Attention à bien gérer la borne supérieure! Voici un exemple venant illustrer les possibilités de variations de l'indice de la boucle.

```
>> for r=1.1 :-0.1 :0.75
     disp(['r = ', num2str(r)]);
   end
r = 1.1
r = 1
r = 0.9
r = 0.8
>>
```

Voici un exemple d'utilisation d'une boucle pour calculer $n!$ (le lecteur attentif sait calculer $n!$ plus simplement, par exemple en exécutant `prod([1 :n])`).

```
>> n = 4;
>> nfac = 1;
>> for k = 1 :n
     nfac = nfac*k;
   end
>> nfac
nfac =
    24
>>
```

A.5.3.2 Boucle WHILE : tant que . . . faire

Une seconde possibilité pour exécuter une séquence d'instructions de manière répétée consiste à effectuer une boucle tant qu'une condition reste vérifiée. On arrête de boucler dès que cette condition n'est plus satisfaite. Ce processus est mis en oeuvre par la boucle `while`.
Syntaxe :

```
while expression logique

séquence d'instructions

end
```

où
– `expression logique` est une expression dont le résultat peut être vrai ou faux;
– `séquence d'instructions` est le traitement à effectuer tant que expression logique est vraie.
Interprétation :
Tant que `expression logique` est vraie le traitement `séquence d'instructions` est exécuté sous forme d'une boucle. Lorsque `expression logique` devient faux, on passe à l'instruction qui suit immédiatement l'instruction de fin de boucle `end`.
Remarque :
`expression logique` est en général le résultat d'un test (par exemple `i < Imax`) ou le résultat d'une fonction logique (par exemple `all(x)`). Il est impératif que le traitement de la séquence d'instructions agisse sur le résultat de expression logique sans quoi on boucle indéfiniment.
Voici comment calculer $n!$ avec une boucle `while` :

```
>> n = 4;
>> k = 1; nfac = 1;
>> while k <= n
     nfac = nfac*k;
     k = k+1;
   end
>> nfac
nfac =
    24
>>
```

A.5.3.3 L'instruction conditionnée IF

On a parfois besoin d'exécuter une séquence d'instructions seulement dans le cas où une condition donnée est vérifiée au préalable. Différentes formes d'instruction conditionnée existent sous Matlab.
L'instruction conditionnée la plus simple a la forme suivante :
Syntaxe :

```
If expression logique

séquence d'instructions

end
```

où

– `expression logique` est une expression dont le résultat peut être vrai ou faux;
– `séquence d'instructions` est le traitement à effectuer si expression logique est vraie.

Interprétation :
La séquence d'instructions n'est exécutée que si le résultat de l'évaluation de l'expression logique est vraie (c'est-à-dire vaut 1). Dans le cas contraire on exécute l'instruction qui suit le mot clé end. Dans le cas où l'expression logique est vraie, après exécution de la séquence d'instructions on reprend le programme à l'instruction qui suit le mot clé `end`.

Remarque :
Contrairement à certains langages de programmation, il n'y a pas de mot clé `then` dans cette instruction conditionnée. Notez également que la marque de fin de bloc conditionné est le mot clé `end` et non pas `endif`.
Il existe une séquence conditionnée sous forme d'alternatives :

Syntaxe :

```
If expression logique

séquence d'instructions 1

else

séquence d'instructions 2

end
```

où
– `expression logique` est une expression dont le résultat peut être vrai ou faux;
– `séquence d'instructions 1` est la séquence d'instructions à exécuter dans le cas où expression logique est vraie et `séquence d'instructions 2` est la séquence d'instructions à exécuter dans le cas où expression logique est faux.

Interprétation :
Si expression logique est vraie la séquence d'instructions 1 est exécutée, sinon c'est la séquence d'instructions 2 qui est exécutée. Le déroulement du programme reprend ensuite à la première instruction suivant le mot clé `end`.
Il est bien entendu possible d'imbriquer des séquences d'instructions conditionnées (au sens où la séquence d'instruction conditionnée contient des séquences d'instructions conditionnées). Pour une meilleure lisibilité, il est recommandé d'utiliser des indentations afin de mettre en évidence l'imbrication des séquences d'instructions conditionnées.
Il est possible d'effectuer un choix en cascade :

Syntaxe :

```
If expression logique 1

séquence d'instructions 1

elseif expression logique 2

séquence d'instructions 2
...

elseif expression logique N
```

```
séquence d'instructions N

else

séquence d'instructions par défaut

end
```

Interprétation :
Si `expression logique 1` est vraie la `séquence d'instructions 1` est exécutée et le programme reprend ensuite à la première instruction suivant le mot clé `end`, sinon si `expression logique 2` est vraie la `séquence d'instructions 2` est exécutée et le programme reprend ensuite à la première instruction suivant le mot clé `end`, etc. Si aucune des expressions logiques 1 à N n'est vraie alors séquence d'instructions par défaut est exécutée.
Remarque :
Attention à ne pas laisser d'espace entre `else` et `if`; le mot clé est `elseif`.
On utilise fréquemment un choix en cascade lors d'initialisation de données. Par exemple, on initialise une matrice A en fonction de la valeur d'une variable *numex* (numéro d'exemple) de la manière suivante :

```
if numex == 1
 A = ones(n);
elseif numex == 2
 A = magic(n);
elseif numex == 3 | numex == 4
 A = rand(n);
else
 error('numero d''exemple non prevu ...');
end
```

A.5.3.4 Choix ventilé, l'instruction switch

Une alternative à l'utilisation d'une séquence d'instructions conditionnées pour effectuer un choix en cascade existe. Il s'agit de l'instruction `switch`.
Syntaxe :

```
switch var

case cst_1,

séquence d'instructions 1

case cst_2,

séquence d'instructions 2
...

case cst_N,

séquence d'instructions N

otherwise
```

```
séquence d'instructions par défaut

end
```

où
– `var` est une variable numérique ou une variable chaîne de caractères;
– `cst_1, ..., cst_N`, sont des constantes numérique ou des constantes chaîne de caractères;
– `séquence d'instructions i` est la séquence d'instructions à exécuter si le contenu de la variable var est égal à la constante `cst_i` (`var==cst_i`).

Interprétation :
Si la variable `var` est égale à l'une des constantes `cst_1, ..., cst_N`, (par exemple `cst_i`) alors la séquence d'instructions correspondante (ici séquence d'instructions i) est exécutée. Le programme reprend ensuite à la première instruction suivant le mot-clé `end`. Si la variable `var` n'est égale à aucune des constantes la séquence d'instructions par défaut est exécutée.

Remarque :
La variable `var` doit bien entendu être du même type que les constantes `cst_1, ..., cst_N`. Il n'est pas nécessaire de prévoir un cas par défaut (bien que cela soit préférable). S'il n'y a pas de cas par défaut et si la variable `var` n'est égale à aucune des constantes, alors le programme continue à la première instruction suivant le mot-clé `end`.
Il est possible de regrouper plusieurs cas si la séquence d'instructions à exécuter est la même pour ces différents cas. La syntaxe est alors,

```
case {cst_k , cst_l , ...}

séquence d'instructions commune
```

Reprenons l'exemple où l'on souhaite initialiser une matrice A en fonction de la valeur prise par une variable numérique numex (numéro d'exemple). En utilisant un choix ventilé on obtient :

```
function  A = initA(n,numex)

switch numex
  case 1,
    A = ones(n)
  case 2,
    A = magic(n);
  case {\{}3,4{\}},
    A = rand(n);
  otherwise
    error('numero d''exemple non prevu ...');
end
```

Voici un exemple de choix ventilé portant sur une variable de type chaîne de caractères.

```
rep = input('Votre reponse (oui, non, chepas) :');

switch rep
  case {'oui','o'},
    disp('bravo ...');
  case {'non','n'}
    disp('perdu ...');
```

```
  case 'chepas'
    disp('c''est pourtant facile ...');
end
```

