UNIVERSITÉ DE GENÈVE

*Atelier d'outils informatiques pour la physique (InfoPhys)*

# Eléments de MATLAB

***Alfred A. Manuel***
Département de la Physique de la Matière Condensée
*alfred.manuel@physics.unige.ch*

15 October 2004

# Introduction

***Matlab*** est un logiciel de calcul numérique produit par MathWorks (voir le site web `http://www.mathworks.com/`). Il est disponible sur plusieurs plateformes.

***Matlab*** est un langage simple et très efficace, optimisé pour le traitement des matrices, d'où son nom. Pour le calcul numérique, ***Matlab*** est beaucoup plus concis que les "vieux" langages (C, Pascal, Fortran, Basic). Un exemple: plus besoin de programmer des boucles modifier pour un à un les éléments d'une matrice. On peut traiter la matrice comme une simple variable. ***Matlab*** contient également une interface graphique puissante, ainsi qu'une grande variété d'algorithmes scientifiques.

On peut enrichir ***Matlab*** en ajoutant des "boîtes à outils" (*toolbox*) qui sont des ensembles de fonctions supplémentaires, profilées pour des applications particulières (traitement de signaux, analyses statistiques, optimisation, etc.).

L'Université de Genève dispose d'une centaine de licences ***Matlab*** qui sont à la disposition de la communauté universitaire. Il suffit d'être connecté au réseau de l'université pour pouvoir l'utiliser. Plusieurs toolbox sont aussi disponibles (voir Appendice 1), leur nombre dépend de l'environnement, il est parfois plus restreint.

***Ces notes ne constituent pas une référence exhaustive sur MatLab, mais soulignent les notions principales de manière succincte et servent de guide pour le travail en laboratoire.*** Elles devraient inciter l'étudiant à chercher lui-même les compléments d'information qui lui sont nécessaires, soit avec les outils d'aide en ligne, soit dans les ouvrages suivants:

- **Introduction à Matlab**
  J.-T. Lapresté (Ellipses, 1999)
- **Mastering Matlab 6**
  D. Hanselman B. Littlefield (Prentice Hall, 2001)
- **Apprendre et maîtriser Matlab**
  M. Mokhtari A. Mesbah, (Springer, 1997)
- **Solving problems in scientific computing using Maple and Matlab**
  W. Gander, J. Hrebicek (Springer, 1995, second edition)
- **Numerical Methods Using Matlab**
  G. Lindfield J. Penny (Prentice Hall, 2nd edition : 2000)

***Matlab*** est relativement coûteux car il contient un nombre impressionnant de fonctions. Il existe une version étudiant à un prix abordable et un clone (***Octave***), disponible en freeware, dont la compatibilité avec ***Matlab*** est assez bonne: (http://www.octave.org/ ou dans la distribution SuSE de Linux).

Les exemples donnés dans le cours sont disponibles sur le serveur **servtp3-1** des Travaux Pratiques Avancés, dossier `public/infophys/matlab/cours/`. Les solutions d'exercices sont dans le dossier `public/infophys/matlab/exercices/`.

Nous nous bornons ici à décrire le language ***Matlab*** qui est indépendant de la plateforme utilisée. L'environnement de travail offre plusieurs fonctionalités assez conviviales qui sont utiles au développement. L'utilisateur les mettra à profit sans difficulté.

# 1. Aspects élémentaires

## 1.1 Aides

| | | |
|---|---|---|
| *help* | -> | donne de l'aide sur une fonction ou un toolkit (help help) |
| *helpdesk* | -> | documentation en hypertexte (requiert Netscape ou autre) |
| *helpwin* | -> | aide en ligne dans une fenêtre séparée |
| *lookfor* | -> | recherche d'un mot clé (lent) |
| *which* | -> | localise fonctions et fichiers |
| *what* | -> | liste des fichiers matlab dans le répertoire courant |
| *exist* | -> | check si une fonction ou une variable existe dans le workspace |
| *who, whos* | -> | liste des variables dans le workspace |

## 1.2 Variables scalaires, workspace, opérations élémentaires

```
>> var=2
var = 2

>> autre=3;

>> who % fournit la liste des fonctions définies dans le workspace
Your variables are:
autre var

>>whos % donne plus d'informations que who
```

Sous Windows, vous avez accès au "Workspace browser" dans la barre d'outils.

```
>> clear autre
>> who
Your variables are:
 var

>>clear % efface toutes les variables du workspace
```

Opérations élémentaires:

```
+  -  *  / or \  ^
>> 4/2
ans =
     2
>> 2\4
ans =
     2
```

## 1.3 Commentaires, ponctuation

```
>> s=2+3  % je fais une somme
s = 5

>> cout_moyen = cout ...   % commande sur deux lignes
                / nombre;
```

# 1.4 Variables spéciales

*pi inf i* or *j realmin realmax eps ans flops* (# d'opérations effectuées)

```
>> eps
ans =
   2.2204e-16
>> realmax
ans =
  1.7977e+308
>> realmin
ans =
  2.2251e-308
```

# 1.5 Nombres complexes

```
>> c1 = 1-2i
c1 =
   1.0000 - 2.0000i

c2 = 3*(2-sqrt(-1)*3)
c2 =
   6.0000 - 9.0000i
```

real(c1) imag(c1) abs(c1)

```
>> angle(c1)
ans =
   -1.1071
```

autres fonctions:
conj isreal

# 1.6 Fonctions mathématiques

*sin cos tan sinh cosh tanh ...*
*asin acos atan asinh acosh atanh ...*
*exp log log10 sqrt*
*fix floor ceil round mod rem sign*
*cart2sph cart2pol pol2cart sph2cart*
*factor isprime primes gcd (*pgcd*) lcm (*ppcm*)*

*nchoosek* (nombre de combinaisons différentes de N éléments pris k à k)

```
>> nchoosek(30,4)
ans =
       27405
```

perms (toutes les permutations possibles d'un vecteur V)

```
>> perms([1 2 3])
ans =
     3     2     1
     2     3     1
     3     1     2
     1     3     2
     2     1     3
     1     2     3
```

Attention au CPU! Vous pouvez stopper avec Ctrl-C

*besselj besselh .... beta ... erf ... gamma ... legendre*

*cross* (produit vectoriel)
*dot* (produit scalaire)

```
>> v1=[1 3 5]
v1 =
     1     3     5

>> v2=[2 4 6]
 v2 =
     2     4     6

>> cross(v1,v2)
ans =
    -2     4    -2

>> dot (v1, v2)
ans =
    44

>> whos
  Name      Size         Bytes  Class
   ans       1x1             8  double array
  c1        1x1            16  double array (complex)
  c2        1x1            16  double array (complex)
  v1        1x3            24  double array
  v2        1x3            24  double array
```

# 1.7 Affichage

FORMAT Set output format.
All computations in MATLAB are done in double precision.
FORMAT may be used to switch between different output
display formats as follows:
FORMAT SHORT (default) Scaled fixed point format with 5 digits.
FORMAT LONG Scaled fixed point format with 15 digits.
FORMAT SHORT G Best of fixed or floating point format with 5 digits.
FORMAT LONG G Best of fixed or floating point format with 15 digits.
Autres format : cf help format
Spacing:
FORMAT COMPACT Suppress extra line-feeds.
FORMAT LOOSE Puts the extra line-feeds back in.
examples:

```
>> pi
 ans =
    3.1416
>> format long g
>> pi
 ans =
          3.14159265358979
```

## 1.8 Entrées-sorties

Deux commandes utiles pour gérer le workspace, dont la taille dépend de votre espace de swap:

```
>> save % écrit toutes les variables du workspace dans le fichier matlab.mat
>> load % charge dans le workspace toutes les variables du fichier matlab.mat
```

Entrées-sorties sur des fichiers disques:
`fopen` (ouverture d'un fichier) `fclose` (fermeture d'un fichier)
`fscanf` (lecture formatée) `fprintf` (écriture formatée)

```
N = input( 'Nombre de boucles désirées >'); % entrée interactive

disp(N) % affiche la valeur de N
```