A.5.3.5 Interruption d'une boucle de contrôle

Il est possible de provoquer une sortie prématurée d'une boucle de contrôle. L'instruction `break` permet de sortir d'une boucle `for` ou d'une boucle `while`. L'exécution se poursuit alors séquentiellement à partir de l'instruction suivant le mot clé end fermant la boucle. En cas de boucles imbriquées, on interrompt seulement l'exécution de la boucle intérieure contenant l'instruction `break`. L'instruction `return` provoque un retour au programme appelant (ou au clavier). Les instructions suivant le `return` ne sont donc pas exécutées. L'instruction `return` est souvent utilisée conjointement avec une instruction conditionnée par exemple pour tester dans le corps d'une fonction si les paramètres d'entrée ont les valeurs attendues.
L'instruction `error` permet d'arrêter un programme et d'afficher un message d'erreur. La syntaxe est `error(' message d''erreur ')`. L'instruction `warning` permet d'afficher un message de mise en garde sans suspendre l'exécution du programme. La syntaxe est `warning(' message de mise e`
Il est possible d'indiquer à Matlab de ne pas afficher les messages de mise en garde d'un programme en tapant `warning off` dans la fenêtre de commandes. On rétablit l'affichage en tapant `warning on`.
On peut ainsi améliorer la fonction matale de la manière suivante :

```
function [A,rang] = matale(T,m,n)
% Construit une matrice A de m lignes et n colonnes ayant des elements

% entiers generes de maniere aleatoire entre 0 et T.
% Calcule le rang de la matrice si l'appel est effectue avec 2 arguments

% de sortie.
% Si la matrice est carree, le parametre n peut etre omis.
%
% Appels :
%       [A,r] = Matale(T,m,n)
%       [A,r] = Matale(T,m)
%           A = Matale(T,m,n)
%           A = Matale(T,m)

% si la fonction est appelee avec un nombre d'arguments d'entree
% different de 2 ou 3, on arrete tout ...
if nargin $\sim $= 2 {\&}  nargin $\sim $= 3,
   error(' La fonction matale doit avoir 2 ou 3 arguments d''entree ');

end

if nargin == 2
  A = fix(T*rand(m));
else
  A = fix(T*rand(m,n));
end

if nargout == 2
```

```
  rang = rank(A);
  if nargin == 2,
    rangx = m;
  else
    rangx = min([m,n]);
  end
  if rang $\sim $= rangx, warning(' Le rang n''est pas maximum '); end;

end
```

On obtient alors les messages suivants :

```
>> A = matale(3);
??? Error using ==> matale
 La fonction matale doit avoir 2 ou 3 arguments d'entree

>> A = matale(20,3)
A =
     8    18     8
    12    14    18
    15     3    18
>> [A,r] = matale(20,3)

Warning :  Le rang n'est pas maximum
> In /home0/maths/balac/DOCMATLAB/matale.m at line 34
A =
     1     4     3
    10    15    11
     3    12     9
r =
     2
>>
```

La commande pause permet d'interrompre l'exécution du programme. L'exécution normale reprend dès que l'utilisateur enfonce une touche du clavier. L'instruction `pause(n)` suspend l'exécution du programme pendant n secondes.

A.5.4 Un exemple complet

Une technique de construction de carrés magiques d'ordre impair a été proposée par Manuel Moschopoulos au début du XIV siècle. Cette technique est décrite dans [Histoire d'Algorithmes, du caillou à la puce, J.L. Chabert éditeur, Belin 1994].
Notons $l(x)$ le numéro de la ligne et $c(x)$ le numéro de la colonne du carré sur lequel se trouve l'entier x. Partant d'un carré d'ordre impair $n = 2k + 1$, la technique de Moschopoulos peut se formuler comme suit :
Initialisation de l'algorithme en plaçant l'unité dans la case immédiatement au dessous de la case centrale, autrement dit à l'intersection de la $(k + 1)$éme colonne et de la $(k + 2)$ème ligne:

$$l(1) = k + 2$$
$$c(1) = k + 1$$

Connaissant la position $(l(x), c(x))$ de l'entier x, on place l'entier $x+1$ suivant les règles suivantes :
si x n'est pas un multiple de n, alors

$$l(x+1) = 1 + l(x) modulo(n)$$
$$c(x+1) = 1 + c(x) modulo(n)$$

si x est un multiple de n, alors

$$l(x+1) = 2 + l(x) modulo(n)$$
$$c(x+1) = c(x) modulo(n)$$

Dans ces règles pour la prise du modulo, le système de résidus que l'on considère est 1, 2, ..., n et non pas 0, 1, ..., $n-1$.
La fonction `magik` met en oeuvre la méthode de Moschopoulos.

```
function M = magik(n)
%
% Calcule le carre magique d'ordre n selon la methode
% de Moschopoulous.
%
% Entree :
%  n : ordre du carre (entier impair)
% Sortie :
%  M : le carre magique
%

if rem(n,2) == 0,
  msg = ['la methode de Moschopoulous ne construit que des carres' ...
       ,' d''ordre impair'];
  error(msg)
end

k = (n-1)/2;
l(1) = k+2;
c(1) = k+1;
M(l(1),c(1)) = 1;

for x = 2 :n.\ 2
  [l(x),c(x)] = pos(x-1,l(x-1),c(x-1),n);
  M(l(x),c(x)) = x;
% ou plus simplement M(pos(x,l(x-1),c(x-1))) = x;
end
```

La fonction utilise la fonction `pos`. Cette dernière fonction peut soit être écrite à la suite de la fonction `magik` si l'on dispose de la version 5 de Matlab (dans ce cas il s'agira d'une sous-fonction qui ne sera visible que de la fonction `magik`) soit être sauvegardée dans un fichier `pos.m`.

```
function [ly,cy] = pos(x,lx,cx,n)
%
% Retourne la position (indice de ligne ly et indice de colonne cy)
```

```
% dans le carre magique d'ordre n de l'entier y = x+1 selon la
% regle de Moschopoulous.
%
% Entree :
%  n : ordre du carre (entier impair)
%  x : l'entier considere
% lx : indice de ligne de l'entier x dans le carre magique
% cx : indice de colonne de l'entier x dans le carre magique
%
% Sortie :
%  ly : indice de ligne de l'entier y=x+1 dans le carre magique
%  cy : indice de colonne de l'entier y=x+1 dans le carre magique

%

if rem(x,n) == 0
  ly = modulo(2+lx,n);
  cy = modulo(cx,n);
else
  ly = modulo(1+lx,n);
  cy = modulo(1+cx,n);
end
```

Voici quelques exemples d'utilisation de la fonction `magik`. On vérifie que la somme des éléments des différentes lignes ainsi que la somme des éléments des différentes colonnes sont bien constantes. Pour calculer la somme des éléments diagonaux c'est un peu plus compliqué. On remarque que le carré magique construit diffère du carré magique retourné par la fonction Matlab incorporée magic.

```
>> magik(4)
??? Error using ==> magik
la methode de Moschopoulous ne construit que des carres d'ordre impair

>> magik(5)
ans =
    11    24     7    20     3
     4    12    25     8    16
    17     5    13    21     9
    10    18     1    14    22
    23     6    19     2    15
>> sum(magik(5),1)
ans =
    65    65    65    65    65
>>  sum(magik(5),2)
ans =
    65
    65
    65
    65
    65
```

```
>> magic(5)
ans =
    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
```

A.6 Graphisme

A.6.1 Gestion des fenêtres graphiques

Une instruction graphique ouvre une fenêtre dans laquelle est affiché le résultat de cette commande. Par défaut, une nouvelle instruction graphique sera affichée dans la même fenêtre et écrasera la figure précédente. On peut ouvrir une nouvelle fenêtre graphique par la commande `figure`. Chaque fenêtre se voit affecter un numéro `n`. Ce numéro est visible dans le bandeau de la fenêtre sous forme d'un titre. Le résultat d'une instruction graphique est par défaut affiché dans la dernière fenêtre graphique ouverte qui est la fenêtre graphique active. On rend active une fenêtre graphique précédemment ouverte en exécutant la commande `figure(n)`, où `n` désigne le numéro de la figure. La commande close permet de fermer la fenêtre graphique active. On ferme une fenêtre graphique précédemment ouverte en exécutant la commande `close(n)`, où `n` désigne le numéro de la figure. Il est également possible de fermer toutes les fenêtres graphiques en tapant `close all`. La figure représente la fenêtre graphique numéro 1 où est affiché le graphe de la fonction cosinus entre 0 et2π, résultat de la commande `fplot('cos',[0 2*pi])`. L'apparence de la fenêtre graphique peut varier légèrement suivant le système informatique utilisé.

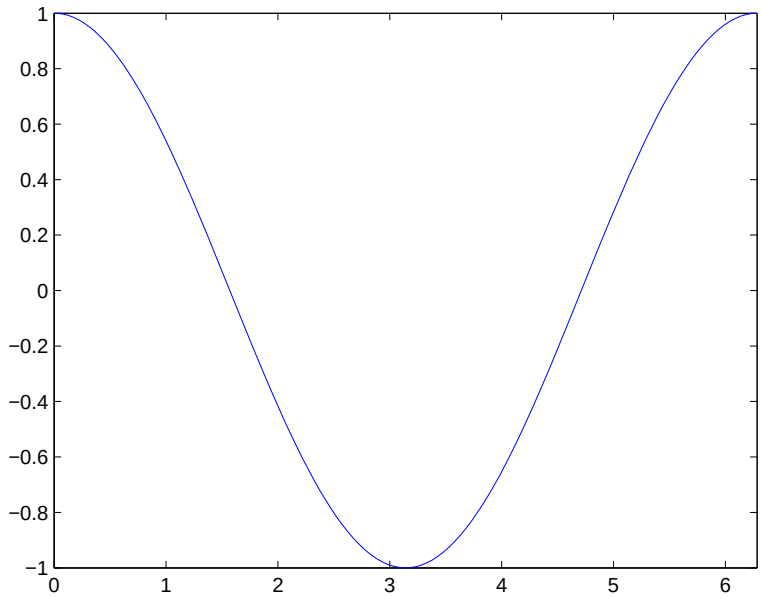


FIG. A.4 – Fenêtre graphique et résultat de la commande `fplot('cos',[0 2*pi])`

A.6.2 Graphisme 2D

A.6.2.1 Tracer le graphe d'une fonction ; la commande `fplot`

La commande `fplot` permet de tracer le graphe d'une fonction sur un intervalle donné. La syntaxe est :

```
fplot('nomf', [x_min , x_max])
```

où
– `nomf` est soit le nom d'une fonction Matlab incorporée, soit une expression définissant une fonction de la variable x, soit le nom d'une fonction utilisateur.
– `[x_min ,x_max]` est l'intervalle pour lequel est tracé le graphe de la fonction.
Illustrons par des exemples les trois façons d'utiliser la commande `fplot`. On obtient le graphe de la fonction incorporée sinus entre -2π et 2π par l'instruction :

```
fplot('sin',[-2*pi 2*pi])
```

Pour tracer le graphe de la fonction $h(x) = x\sin(x)$ entre -2π et2π, on peut définir la fonction utilisateur h dans le fichier `h.m` de la manière suivante (attention de bien lire `x.*sin(x)` et non pas `x*sin(x)`) :

```
function y=h(x)
y=x.*sin(x);
```

On obtient alors le graphe de la fonction h par l'instruction :

```
fplot('h', [-2*pi 2*pi])
```

L'autre façon de procéder est d'exécuter l'instruction (là on a le choix entre écrire `x.*sin(x)` ou `x*sin(x)`) :

```
fplot('x*sin(x)', [-2*pi 2*pi])
```

Dans les deux cas on obtient le dessin représenté à la figure A.5.

Il est possible de tracer plusieurs fonctions sur la même figure. Il faut pour cela utiliser la commande `fplot` de la manière suivante :

```
fplot('[nomf_1 , nomf_2 , nomf_3]', [x_min , x_max])
```

où `nomf_1`, `nomf_2`, `nomf_3` est soit le nom d'une fonction Matlab incorporée, soit une expression définissant une fonction de la variable x, soit le nom d'une fonction utilisateur.
Il est également possible de gérer les bornes des valeurs en ordonnées. Pour limiter le graphe aux ordonnées comprises entre les valeurs `y_min` et `y_max` on passera comme second argument de la commande `fplot` le tableau `[x_min, x_max, y_min, y_max]`. Une autre possibilité pour gérer les bornes des valeurs en ordonnées est d'utiliser la commande `axis` après utilisation de la commande `fplot`. La syntaxe est `axis([x_min, x_max, y_min, y_max])`. Voici un exemple dont le résultat est affiché à la figure A.6 :

```
>> fplot('[sin(x)/x , cos(x)/x]', [-5, 5, -1, 1])
```

On comprend très vite l'intérêt de gérer les bornes des valeurs en ordonnées si l'on exécute la commande `fplot('cos(x)/x', [-5, 5])` pour tracer le graphe de la fonction $cos(x)/x$ entre -5 et 5. La figure, qui n'est pas vraiment esthétique, n'est pas reproduite ici.

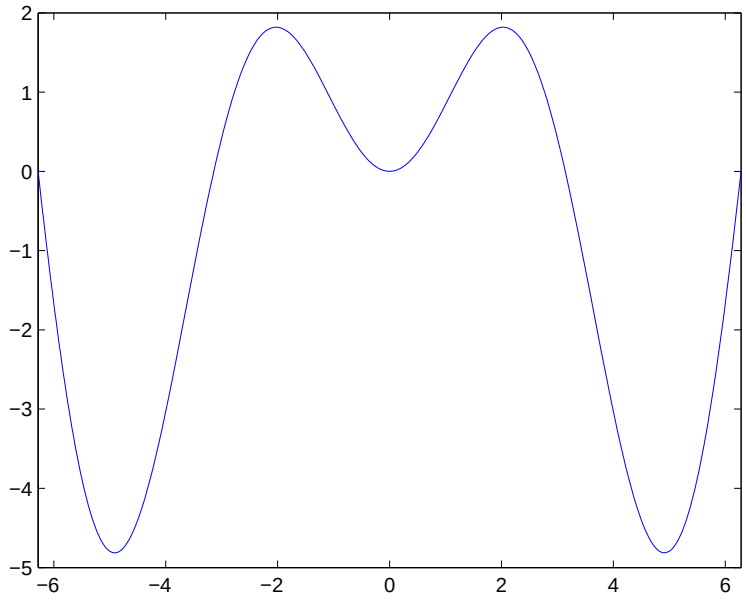


FIG. A.5 – Graphe de la fonction $h(x) = xsinx$ entre -2π et 2π

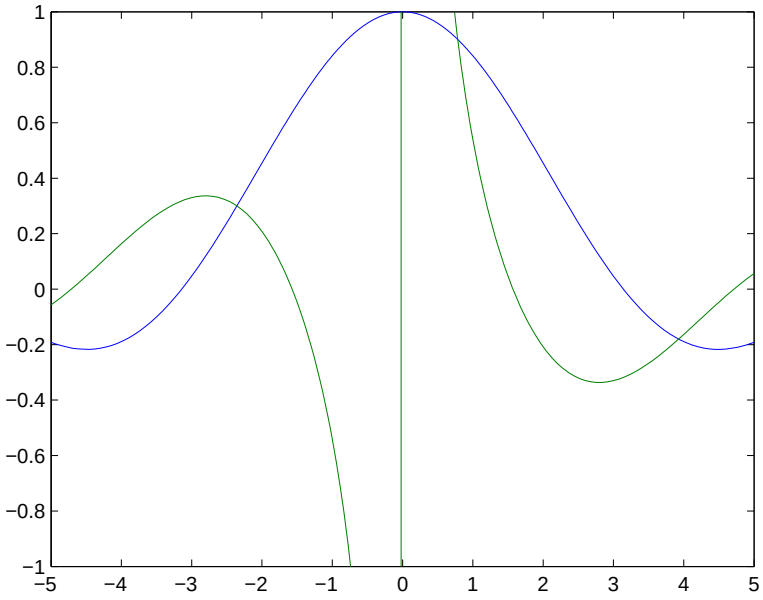


FIG. A.6 – Graphe des fonctions $cos(x)/x$ et $sin(x)/x$ entre -5 et 5.