## 1.9 Terminer Matlab

```
>> quit % retourne le nombre d'opérations effectuées pendant la session
5340584 flops.
```

## 1.10 Personnaliser Matlab

Si, au démarrage, Matlab trouve le fichier `startup.m`, il l'exécute.
Exemple de `startup.m` :

```
disp(' Executing STARTUP.M')
addpath(' H:/Matlab/Cours');
addpath(' H:/Matlab/Exercices');
cd Matlab; % (set current directory to ./Matlab)
format compact
disp(' STARTUP.M completed')
```

# 2. Vecteurs

## 2.1 Création de vecteurs

Par défaut, le vecteur est une ligne à plusieurs colonnes

***a) vecteur ligne par énumération des composantes***:

```
>> v = [1 3.4 5 -6]
v =
    1.0000    3.4000    5.0000   -6.0000
```

***b) vecteur ligne par description***:

```
>> x = [0 : pi/10 : pi] % [valeur-initiale : incrément : valeur-finale]
x =
  Columns 1 through 7
         0    0.3142    0.6283    0.9425    1.2566    1.5708    1.8850
  Columns 8 through 11
    2.1991    2.5133    2.8274    3.1416
```

***c) vecteur colonne***:

```
>> xcol = x'
xcol =
         0
    0.2856
    0.5712
    0.8568
    1.1424
    1.4280
    1.7136
    1.9992
    2.2848
    2.5704
    2.8560
    3.1416
```

***d) génération de vecteurs métriques***

```
>> x = linspace(0, pi, 11) % génère le même x que ci-dessus (11 valeurs. répar-
ties de 0 à pi)
x =
  Columns 1 through 7
         0    0.3142    0.6283    0.9425    1.2566    1.5708    1.8850
  Columns 8 through 11
    2.1991    2.5133    2.8274    3.1416

>>%  linspace(0 , 1, 11) *pi    donne le même résultat

>> logspace(0, 2, 11) % crée un vecteur log à 11 composantes entre 10^0 et 10^2
ans =
  Columns 1 through 7
    1.0000    1.5849    2.5119    3.9811    6.3096   10.0000   15.8489
  Columns 8 through 11
   25.1189   39.8107   63.0957  100.0000
```

# 2.2 Adressages et indexages

```
>> x(3) % 3ème élément du vecteur x
ans =
    0.5712

>> x(2 : 4) % un bloc de composantes
ans =
    0.2856    0.5712    0.8568

>> x([8 3 9 1]) % une sélection de composantes (on les désigne avec un autre
vecteur!)
ans =
    1.9992    0.5712    2.2848         0
```

# 2.3 Combinaison de vecteurs

***a) Accolage de deux vecteurs***:

```
>> a = [1:3]
a =
     1     2     3
>> b=[10:10:30]
b =
    10    20    30

>> c = [a b]
c =
     1     2     3    10    20    30
```

On peut faire plus compliqué:

```
>> d=[a(2:-1:1) b] % on accole b avec une portion de a dans l'ordre renversé
d =
     2     1    10    20    30
```

Notez la différence entre () et [] :

**Tableau 1: Notations**

| | |
|---|---|
| **[ ]** | énumération d'éléments |
| **:** | descripteur d'éléments de vecteur/matrice |
| **( )** | ensemble d'arguments |
| **,** | séparateur d'arguments |
| **;** | séparateur des lignes dans les matrices<br>supression du résultat de l'évaluation d'une instruction |
| **'** | transposition de matrice |
| **.** | force l'opérateur à s'appliquer sur chaque élément du vecteur/matrice |
| **%** | délimitateur de commentaire |
| **...** | continuation de l'instruction sur la ligne suivante |

# 3. Matrices

## 3.1 Création de matrices

Un matrice est un ensemble de lignes comportant toutes le même nombre de colonnes
Matlab, depuis la version 5, supporte les matrices à n dimensions (n>2)

***a) Par énumération des éléments***

```
>> m1 = [ 1 2 3 ; 4 5 6 ; 7 8 9]  % on sépare les lignes par des point-virgules
m1 =
     1     2     3
     4     5     6
     7     8     9
```

On peut étendre aux matrices les autres manières de définir des vecteurs.
Par exemple:

```
>> m2 = [1:1:3 ; 11:1:13]
m2 =
     1     2     3
    11    12    13

>> m3 = [1:1:3 ; logspace(0, 1, 3)]
m3 =
    1.0000    2.0000    3.0000
    1.0000    3.1623   10.0000
```

## 3.2 Transposition

l'opérateur apostrophe utilisé pour créer un vecteur colonne est en fait l'opérateur transposition:

```
>> m2'
ans =
     1    11
     2    12
     3    13
```

## 3.3 Opérations scalaires-matrices

Une telle opération agit sur chaque élément de la matrice:

```
>> m2' * 10         % de même:  4*m2     m2-10     m2/4
ans =
    10   110
    20   120
    30   130
```

Une exception:

```
>> m2^2
??? Error using ==> ^
Matrix must be square.
```

Dans ce cas, Matlab veut calculer le produit matriciel m2 * m2

La solution est l'usage du point qui force l'opération sur chaque élément:

```
>> m2  .^ 2
ans =
```

```
     1     4     9
   121   144   169
```

# 3.4 Opérations entre matrices

***a) Multiplications***

```
>> m1 % rappelons la définition de m1
m1 =
     1     2     3
     4     5     6
     7     8     9

>> m2 % rappelons la définition de m2
m2 =
     1     2     3
    11    12    13

>> m1 * m2' % le produit matriciel n'est possible que lorsque les dimensions
sont cohérentes
ans =
    14    74
    32   182
    50   290

>> m1 * m2
??? Error using ==> *
Inner matrix dimensions must agree.
```

Multiplication élément par élément:

```
>> m2 .*m3 % (m2 et m3 ont les mêmes dimensions)
ans =
    1.0000    4.0000    9.0000
   11.0000   37.9473  130.0000
```

***b) Divisions***

```
>> m2/m3  % division matricielle à droite
ans =
    1.0000   -0.0000
    9.5406   -1.5960
>> m2\m3 % division matricielle à gauche (cf algèbre linéaire!)
ans =
   -0.5000   -0.8257   -0.4500
         0         0         0
    0.5000    0.9419    1.1500
```

Division élément par élément:

```
>> m2./m3 % chaque élément de m2 est divisé par l'élément équivalent de m3
ans =
    1.0000    1.0000    1.0000
   11.0000    3.7947    1.3000
>> m2.\m3 % chaque élément de m3 est divisé par l'élément équivalent m2
ans =
    1.0000    1.0000    1.0000
    0.0909    0.2635    0.7692
```

```
>> m3./m2 % chaque élément de m3 est divisé par l'élément équivalent m2
ans =
    1.0000    1.0000    1.0000
    0.0909    0.2635    0.7692
```

# 3.5 Matrices particulières

```
>> ones(3)
ans =
     1     1     1
     1     1     1
     1     1     1

>> zeros(2,5)
ans =
     0     0     0     0     0
     0     0     0     0     0

>> eye(4) % aussi: eye(2,4)
ans =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1

>> diag([1 : 4])
ans =
     1     0     0     0
     0     2     0     0
     0     0     3     0
     0     0     0     4

>> rand(1,7) % nombres aléatoires entre 0 et 1
ans =
    0.9355    0.9169    0.4103    0.8936    0.0579    0.3529    0.8132
```

# 3.6 Caractéristiques des matrices

```
>> size(m3) % dimensions
ans =
     2     3

>> length(m3) % equivalent à max(size(m3)) : dimension maximum
ans =
     3
>> rank (m3) %rang (nombre de colonnes ou lignes linéairement indépendantes)
ans =
     1
```

# 3.7 Manipulations de matrices et sous-matrices

**a)** ***Exercices de manipulations avec les notions vues jusqu'ici***

- *Définissez A une matrice 3x3*
- *Mettez à zéro l'élément (3,3)*

*- Changez la valeur de l'élément dans la 2ème ligne, 6ème colonne, que se passe-t-il?*
*- Mettez tous les éléments de la 4ème colonne à 4*
*- Créez B en prenant les lignes de A en sens inverse*
*- Créer C en accolant toutes les lignes de la première et troisième colonne de B à la droite de A*
*- Créer D sous-matrice de A faite des deux premières lignes et les deux dernières colonnes de A. Trouvez aussi une manière de faire qui ne dépende pas de la taille de A.*