A.6.2.2 La commande plot

La commande plot permet de tracer un ensemble de points de coordonnées $(x_i, y_i), i = 1, ..., N$. La syntaxe est `plot(x,y)` où x est le vecteur contenant les valeurs x_i en abscisse et y est le vecteur contenant les valeurs y_i en ordonnée. Bien entendu les vecteurs x et y doivent être de même dimension mais il peut s'agir de vecteurs lignes ou colonnes. Par défaut, les points (x_i, y_i) sont reliés entre eux par des segments de droites. Voici par exemple une autre façon de tracer le graphe de la fonction $h(x) = xsin(x)$ entre -2π et 2π,

```
>> x=[-2*pi :0.01 :2*pi]; y = x.*sin(x);
>> plot(x,y)
```

Essayez aussi :

```
>> x=[-2*pi :1 :2*pi]; y = x.*sin(x);
>> plot(x,y)
```

Dans cet exemple on a défini un vecteur x de valeurs équi-réparties entre -2π et2π (avec un pas de 0.01 dans le premier cas et de 1 dans le deuxième cas) et on a calculé l'image par la fonction h de ces valeurs (vecteur y). On affiche les points de coordonnées $(x(i), y(i))$.
On peut spécifier à Matlab quelle doit être la couleur d'une courbe, quel doit être le style de trait et/ou quel doit être le symbole à chaque point (x_i, y_i). Pour cela on donne un troisième paramètre d'entrée à la commande `plot` qui est une chaîne de 3 caractères de la forme `'cst'` avec `c` désignant la couleur du trait, `s` le symbole du point et `t` le type de trait. Les possibilités sont les suivantes :

`y`	jaune	`.`	point	`-`	trait plein
`m`	magenta	`o`	cercle	`:`	pointillé court
`c`	cyan	`x`	marque x	`--`	pointillé long
`r`	rouge	`+`	plus	`-.`	pointillé mixte
`g`	vert	`*`	étoile		
`b`	bleu	`s`	carré		
`w`	blanc	`d`	losange		
`k`	noir	`v`	triangle (bas)		
`^`	triangle (haut)				
`<`	triangle (gauche)				
`>`	triangle (droit)				
`p`	pentagone				
`h`	hexagone				

Les valeurs par défaut sont `c = b`, `s = .` et `t = -` ce qui correspond à un trait plein bleu reliant les points entre eux. Il n'est pas obligatoire de spécifier chacun des trois caractères. On peut se contenter d'en spécifier un ou deux. Les autres seront les valeurs par défaut. La commande grid permet d'obtenir un quadrillage de la figure, voir par exemple la figure A.6.
Il est possible de tracer plusieurs courbes sur la même figure en spécifiant plusieurs tableaux `x1, y1, x2, y2, ...,` comme paramètres de la commande plot. Si l'on souhaite que les courbes aient une apparence différente, on utilisera des options de couleurs et/ou de styles de traits distincts après chaque couple de vecteurs x, y.
Voici un exemple dont le résultat est affiché à la figure A.7. On trace sur l'intervalle $[-5, 5]$ la fonction $x^2 \times cos(x)$ en trait plein bleu et la fonction $x \times cos(x)$ en trait pointillé rouge.

```
>> x = [-5 :0.01 :5];
>> y = x.^2.*cos(x); z = x.*cos(x);
>> plot(x,y,'b-',x,z,'r :');
```

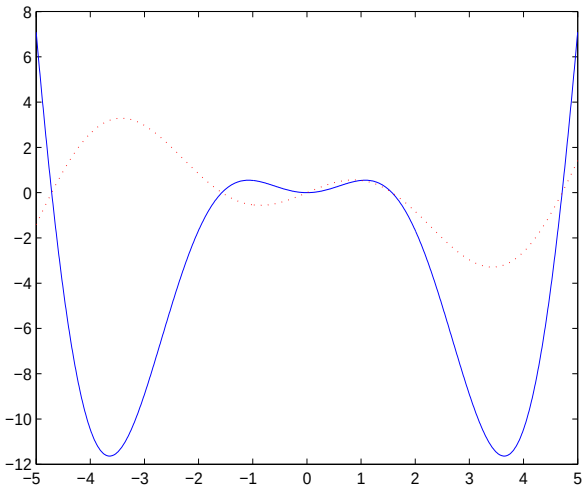


FIG. A.7 – Résultat de la commande `plot(x,y,'b-',x,z,'r :')`.

Voici un autre exemple amusant (la commande whitebg permet de passer la fenêtre graphique en inverse vidéo).

```
>> N=200;
>> x = rand(1,N); y = rand(1,N);
>> plot(x,y,'bd'), whitebg
```

A.6.2.3 La commande loglog

Si x et y sont deux vecteurs de même dimension, la commande $loglog(x, y)$ permet d'afficher le vecteur $log(x)$ contre le vecteur $log(y)$. La commande *loglog* s'utilise de la même manière que la commande plot. Voici un exemple dont le résultat est affiché à la figure A.8. Quelle est la pente de la droite?

```
>> x = [1 :10 :1000]; y = x.^3;
>> loglog(x,y)
```

A.6.3 Améliorer la lisibilité d'une figure

A.6.3.1 Légender une figure

Il est recommandé de mettre une légende à une figure. La commande `xlabel` permet de mettre un texte en légende sous l'axe des abscisses. La syntaxe est `xlabel(' légende ')` pour obtenir le mot légende en légende. La commande `ylabel` fait de même pour l'axe des ordonnées. La commande `title` permet de donner un titre à la figure. La syntaxe est `title(' le titre')` pour obtenir comme titre le titre. On peut aussi écrire un texte donné à une position précise

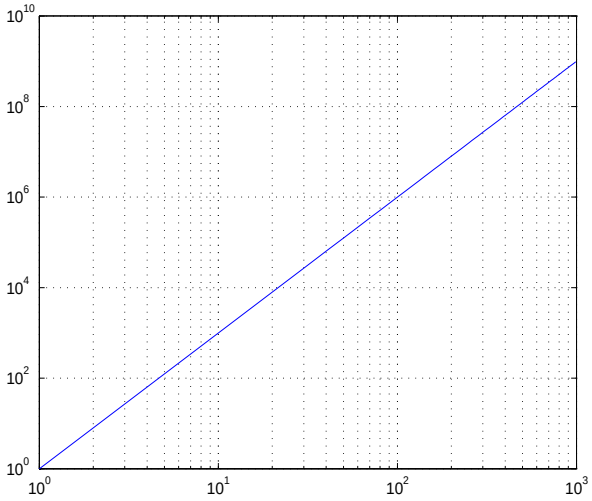


FIG. A.8 – Résultat de la commande `loglog(x,y)`.

sur la figure grâce à la commande `text`. La syntaxe est `text(posx,posy,' un texte ')` où `posx` et `posy` sont les coordonnées du point (dans le système associé au dessin) où doit débuter l'écriture du texte un texte. La commande `gtext` permet quant à elle de placer le texte à une position choisie sur la figure à l'aide de la souris. La syntaxe est `gtext(' un texte ')`. Une mire, que l'on déplace en utilisant la souris, apparaît. Il suffit d'un clic-souris pour que le texte apparaisse à la position sélectionnée.
Il est possible avec ces commandes d'afficher une valeur contenue dans une variable au milieu de texte. Pour cela on construit un tableau de type chaîne de caractères en convertissant la valeur contenue dans la variable en une chaîne de caractères grâce à la commande num2str. Par exemple, supposons que la variable numex contienne le numéro de l'exemple traité, disons 5. On obtiendra pour titre de la figure `Exemple numero 5` par l'instruction :

```
title(['Exemple numero ', num2str(numex)]).
```

L'exemple suivant dont le résultat est affiché à la figure A.9 illustre l'utilisation des différentes commandes permettant de légender une figure.

```
>> P = 5;
>> t = [0 :.01 :2];
>> c = 12*exp(-2*t) - 8*exp(-6*t);
>> plot(t,c); grid
>> xlabel('temps en minutes')
>> ylabel('concentation en gramme par litre')
>> title(['evolution de la concentration du produit ', num2str(P), ...
    ' au cours du temps '])
>> gtext('concentration maximale')
```