***Note**: chacun de ces exercices se fait en une seule instruction, sans boucles ittératives.*

### b) Fonctions de manipulation des matrices:

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

>> flipud(A)    % flip up-down
ans =
     7     8     9
     4     5     6
     1     2     3

>> fliplr(A)    % flip left-right
ans =
     3     2     1
     6     5     4
     9     8     7

>> rot90(A,2)     %2 rotations de 90 degres (sens trigo)
ans =
     9     8     7
     6     5     4
     3     2     1

>> reshape(A,1,9)    % change la forme de la matrice
ans =
     1     4     7     2     5     8     3     6     9

>> diag(A)       % extrait la diagonale de A
ans =
     1
     5
     9

>> diag (ans)    % diag travaille dans les 2 sens !
ans =
     1     0     0
     0     5     0
     0     0     9

>> triu(A)    % extrait le triangle supérieur de A
ans =
     1     2     3
     0     5     6
     0     0     9
```

```
>> tril(A)     % triangle inférieur
ans =
     1     0     0
     4     5     0
     7     8     9
```

Il y a encore bien d'autres fonctions pour travailler les matrices, voir la liste dans l'annexe.

**c) *Exercice (avancé):***
*Sans utiliser de boucles d'ittération, ajouter aux éléments d'une matrice l'indice de leur colonne.*

# 3.8 Matrices clairsemées

Lorsque seulement quelques éléments d'une matrice sont non-nuls, on peut la définir comme une *sparse matrix*. Sa description contient seulement les éléments non nuls.

```
>> A_normal = [0 1 0 ; 1 0 0; 0 0 1] % matrice normale
A_normal =
   0   1   0
   1   0   0
   0   0   1
>> A_sparse = sparse(A_normal)  % matrice clairsemée
 A_sparse =
  (2,1)       1
  (1,2)       1
  (3,3)       1
```

*sparse* peut aussi être utilisé pour créer directement une matrice clairsemée:

```
>> S = sparse([2 1 3 4], [1 2 3 1], [4 5 6 7], 4, 3)
 S =
   (2,1)        4
   (4,1)        7
   (1,2)        5
   (3,3)        6
```

Le gain de place dans le workspace est d'autant plus significatif que la matrice est grande (on utilise `bucky`, une matrice clairsemée prédéfinie):

```
>> clear
>> B_sparse = bucky;%  matrice clairsemée
>> B_full = full(B_sparse); % matrice complète
>> whos
  Name             Size         Bytes  Class
  B_sparse        60x60         28800  double array
  B_full          60x60          2404  sparse array
```

*full* a convertit la matrice clairsemée en matrice complète.

# 4. Programmer en Matlab

## 4.1 Opérateurs logiques et de relation

< plus petit
> plus grand
<= plus petit ou égal
>= plus grand ou égal
== égal
~= pas égal
& et
| ou
~ not
`xor(x,y)` ou exclusif
`any(x)` retourne 1 si un des éléments de x est non nul
`all(x)` retourne 1 si tous les éléments de x sont nuls
`isequal(A,B), ischar etc...`

## 4.2 Contrôler l'exécution

***a)For***

```
for n = 1:5
  for m = 5:-1:1
    A(n,m) = n^2 + m^2;
  end
disp(n)
end
     1
     2
     3
     4
     5
>> A
A =
     2     5    10    17    26
     5     8    13    20    29
    10    13    18    25    34
    17    20    25    32    41
    26    29    34    41    50
```

***b) While***

```
while expression
   (commands)
end
```

***c) If-then-else***

```
if expression1
   (commandes à exécuter si expression1 est“vrai”)
elseif expression2
   (commandes à exécuter si expression2 est“vrai”)
else
   (commandes à exécuter si aucune expression est “vrai”)
end
```

## 4.3 M-Files ou scripts

Un script (ou M-file) est un fichier (message.m par exemple) contenant des instructions Matlab. Voici un exemple de script:

```
% message.m affiche un message
% ce script affiche le message que s'il fait beau
beau_temps=1;
if beau_temps~=0
  disp('Hello, il fait beau')
  end
return % (pas nécessaire à la fin d'un M-file)
```

Matlab vous offre un éditeur pour écrire et mettre au point vos M-files:

```
>> edit % lance l'éditeur de MatLab. Voir aussi la barre d'outils
```

Tout autre éditeur de texte convient aussi.

Les M-files sont exécutés séquentiellement dans le "workspace", c'est à dire qu'ils peuvent accéder aux variables qui s'y trouvent déjà, les modifier, en créer d'autres etc.

On exécute un M-file en utilisant le nom du script comme commande:

```
>> message
Hello, il fait beau
```

*Exercice:*
*Dans un script "pluspetitnombre.m", utiliser la structure "while" pour évaluer le plus petit nombre tel que, ajouté à 1, on obtient un nombre supérieur à 1.*

## 4.4 Fonctions

On peut écrire des fonctions MatLab que l'on peut ensuite appeler depuis un script.
Voici une fonction" temps" définie dans le fichier temps.m:

```
function y=temps(x)
% TEMPS(X) affiche un message suivant le temps qu'il fait
% et retourne le paramètre d'entrée X changé de signe
if length(x)>1 error('X doit être un scalaire'); end
if x~=0
  disp('Hello, il fait beau')
else
  disp('Espérons que demain sera meilleur!')
end
y=-x;
return
```

Utilisation de cette fonction:

```
>> clear
>> help temps
  TEMPS(X) affiche un message suivant le temps qu'il fait
  et retourne le paramètre d'entrée X changé de signe
>> temps(1)
Hello, il fait beau
    -1
```

Remarquez que les variables internes d'une fonction sont locales et n'entrent pas dans le workspace. Vous pouvez, dans une fonction, déclarer une ou plusieurs variables globales afin de pou-

voir les utiliser depuis l'extérieur de la fonction. Par exemple:

```
global x y
```

Matlab offre plusieurs moyens de vérifier les arguments d'entrées et de sorties d'une fonction:

| | |
|---|---|
| `nargin` | retourne le nombre d'arguments d'entrée |
| `nargout` | retourne le nombre d'arguments de sortie |
| `nargchk` | vérifie le nombre d'arguments d'entrée |
| `error` | Affiche un message d'erreur |
| `inputname` | retourne le nom d'un argument d'entrée |

*Exercice:*
*Ecrivez une fonction "index_of_max" qui retourne l'index de l'élément le plus grand d'un vecteur A donné comme argument d'entrée.*

# 4.5 Gestion du système de fichiers

Matlab utilise la syntaxe Linux/Unix (DOS est assez semblable à ce niveau) pour la gestion des dossiers. Ainsi on peut connaître le dossier courant et se déplacer dans l'arbre des dossiers à l'aide des commandes suivantes :

| | |
|---|---|
| `pwd` | permet de connaître le dossier courant (*Print Working Directory*) |
| `cd ..` | remonte au dossier supérieur |
| `cd sub` | sélectionne le dossier inférieur nommé *sub* |
| `ls` | donne la liste des fichiers dans le dossier courant (*LiSt*) |

Examples:

```
>> pwd
E:\matlab
>> cd E:\infophys\matlab\cours
>> ls
ReO3_GX.m       diffsplines.m   rex.m           test_filter.m
ballistic.m     message.m       temps.m
ballistic_ODE.m myfun1.m        test.dat

>> cd ..
>> cd exercices
>> pwd
E:\infophys\matlab\exercices
```

# 5. Graphisme

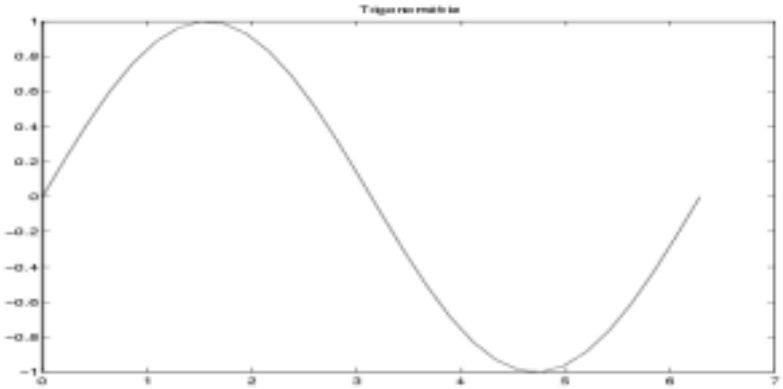
Nous donnons ici les indications minimum. Utilisez help et les autres commandes d'aide pour affiner vos connaissances et vos graphiques.

## 5.1 Graphiques à 2D

```
>> help graph2d % intro. au graphisme 2D et tableau des fonctions disponibles
```

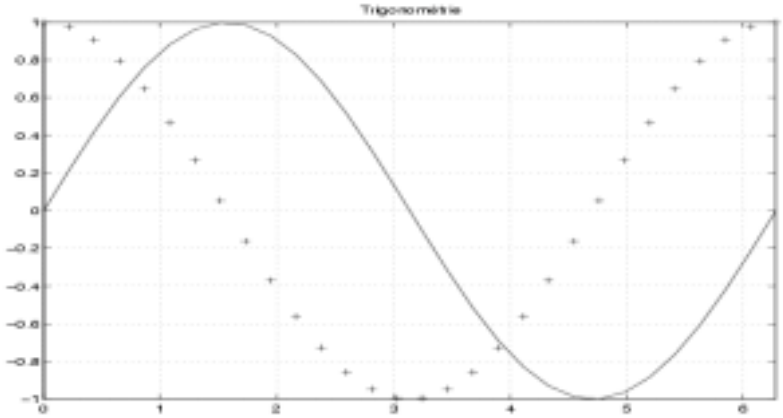
a) courbes: ***plot***

```
>> x=linspace(0,2*pi,30);
>> y=sin(x);
% un plot rudimentaire:
>> plot(x,y)
>> title('Trigonométrie') % MatLab1_fig1
```



***MatLab1_fig1***

```
% quelques améliorations:
>> grid on
>> axis([ 0 2*pi -1 1])
% add a second curve
>> hold on
>> z=cos(x)
>> plot(x,z,'c+') % MatLab1_fig2
>> clf % efface la figure
```



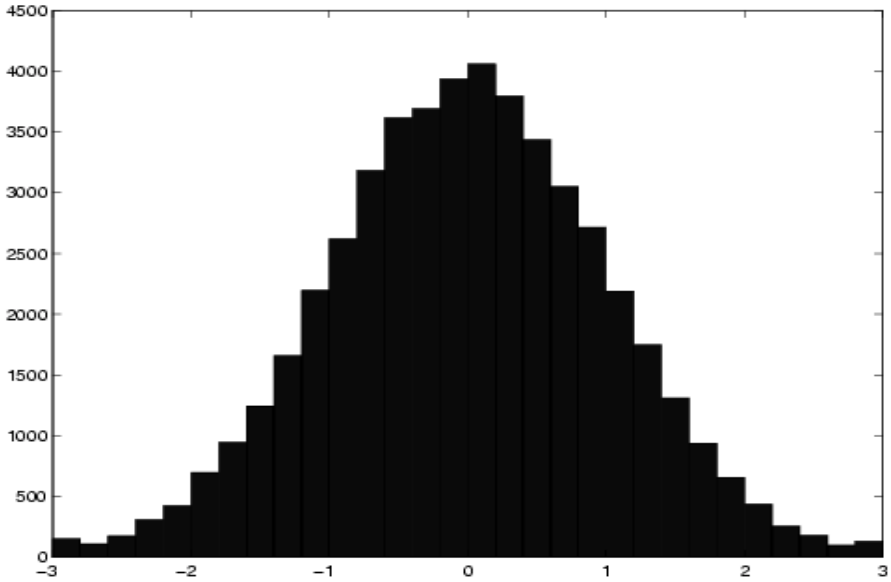
***MatLab1_fig2***

`semilogx`, `semilogy` et `loglog` sont semblables à plot mais permettent de faire des plots log.

*Exercice :*
*Ecrivez un script pour montrer l'évolution graphique de vos dépenses pour la santé en fonction du montant de vos frais médicaux annuels. Evaluez 2 situations: 1) Prime mensuelle: 300.- ; la caisse rembourse 90% des frais avec une franchise de 150.- 2) rabais de 37% sur vos primes si la franchise est de 1'500.- Quand êtes-vous gagnant?*

b) histogrammes: ***hist***

```
>> x=-2.9 : 0.2 : 2.9; %défini l'intervalle et la largeurs des canaux de l'his-
togramme
>> y=randn(50000,1); % génère des nombres aléatoires répartis selon une distr.
normale
>> hist(y,x) % dessine l'histogramme MatLab1_fig3
```




***MatLab1_fig3***

c) graphes pour matrices clairsemées: ***spy, gplot***

```
>> spy(bucky) % donne un schéma des composantes non-nulles de bucky
```

Les matrices clairsemées sont parfois utilisées pour décrire les connections dans les graphes. gplot permet de dessiner ces connections:

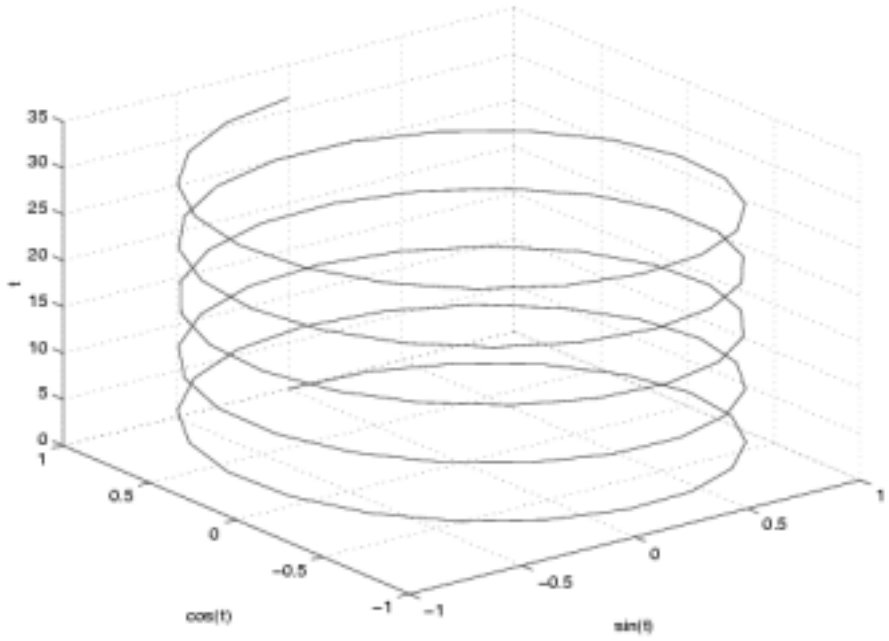
```
>> [B, v] = bucky;
>> gplot(B,v); axis equal %
```

# 5.2 Graphiques à 3D

```
>> help graph3d % introduction au graphisme 3D et tableau des fonctions dis-
ponibles
```

a) ligne dans l'espace: ***plot3***

```
>> t = linspace(0, 10*pi);
>> plot3(sin(t), cos(t), t)
>> xlabel('sin(t)'), ylabel('cos(t)'), zlabel('t')
>> grid on % MatLab1_fig4
```




***MatLab1_fig4***

b) Construction d'un maillage dans le plan (x,y) : ***meshgrid***
Pour la représention d'une surface f(x, y), on a besoin de connaître les triplets de coordonnées ($x_i$, $y_j$, $Z_{i,j}$), avec $Z_{i,j}=f(x_i,y_j)$, pour un certain nombre de points ($x_i$, $y_j$) où i=1, ..., N, j=1, ..., M. On voit que Z a la structure d'une matrice $\{Z_{i,j}\}$. Par soucis d'harmonisation, *Matlab* utilise également une représentation matricielle pour les coordonnées x et y dans le plan, bien qu'à priori, des vecteurs soient suffisants. Il faut accepter cette stratégie... Matlab fournit la fonction ***meshgrid*** pour générer les matrices $\{X_{i,j}\}$ et $\{Y_{i,j}\}$ à partir des vecteurs $\{x_i\}$ et $\{y_i\}$. L'exemple ci-dessous montre comment s'utilise ***meshgrid*** et quelle est la forme des matrices X et Y.