A.6.3.2 Afficher plusieurs courbes dans une même fenêtre

Il est possible d'afficher plusieurs courbes dans une même fenêtre graphique grâce à la commande `hold on`. Les résultats de toutes les instructions graphiques exécutées après appel à la commande `hold on` seront superposés sur la fenêtre graphique active. Pour rétablir la situation

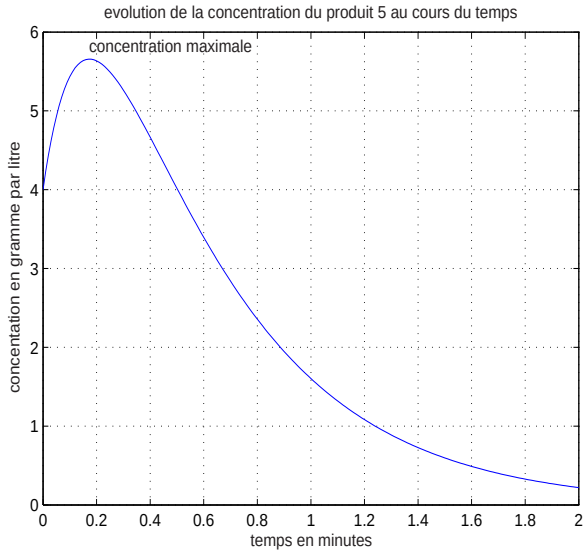


FIG. A.9 – Exemple de légende d'une figure.

antérieure (le résultat d'une nouvelle instruction graphique remplace dans la fenêtre graphique le dessin précédent) on tapera `hold off`. Voici un exemple d'utilisation de la commande hold on. Le résultat est présenté à la figure A.10.

```
>> e = exp(1);
>> figure
>> hold on
>> fplot('exp',[-1 1])
>> fplot('log',[1/e e])
>> plot([-1 :0.01 :e],[-1 :0.01 :e])
>> grid
>> hold off
```

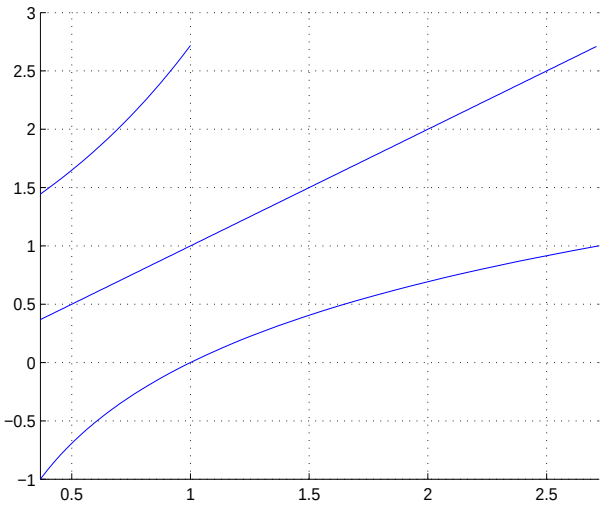


FIG. A.10 – Superposition de plusieurs courbes dans la même fenêtre graphique.

On dispose donc de deux façons de superposer plusieurs courbes sur une même figure. On peut soit donner plusieurs couples de vecteurs abscisses/ordonnées comme argument de la commande `plot`, soit avoir recours à la commande hold on. Suivant le contexte on privilégiera l'une de ces solutions plutôt que l'autre.
Il est possible de décomposer une fenêtre en sous-fenêtres et d'afficher une figure différente sur chacune de ces sous-fenêtres grâce à la commande subplot. La syntaxe est :

```
subplot(m,n,i)
```

où
– `m` est le nombre de sous-fenêtres verticalement;
– `n` est le nombre de sous-fenêtres horizontalement;
– `i` sert à spécifier dans quelle sous-fenêtre doit s'effectuer l'affichage.
Les fenêtres sont numérotées de gauche à droite et de haut en bas. L'exemple suivant dont le résultat apparaît à la figure A.11 illustre l'utilisation de la commande `subplot`.

```
>> figure
>> subplot(2,3,1), fplot('cos',[0 4*pi]), title('cosinus'), grid
>> subplot(2,3,2), fplot('sin',[0 4*pi]), title('sinus'), grid
>> subplot(2,3,3), fplot('tan',[-pi/3 pi/3]), title('tangente'), grid
>> subplot(2,3,4), fplot('acos',[-1 1]), title('arc-cosinus'), grid
>> subplot(2,3,5), fplot('asin',[-1 1]), title('arc-sinus'), grid
>> subplot(2,3,6), fplot('atan',[-sqrt(3) sqrt(3)]), title('arc-tangente'),
grid
>>
```

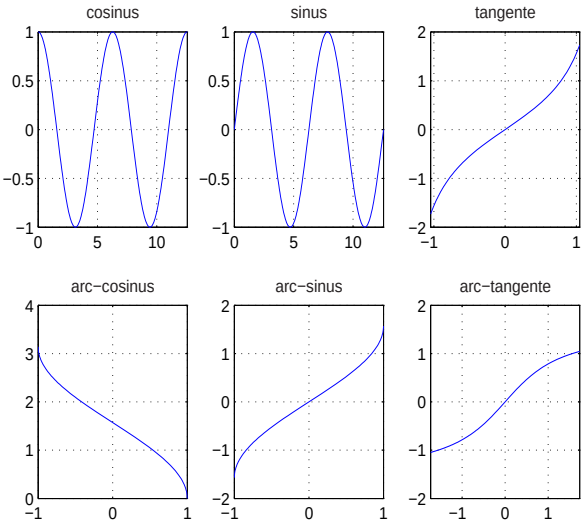


FIG. A.11 – Fenêtre graphique décomposée en sous fenêtres.

A.6.3.3 Sauvegarder une figure

La commande `print` permet de sauvegarder la figure d'une fenêtre graphique dans un fichier sous divers formats d'images. La syntaxe de la commande `print` est :

```
print -f<num> -d<format> <nomfic>
```

où
– `<num>` désigne le numéro de la fenêtre graphique. Si ce paramètre n'est pas spécifié, c'est la fenêtre active qui est prise en compte.
– `<nomfic>` est le nom du fichier dans lequel est sauvegardée la figure. Si aucune extension de nom n'est donnée, une extension par défaut est ajoutée au nom du fichier en fonction du format choisi (`.ps` pour du PostScript, `.jpg` pour du jpeg, par exemple).
– `<format>` est le format de sauvegarde de la figure. Ces formats sont nombreux. On pourra obtenir la liste complète en tapant help plot. Les principaux sont :

`ps`	:	PostScript noir et blanc
`psc`	:	PostScript couleur
`eps`	:	PostScript Encapsulé noir et blanc
`epsc`	:	PostScript Encapsulé couleur
`jpeg`	:	Format d'image JPEG
`tiff`	:	Format d'image TIFF

A.6.4 Graphisme 3D

A.6.4.1 Tracer les lignes de niveau d'une fonction de deux variables

La commande `contour` permet de tracer les lignes de niveau d'une fonction de deux variables réelles. Cette fonction peut être définie par une expression Matlab (par exemple `x.*exp(-x.^2-y.^2)`), ou être définie comme une fonction utilisateur. Pour tracer les lignes de niveau de la fonction $g(x, y)$ pour $x_{min} < x < x_{max}$ et $y_{min} < y < y_{max}$ on procède de la manière suivante :
– création d'un maillage, de maille de longueur h, du domaine $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$ grâce à la commande `meshgrid` :
`[X,Y] = meshgrid(x_min :h :x_max, y_min :h :y_max)`
– évaluation de la fonction aux noeuds de ce maillage, soit par appel à la fonction utilisateur définissant la fonction : $Z = g(X, Y)$ soit directement en définissant la fonction par une expression Matlab.
– Affichage des lignes de niveau grâce à la commande contour : `contour(X,Y,Z)`.
Ainsi pour tracer les lignes de niveau de la fonction $f(x, y) = xe^{-(x2+y2)}$ sur le domaine $[-2, 2] \times [-2, 2]$ en prenant un maillage de maille de longueur $h = 0.2$, on exécute :

```
>> [X,Y] = meshgrid(-2 :.2 :2, -2 :.2 :2);
>> Z = X.*exp(-X.^2-Y.^2);
>> contour(X,Y,Z)
```

On peut également écrire une fonction utilisateur g.m,

```
function x3 = g(x1,x2)
x3 =  x1.*exp(-x1.\ 2-x2.\ 2);
```

et exécuter

```
>> [X,Y] = meshgrid(-2 :.2 :2, -2 :.2 :2);
>> Z = g(X,Y);
>> contour(X,Y,Z)
```

Dans les deux cas on obtient le résultat présenté à la figure A.12.