```
% TESTMESHGRID -- Pour la notion de "mesh" (maillage) dans Matlab
%
clear
% on part de deux vecteurs qui définissent le maillage sur chaque axe du plan
x = logspace(0, 2, 3)
y = linspace(0, 100, 5)

% meshgrid fournit les matrices X et Y à partir des vecteurs x et y :
[X Y] = meshgrid(x, y)

% La maille (4, 2) est donnee par :
P_4_2 = [X(4,2) ; Y(4,2)]
% mais aussi, indifférement, par une des expressions suivantes
%P_4_2 = [X(1,2) ; Y(4,1)]
%P_4_2 = [X(3,2) ; Y(4,3)]
%P_4_2 = [X(2,2) ; Y(4,2)]
```

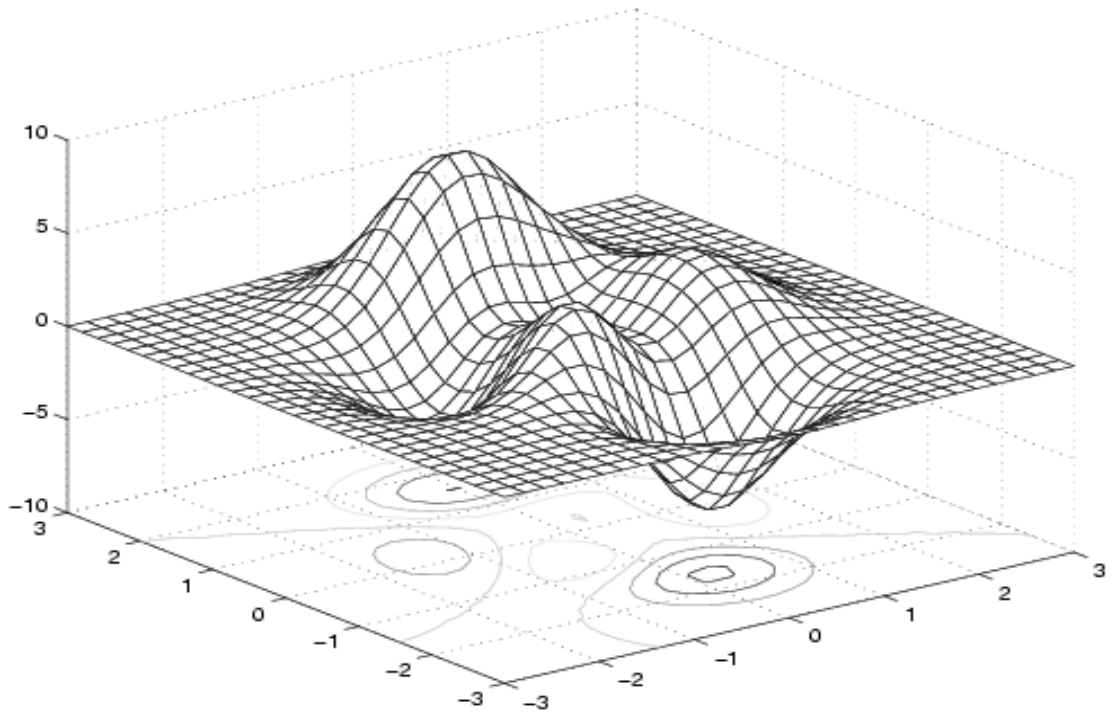
Résultat de testmeshgrid.m :

```
>> x =
     1    10   100
```

```
y =
     0    25    50    75   100
X =
     1    10   100
     1    10   100
     1    10   100
     1    10   100
     1    10   100
Y =
     0     0     0
    25    25    25
    50    50    50
    75    75    75
   100   100   100
P_4_2 =
    10
    75
```

c) grillage en perspective: ***mesh***

```
>> clf
>> x = linspace(-3, 3, 30);
>> y = linspace(-3, 3, 30);
>> [X Y] = meshgrid(x,y);
>> Z = peaks(X, Y) % peaks est une fonction à 2-D prédéfinie dans matlab
>> mesh(X, Y, Z) % grillage 3D
>> meshc(X, Y, Z) % grillage 3D avec les contours sur le plan de base
>>                % (voir MatLab1_fig5 )
```

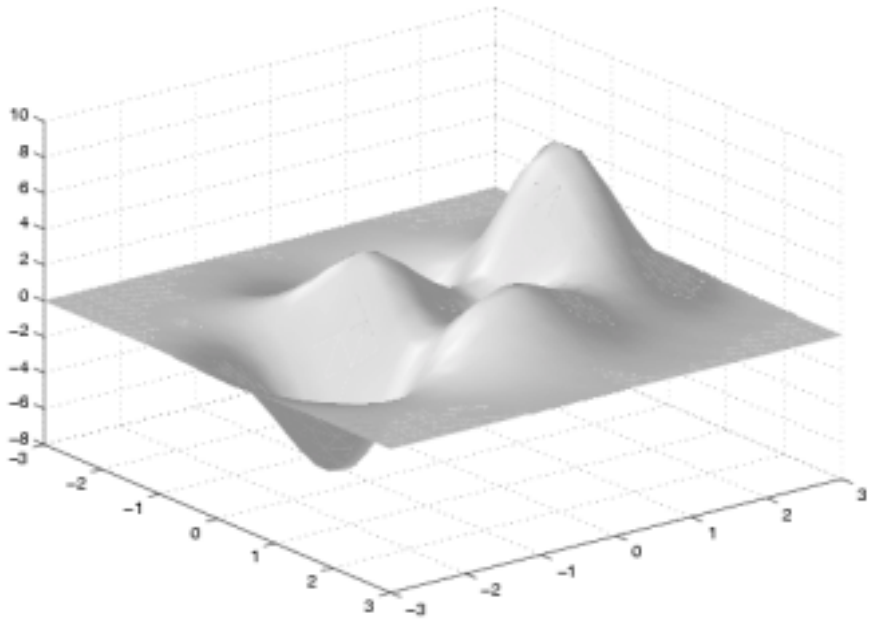



***MatLab1_fig5***

d) Surface avec illumination: ***surfl***