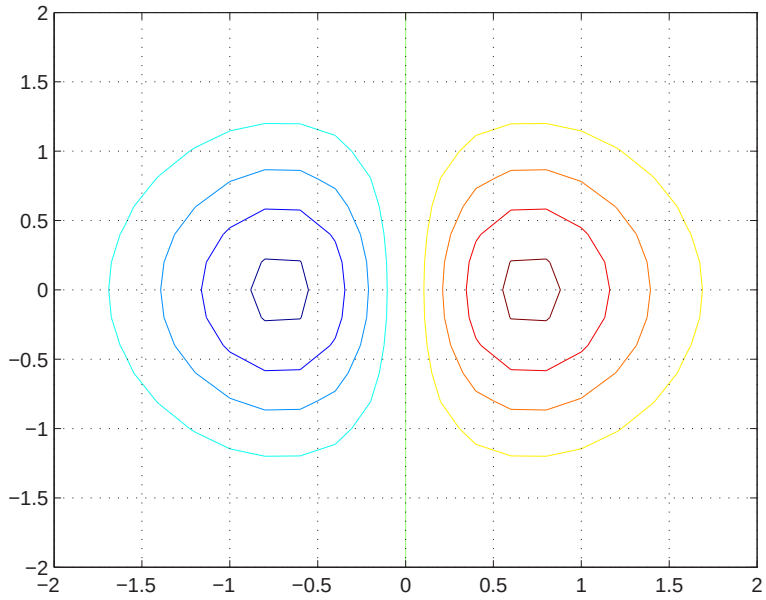


FIG. A.12 – Exemple de visualisation des lignes de niveau par la commande `contour`.

Le nombre de lignes de niveau est déterminé de manière automatique à partir des valeurs extrêmes prises par la fonction sur le domaine considéré. Pour imposer le nombre n de lignes de niveau à afficher, il suffit d'appeler la fonction `contour` avec la valeur n comme quatrième paramètre, `contour(X,Y,Z,n)`.
Il existe deux manières d'afficher les valeurs des lignes de niveau sur la figure. Si l'on souhaite afficher les valeurs pour toutes les lignes de niveau, on utilise la commande `clabel` de la manière suivante :

```
>> [C,h] = contour(X,Y,Z,n)
>> clabel(C,h)
```

Si l'on souhaite afficher uniquement les valeurs de quelques lignes de niveau, on utilise la commande `clabel` de la manière suivante :

```
>> [C,h] = contour(X,Y,Z,n)
>> clabel(C,h,'manual')
```

On peut alors grâce à la souris sélectionner les lignes de niveau pour lesquelles on souhaite afficher la valeur. Ainsi pour tracer 30 lignes de niveau de la fonction $z = (x-1)^2+10(x^2-y)^2$sur le domaine $[-1,1]\times[-1,1]$ et pour choisir à l'aide de la souris les lignes de niveau pour lesquelles l'iso valeur doit être affichée, on exécute :

```
>> [X,Y] = meshgrid(-2 :.2 :2, -2 :.2 :2);
>> Z = (X-1).^2 + 10*(X.^2-Y).^2;
>> [C,h] = contour(X,Y,Z,30);
>> clabel(C,h,'manual')
```

Le résultat est présenté à la figure **??**.

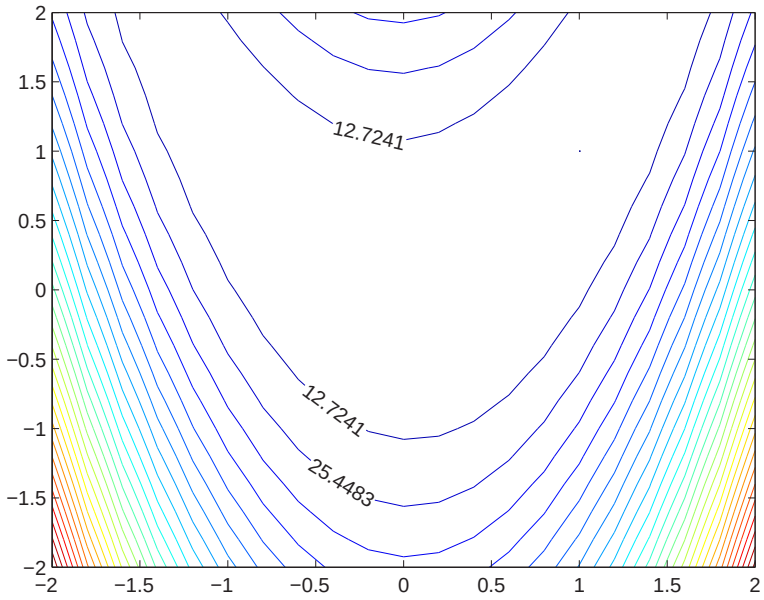


FIG. A.13 – Exemple de visualisation des lignes de niveau par la commande contour.

Il est possible de modifier la palette des couleurs en utilisant la commande `colormap`. En tapant `help graph3d` dans la fenêtre de contrôle Matlab, vous obtiendrez toutes les palettes de couleurs disponibles. A vous de les tester pour obtenir le meilleur effet (La commande `colormap(gray)` permet d'utiliser une palette en dégradé de gris, ce qui est très utile si l'on souhaite une impression de la figure sur une imprimante noir et blanc.
La commande `contourf` s'utilise de la même manière que la commande contour. Elle permet d'afficher, en plus des lignes de niveau, un dégradé continu de couleurs qui varie en fonction des valeurs prises par la fonction. La figure A.14 présente un exemple d'utilisation de la commande `contourf` obtenu de la manière suivante :

```
>> [X,Y] = meshgrid(-2 :.2 :2, -2 :.2 :2);
>> Z = (X-1).^2 + 10*(X.^2-Y).^2;
>> contourf(X,Y,Z,30);
>> colormap(cool);
```

A.6.4.2 Représenter une surface d'équation $z = g(x, y)$

La commande `mesh` permet de tracer une surface d'équation $z = g(x, y)$. La fonction g peut être définie directement par une expression Matlab ou être définie comme une fonction utilisateur. Pour tracer la surface d'équation $z = g(x, y)$ pour $x_{min} < x < x_{max}$ et $y_{min} < y < y_{max}$ on procède de la manière suivante :

- création d'un maillage, de maille de longueur h, du domaine $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$ grâce à la commande `meshgrid` :
  `[X,Y] = meshgrid(x_min :h :x_max, y_min :h :y_max).`
- évaluation de la fonction aux noeuds de ce maillage, soit par appel à la fonction utilisateur définissant la fonction : $Z = g(X, Y)$ soit directement en définissant la fonction par d'une expression Matlab.
- Affichage de la surface grâce à la commande `mesh` : `mesh(X,Y,Z)`.

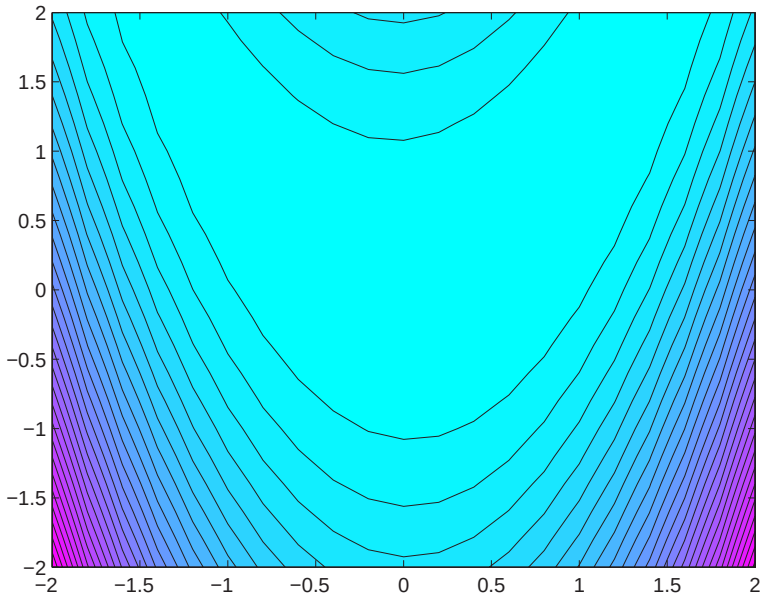


FIG. A.14 – Exemple de visualisation des lignes de niveau par la commande `contourf`.