```
>> clf
>> [X,Y,Z] = peaks(30);
>> surfl(X,Y,Z) % graphique avec illumination
>> shading interp % meilleure interpolation
>> colormap pink %choix d'une palette de couleurs prédéfinie
```

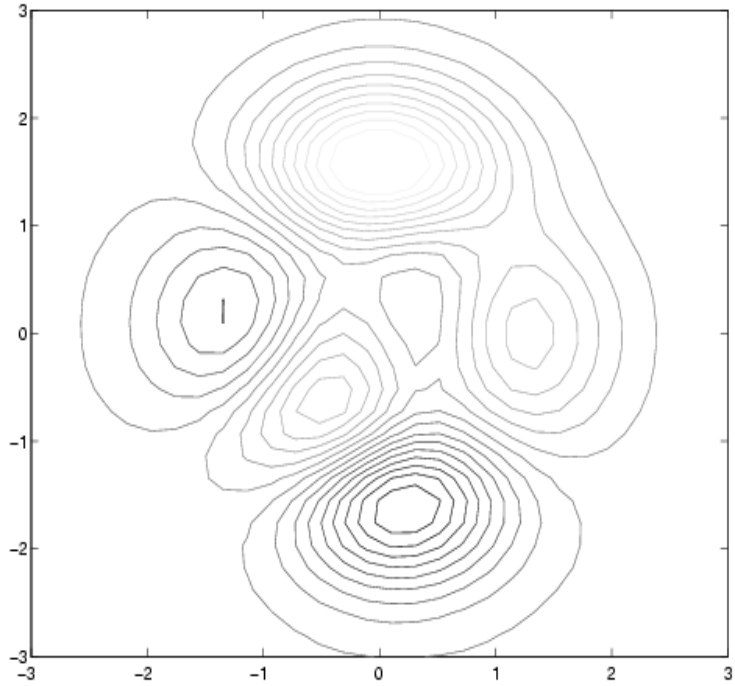
```
>> view(-37.5+90, 30) % changement point de vue: view(azimut, elevation)
MatLab1_fig6
```




***MatLab1_fig6***

e) courbes de niveau: ***contour***

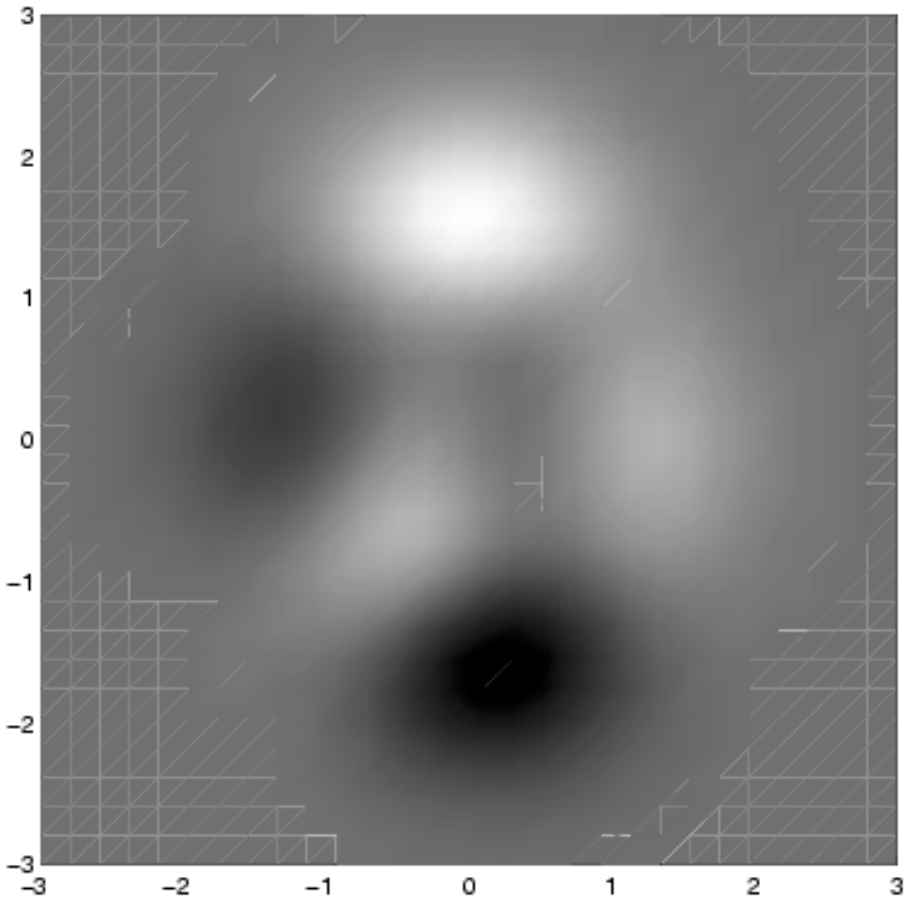
```
contour(X,Y,Z,20) % 20 lignes de niveau MatLab1_fig7
```




***MatLab1_fig7***

```
% pour transformer ce graphe en échelle de couleurs
>> colormap('gray')
>> pcolor(X,Y,Z) % plot pseudo-couleur de Z sur la grille X,Y
>> shading interp
>> axis('square') %  MatLab1_fig8
```

```
>> colormap('default')
```




***MatLab1_fig8***

# 5.3 Animations

moviein: initialisation d'un buffer pour les images
getframe: capture une figure dans une image de l'animation
movie: affiche l'animation

*Exemple:*

```
% RotatePeaks.m - Exemple d'animation
%
% On crée plusieurs frames dans lesquels on met un même graphique 3D
% vu sous des angles différents de manière à créer une impression de
% rotation
%
clear
%
% préparation du graphique qui va servir de base à l'animation
[X,Y,Z] = peaks(30);
surfl(X,Y,Z)
axis([-3 3  -3 3  -10 10])
axis off
shading interp
```

```
colormap (hot)
%
% construction de l'animation
m=moviein(15);  %initialise l'animation (15 images)
for i=1:15
     view(-37.5+24*(i-1),30) % change le point de vue
     m(:,i)=getframe; % capture une image; Notez que m(i)= ... va ausi
     end
%
% présentation du résultat
movie(m,5) % affiche l'animation 5+1 fois
```

***Importer/exporter des animations***

Il est possible d'exporter un movie Matlab dans un fichier MPEG à l'aide de `mpgwrite`, ou `movie2avi`, des commandes que l'on trouve sur le site web de Matworks. De même, il est possible de charger un fichier MPEG dans un movie Matlab avec `mpgread`.

***Exercice****:*

*Construisez une animation du jeu de la vie.*

*On part avec un espace 2D composé de (m*n) cellules qui sont aléatoirement vivantes ou mortes. A chaque génération (itération), cette population évolue en suivant les deux règles suivantes:*

*1) une cellule morte devient vivante si elle est entourée exactement par 3 cellules vivantes parmi les 8 cellules qui l'entourent.*

*2) une cellule vivante survit seulement si elle est entourée par deux ou trois cellules vivantes.*

*Conseil: utiliser la fonction graphique "image" pour visualiser l'espace dans lequel évolue la population de cellules.*

***Extension de l'exercice précédant*** *:*

*Poissons & Requins (voir dossier infophys/matlab/cours/SharksAndFishes)*

# 6. Quelques applications de Matlab

## 6.1 Fits et interpolations

***a) Fits de polynômes***: (`interpolation.m`)

```
>> x = [0 : 0.1 : 1];
>> y = [ -.447 1.978 3.28 6.16 7.08 7.34 7.66 9.56 9.48 9.3 11.2];
>> p = polyfit(x,y,2) % fit un polynôme du deuxième degré
p =
   -9.8108   20.1293   -0.0317
```

La solution est $y = -9.8108x^2 + 20.1293x - 0.0317$.

Regardons le résultat:

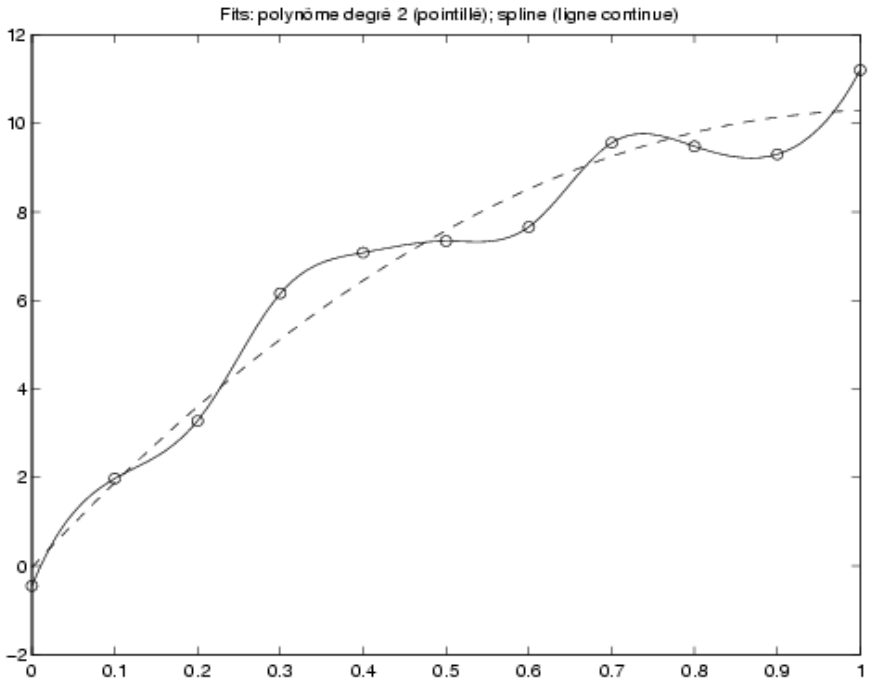
```
>> xi = linspace(0, 1, 100);
>> yi = polyval(p, xi);
>> plot(x, y, 'o', xi, yi, '--') % MatLab1_fig9  (plus bas)
```

***b) fit de fonctions splines***:

Si, pour mieux s'approcher des points à fitter, on utilise des polynômes de degrés élevé, on obtient des oscillations (***Exercice***: *essayez de tels fits en reprenant les vecteurs x et y donnés ci-dessus*). Les splines sont une manière de résoudre ce problème et d'obtenir de meilleurs fits qu'avec les polynômes.

Rappel: une spline cubique (la plus fréquente) est composée de segments de polynômes du troisième degré dont les dérivées premières et deuxièmes sont continues aux points de jonction.

```
>> yi = spline(x, y, xi); % on fitte une spline sur les points (x,y)
>> hold on %on fixe le graphique afin d'y rajouter une courbe
>> plot( xi, yi, '-') % on rajoute la spline => MatLab1_fig9
>> title('Fits: polynôme degré 2 (pointillé); spline (ligne continue)')
```




***MatLab1_fig9***

# 6.2 Intégrations

Note : “humps” est une fonction numérique prédéfinie dans MatLab.

***a) Intégrer un vecteur:***

```
>> x = linspace(-1, 2, 1000);
>> y = humps(x);
>> area=trapz(x,y)
area =
   26.3450
```

***b) intégrer une fonction:***

```
>> quad('humps',-1,2) % intégre à 6 chiffres significatifs
ans =
   26.3450
>> quad8('humps',-1,2) % intégre à 8 chiffres significatifs
ans =
   26.3450
```

Vous pouvez intégrér vos propres fonctions:
Supposons que vous avez écrit le script suivant dans le fichier `myfun1.m`

```
function y = myfun1(x)
%MYFUN1 est une fonction simple définie dans le fichier myfun1.m
y = exp(-x.^2) .* cos(x); %La fonction peut traiter des vecteurs
```

```
>> area = quad('myfun1', 0, 6*pi)
Warning: Recursion level limit reached in quad. Singularity likely.
> In /unige/matlab_5.2/toolbox/matlab/funfun/quad.m (quadstp) at line 102
Warning: Recursion level limit reached 806 times.
> In /unige/matlab_5.2/toolbox/matlab/funfun/quad.m at line 80
area =
    0.6902
```

*Exercice: Intégration double*
*a) Ecrivez une fonction* `myfun2` *de votre choix à deux variables.*
*b) Dessinez-là avec une fonction graphique 3D. Indication: utiliser* `mesgrid` *et* `mesh`
*c) Evaluez son intégrale double avec* `dblquad`.
*d) Mesurer le temps CPU nécessaire pour faire l'intégration : utilisez* `cputime`

# 6.3 Différentiations

Contrairement à l'intégration, la différentiation numérique est difficile, puisqu'elle est très sensible à la forme locale de la fonction. La meilleure manière de faire de la différentiation est de fitter un polynôme ou une fonction spline. Dans les deux cas, on connaît l'expression analytique de la dérivée.

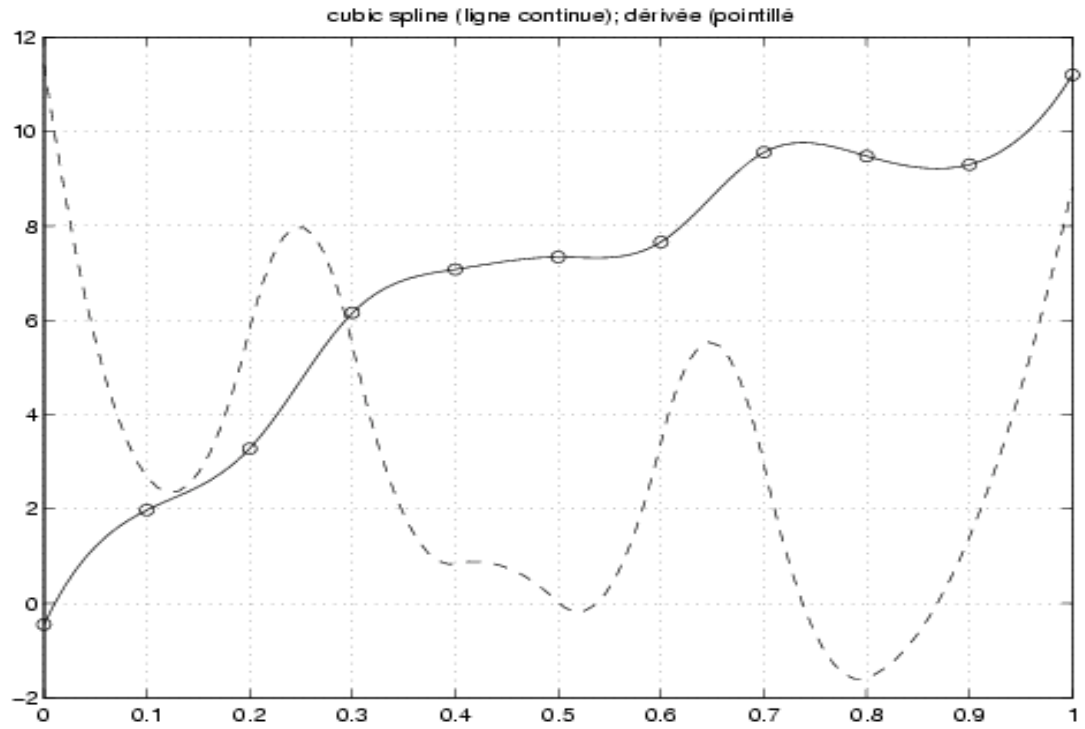
***a) Dérivée approximative***

La fonction `DY = diff(Y)` retourne en `DY` la dérivée de `Y` calculée simplement en prenant la différence entre les éléments adjacants de `Y`. “`diff`” retourne un vecteur moins long que Y, ce qui peut être gênant. Dans ce cas, vous pouvez utiliser “`gradient`”.

***b) différentiation par cubic-splines***

Cette méthode n'est pas tout à fait directe avec MatLab. Voici un script qui la décrit:

```
% DIFFSPLINES.M    Différentiation avec des cubic-splines
% Fitte un ensemble de valeurs avec des splines et calcule la dérivée
clear
clf

% (x;y) sont les points à fitter
x = [0 : 0.1 : 1];
y = [ -.447 1.978 3.28 6.16 7.08 7.34 7.66 9.56 9.48 9.3 11.2];

%Il faut commencer par fitter des cubic splines (voir plus haut)
xi = linspace(0, 1, 100); % mesh fin entre 0 et 1
yi = spline(x, y, xi); % fit des cubic splines.
plot(x, y, 'o', xi, yi, '-');
hold on