Ainsi pour tracer la surface d'équation $z = xe^{-(x2+y2)}$ sur le domaine $[-2, 2] \times [-2, 2]$ avec un maillage de maillage de longueur $h = 0.2$, on exécute :

```
>> [X,Y] = meshgrid(-2 :.2 :2, -2 :.2 :2);
>> Z = X.*exp(-X.^2-Y.^2);
>> mesh(X,Y,Z)
```

Si la fonction est définie comme une fonction utilisateur dans le fichier `g.m`,

```
function x3 = g(x1,x2)
x3 =  x1.*exp(-x1.^2-x2.^2);
```

on exécute :

```
>> [X,Y] = meshgrid(-2 :.2 :2, -2 :.2 :2);
>> Z = g(X,Y);
>> contour(X,Y,Z)
```

Dans les deux cas on obtient le résultat présenté figure **??**.

Par défaut les valeurs extrêmes en z sont déterminées automatiquement à partir des extremums de la fonction sur le domaine spécifié. Il est possible de modifier ces valeurs (et également les valeurs extrêmes en abscisses et ordonnées) par la commande `axis` dont la syntaxe est

```
axis(x_min x_max y_min y_max z_min z_max)
```

Il est également possible de modifier le point de vision grâce à la commande `view`. La commande `view` a deux arguments qui sont l'angle de vision horizontal et l'angle de vision vertical en degré. Par défaut ces angles ont respectivement les valeurs -37.5° et 30°. La figure A.17 montre par exemple le résultat de l'instruction `view(37.5,30)`.

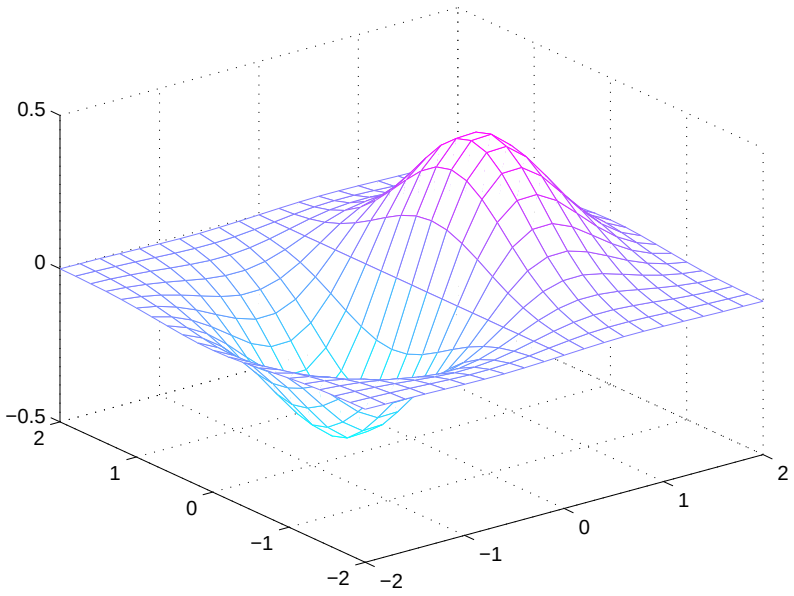


FIG. A.15 – Exemple de visualisation d'une surface d'équation $z = g(x, y)$ grâce à la commande `mesh`.

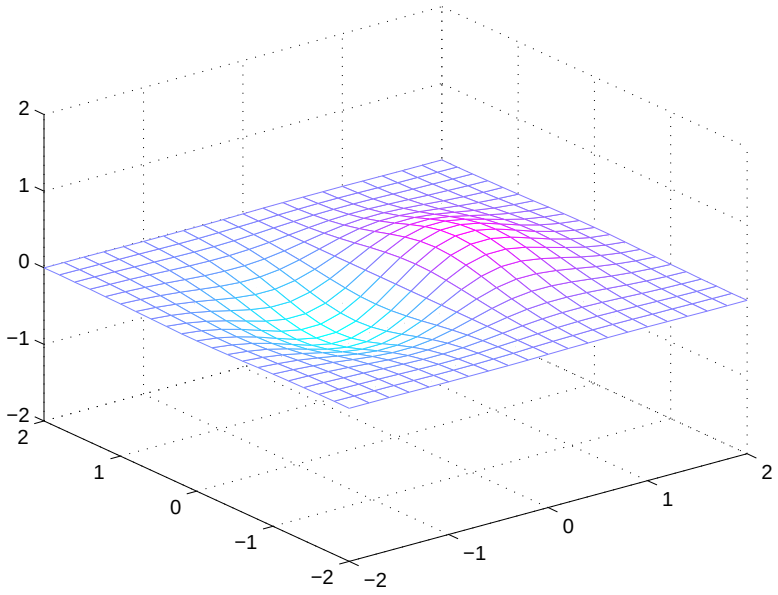


FIG. A.16 – Exemple d'utilisation de la commande `axis`.

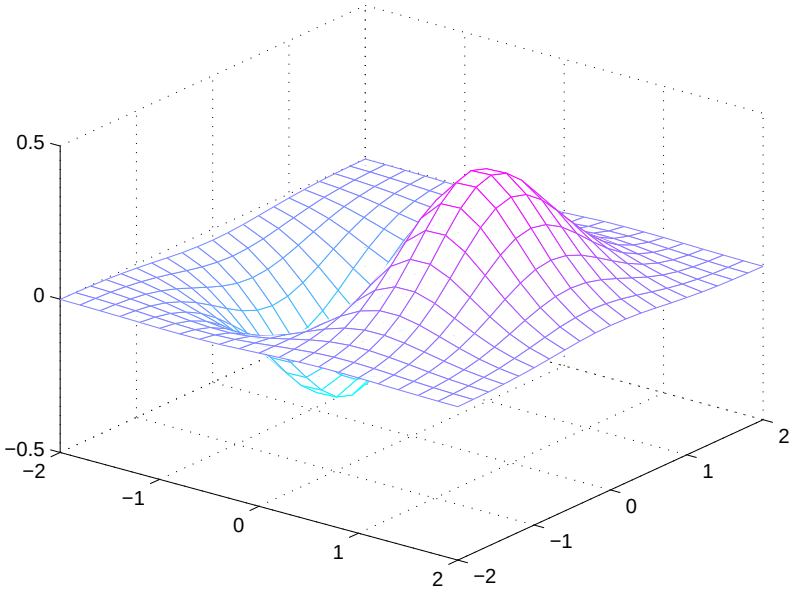


FIG. A.17 – Exemple de modification de l'angle de vision par la commande `view`.

Les commandes `meshc` et `meshz` s'utilisent de la même manière que la commande `mesh`. La commande meshc permet d'afficher des lignes de niveau sous la surface dans le plan $z = z_{min}$. La commande `meshz` permet de tracer une boite sous la surface. Un exemple d'utilisation de ces commandes est présenté aux figures A.18 et A.19 respectivement.

A.6.4.3 Représenter une surface paramétrée

La commande `surf` permet de tracer une surface paramétrée d'équations,

$$(E) = \begin{cases} x &= g_1(u,v) \\ y &= g_2(u,v) \\ z &= g_3(u,v) \end{cases}$$

La fonction $G = (g_1, g_2, g_3)$ peut être définie directement par une expression Matlab ou être définie comme une fonction utilisateur. Pour tracer la surface paramétrée d'équation (E) pour $u_{min} < u < u_{max}$ et $v_{min} < v < v_{max}$ on procède de la manière suivante :

- création d'un maillage, de maille de longueur h, du domaine $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$ grâce à la commande `meshgrid` :
  `[U,V] = meshgrid(u_min :h :u_max, v_min :h :v_max)`
- évaluation de la fonction aux noeuds de ce maillage, soit par appel à la fonction utilisateur définissant la fonction : `[X,Y,Z] = G(U,V)` soit directement en définissant la fonction par une expression Matlab.
- Affichage de la surface grâce à la commande `surf` : `surf(X,Y,Z)`.

Ainsi pour tracer la surface paramétrée d'équations :

$$(E) = \begin{cases} x &= v\cos u \\ y &= v\sin u \\ z &= 2u \end{cases}$$

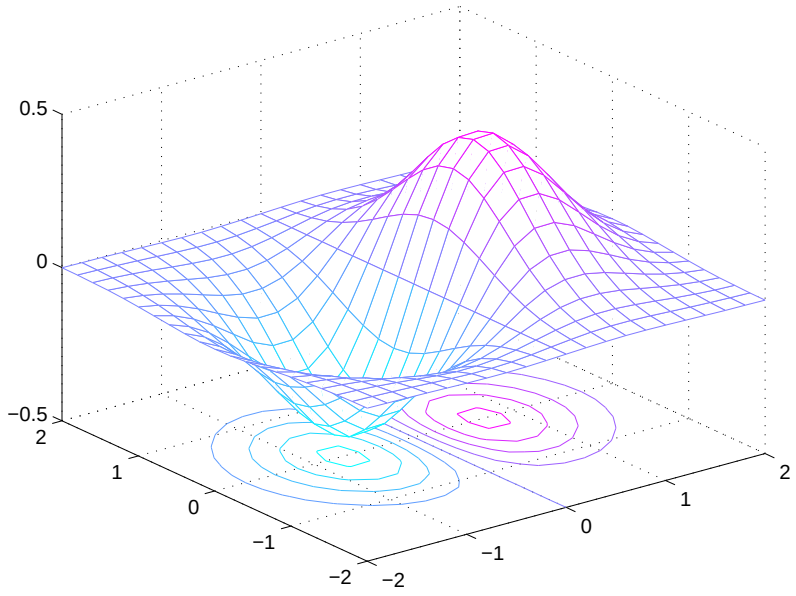


FIG. A.18 – Exemple de visualisation d'une surface d'équation $z = g(x, y)$ grâce au commandes `meshc`.

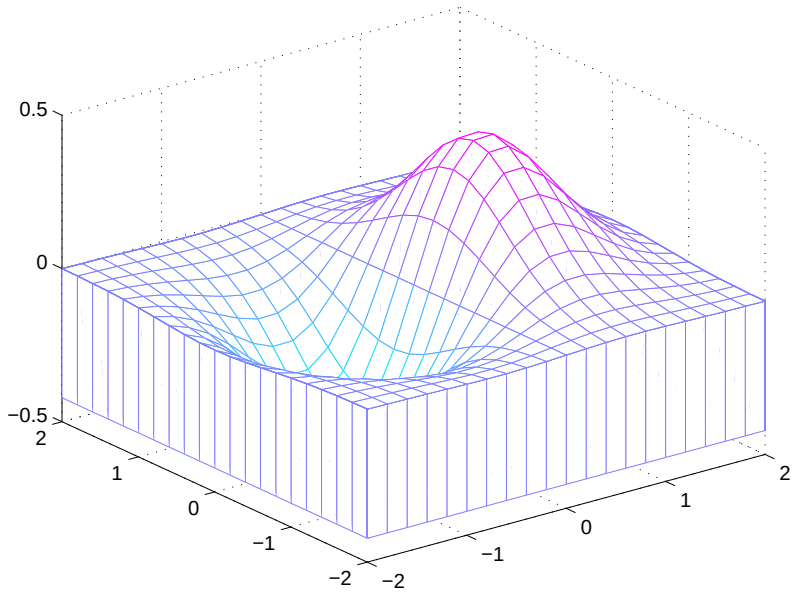


FIG. A.19 – Exemple de visualisation d'une surface d'équation $z = g(x, y)$ grâce au commandes `meshz`.

sur le domaine $[0, 2\pi] \times [0, 2]$ avec un maillage de maillage de longueur $h = 0.2$, on exécute :

```
>> [U,V] = meshgrid(0 :.2 :2*pi, 0 :.2 :2);
>> X = V.*cos(U);
>> Y = V.*sin(U);
>> Z = 2*U;
>> surf(X,Y,Z)
```

Si la fonction est définie comme une fonction utilisateur dans le fichier `G.m`,

```
function [x1, x2, x3] = G(u,v)
x1 = v.*cos(u);
x2 = v.*sin(u);
x3 = 2*u;
```

on exécute :

```
>> [U,V] = meshgrid(0 :.2 :2*pi, 0 :.2 :2);
>> [X,Y,Z] = G(U,V);
>> surf(X,Y,Z)
```

Dans les deux cas on obtient le résultat présenté figure A.20.

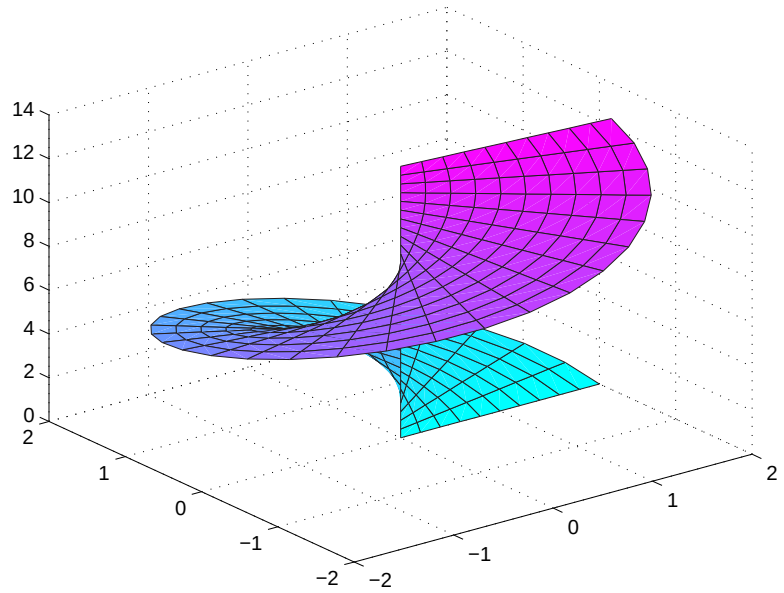


FIG. A.20 – Exemple de visualisation d'une surface paramétrée grâce à la commande `surf`.

A.7 Références

http://www.mathworks.com/access/helpdesk/help/helpdesk.html
http://maths.insa-lyon.fr/~balac/matlab/matlab.html

Cette page est laissée blanche intentionnellement

Annexe B : Le progiciel Matlab

B.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`

B.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.

B.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*.

B.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.

B.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.

B.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
```

B.2.4 Indexation - Extraction de sous-matrices

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

f =

     6     4     5
     9     7     8

>>
```

B.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`.

B.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.

B.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.

B.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`).

B.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)`.

B.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.

B.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.

B.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.

B.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`.

B.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.

B.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*.

B.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.

B.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).

B.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
```

B.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
```

B.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.

B.6 Fonctions Matlab prédéfinies

B.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

B.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

B.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

B.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.

B.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.

B.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]
```

B.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`.

B.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*.

B.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.

B.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.

B.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.

B.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`.

B.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`.

B.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.

B.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.

B.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.

B.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 `!`.

B.11.2 Gestion des répertoires et des fichiers

Sous Matlab, la commande ů`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 `!`.

B.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.

B.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.

B.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.

B.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 $Ax = b$.

B.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`.

B.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.

B.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.

B.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.

B.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.

B.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]);
```

B.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);
```

B.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.

B.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.

B.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`).

B.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.

B.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`.

B.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.

B.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`.

B.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.

B.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 C : Les fonctions usuelles de Matlab

C.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

C.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

C.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

C.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

C.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

C.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

C.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

C.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

C.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

C.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

C.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

C.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

C.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

C.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

C.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