% différentiation:
pp = spline(x,y); % obtient les polynômes partiels de la spline
if pp.pieces(1)~=length(y)-1
  error('pas de pp-form pour cette spline cubique')
  end
[br, coefs, l, k] = unmkpp(pp); % obtient les coefficients des polynômes
sf = k-1:-1:1; % scaling factors
dco = sf(ones(l,1),:) .* coefs(:,1:k-1); % évalue les coeffs de dériva-
tion
ppd = mkpp(br, dco); %construit la pp-form pour la différentiation
yid = ppval(ppd, xi); % calcul de la dérivée
plot( xi, yid/4, '--'); MatLab1_fig10
grid on
title('cubic spline (ligne continue); dérivée (pointillé)')
```




***MatLab1_fig10***

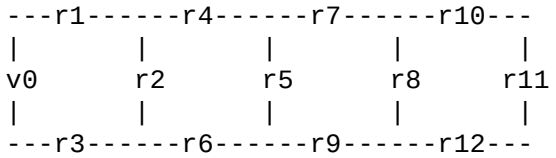
*Exercice 6.3.1 :*
*Ecrivez une* ***fonction*** `derpline(x, y, xi)` *qui calcule la dérivée par la méthode des splines, d'une courbe définie par un ensemble de paires (x,y) et retourne la dérivée aux différentes valeurs spécifiées dans un vecteur xi. Cette fonction doit pouvoir donner un message d'erreur si les paramètres d'entrées ne sont pas ceux qu'elle attend. (Indication: s'inspirer du script diff-splines.m ci-dessus)*

***c) différentiation par polynômes***
*Exercice 6.3.2 : calculer la dérivée du polynôme fitté plus haut sur les points (x;y):*
*x = [0 : 0.1 : 1];*
*y = [ -.447 1.978 3.28 6.16 7.08 7.34 7.66 9.56 9.48 9.3 11.2];*
*Indication: utiliser polyder*

***d) Gradient, Laplacien***
```
>> gradient (F) % dérivée de F si F est un vecteur
>> gradient(H) % gradient de H, si H est une matrice
>> del2(U) % approximation du Laplacien de U
```

# 6.4 Algèbre linéaire

***a) Systèmes d'équations linéaires***
Le système d'équations linéaires
$a_{11}*x_1 + a_{12}*x_2 + ... + a_{1n}*x_n = b_1$
...
$a_{n1}*x_1 + a_{n2}*x_2 + ... + a_{nn}*x_n = b_n$
s'écrit sous forme matricielle: **A** * **x** = **b**
Il a une solution unique ssi rank(A) = rank([A b] )= n
La solution est alors obtenue soit en utilisant la fonction "inv" d'inversion de matrice:
```
>>x = inv(A) * b
```
soit en utilisant "\" la division matricielle par la gauche:
```
>> x = A\b
```
Cette seconde méthode donne des résultats plus précis, lorsqu'elle peut être appliquée.
***Exercice*** *6.4.1:*
*Appliquer les lois de Kirchhoff et d'Ohm pour décrire un circuit électrique composé de résistances formant une échelle. Déterminer le courant dans chaque boucle lorsqu'une extrémité de l'échelle est soumise à une différence de potentiel.*




***Note:***
Selon le système d'équations linéaires, on doit faire attention à la méthode utilisée. Si `inv` ne marche pas, voici un petit guide pour tenter de trouver une solution :
Si A est une matrice triangulaire régulière, essayer \
sinon si A est définie positive ou carrée symétrique ou hermitienne, essayer la factorisation de Cholesky : `chol`
sinon si A est non carrée, essayer la décomposition QR (Q= othogonal unitary matrix, R=upper triangular matrix, ) : `qr`

sinon pour A clairsemée, il faut essayer la famille d'outils que matlab met à disposition dans ce cas (voir la discussion dans *Numerical Methods using Matlab*, 2nd edition, G. Lindfield and J.Penny, Prentice Hall, 2000)

***b) Systèmes à valeurs propres et vecteurs propres***

L'équation de Schödinger indépendante du temps s'écrit $H\Psi=E\Psi$

Dans un solide périodique, la fonction d'onde $\Psi$ se décrit comme un combinaison linéaire de fonctions orthonormées $\phi$:

$$\Psi(\vec{r}) = \sum_{i=1}^{N} b_i \Phi_i(\vec{r})$$

L'équation de Schrödinger se transforme en

$$\sum_i (H_{i,j} - E\delta_{i,j}) b_i = 0$$

qui n'a de solutions que si le déterminant de $(H_{i,j}-E\delta_{i,j})=0$ (voir Mécanique Quantique).

Pour un $H_{i,j}$ donné, on veut trouver les valeurs propres E de l'énergie et les vecteurs propres $b_i$ qui donnent les fonctions d'onde.

La fonctions `eig` permet d'obtenir vecteurs et valeurs propres.

***Illustration:***

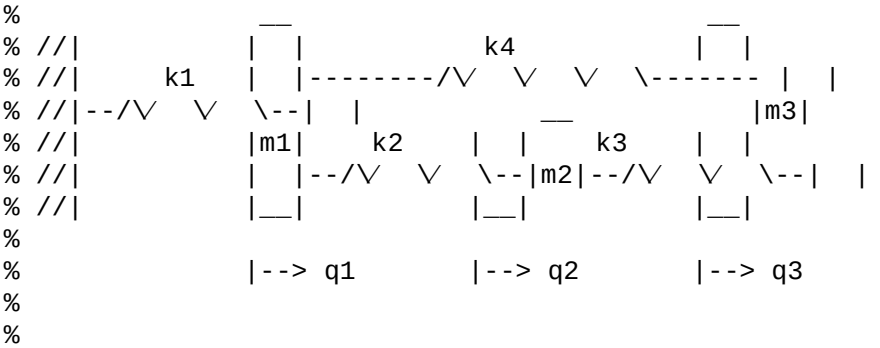
Voici comment se décrivent les états électroniques dans un cristal de $ReO_3$ (approximation des liaisons fortes, voir cours de Physique du Solide):

```
% ReO3GX.m  Band structure of ReO3 by the tight-binding method
%           Plot only Gamma-X; to plot X-M is left as exercise.
%           (Adapted from a script written by Boris Ishii, DPT)
%
clf; clear
d1=0.0592/2; Valeurs des paramètres de l'Hamiltonien (T. Chiba)
d4=0.4218/2;
p1=0.1324;
p2=-0.221;
a1=-0.2761/2;
b1=-0.2294/2;
a=1;
N=40; %Nombre de points où on désire calculer les énergies propres
GX=pi/a; % GX est la limite de la zone de Brillouin (ZB) dans la direction kx
% Calcul pour (kx,ky,kz) entre (0,0,0) et (1,0,0)
clear kx; kx=[0:GX/N:GX]; % varying kx
nk = length(kx); % nombre de points kx
ky=0.; kz=0.; % les composantes ky et kz du vecteur d'onde sont gardées 0.
clear E; % (redondant)
for j=1 : nk % boucle sur les points k le long du chemin défini dans la ZB
   sx=-2*i*p1*sin(0.5*kx(j)*a);
   sy=-2*i*p1*sin(0.5*ky*a);
   sz=-2*i*p1*sin(0.5*kz*a);
   Sx=i*p2*sin(0.5*kx(j)*a);
   Sy=i*p2*sin(0.5*ky*a);
   Sz=-2*i*p2*sin(0.5*kz*a);
   % A ,le triangle supérieur de l'Hamiltonien est défini par:
   A=[d1     sx  sy  0   0   0   0   0   0   0   0           0          0;
      0   b1 0   0   0   0   0   0   0   0   0   0           0          0;
      0   0  b1  0   0   0   0   0   0   0   0   0           0          0;
```

```
      0   0  0   d1  sy  sz  0   0   0   0   0   0           0          0;
      0   0  0   0   b1  0   0   0   0   0   0   0           0          0;
      0   0  0   0   0   b1  0   0   0   0   0   0           0          0;
      0   0  0   0   0   0   d1  sz  sx  0   0   0           0          0;
      0   0  0   0   0   0   0   b1  0   0   0   0           0          0;
      0   0  0   0   0   0   0   0   b1  0   0   0           0          0;
      0   0  0   0   0   0   0   0   0   d4  0   Sx          Sy         Sz;
      0   0  0   0   0   0   0   0   0   0   d4  -sqrt(3)*Sx sqrt(3)*Sy 0;
      0   0  0   0   0   0   0   0   0   0   0   a1          0          0;
      0   0  0   0   0   0   0   0   0   0   0   0           a1         0;
      0   0  0   0   0   0   0   0   0   0   0   0           0          a1];
  E(:,j)=sort(eig(A+A'));%A+A'= Hamiltonien,eig=valeurs propres,
                         %sort=classement
end
[nb nk]=size(E); %nb: nombre de valeurs propres
figure(1)
hold; grid;
for n=1:nb
  plot(kx,E(n,:)) % graphe des bandes (valeurs propres de l'Hamiltonien H)
  end
axis([min(kx) max(kx) -0.6 0.8])
xlabel('kx')
ylabel('Energy')
title('Structure de bandes du ReO3 : G-X');
```

Cet exemple illustre admirablement le language synthétique et les outils qu'offre MatLab!

***Exercice 6.4.2:***
*Déterminez les fréquences propres (naturelles) du système de ressorts couplés ci-dessous.*




# 6.5 Equations différentielles ordinaires (ODE)

Dans ses versions récentes, MatLab contient un ensemble de fonctions pour résoudre quasiment tout problème décrit par des ODE du type suivant, avec des conditions initiales défines.

$$\dot{\vec{y}} = f(t, \vec{y}) \qquad \vec{y}(t_0) = \vec{y}_0$$

Pour une introduction et des exemples:
`>> helpdesk %et chercher "ode45" dans la fenêtre "Go to MATLAB function"`
La routine "`ode45`" est la routine de base. Elle implémente une variante de la méthode de Runge-Kutta (voir par exemple http://www.geog.ubc.ca/numeric/labs/lab4/lab4/node5.html) qui, elle-même est un raffinement de la méthode très simple d'Euler qui utilise la dérivée de f à t pour évaluer f à t+dt. D'autres algorithmes sont à disposition. Par exemple :
`ode23` : Rosenbrock formula of order 2

`ode113` : Adams-Bashforth-Moulton.
MatLab offre aussi une démo (pas très explicite) sur la solution des ODE:
`>> odedemo`

La stratégie pour résoudre les ODE est d'écrire une fonction (par exemple la fonction "myode", contenue dans le M-file myode.m) qui décrive le membre de droite de l' équation à résoudre (voir `help odefile`).
On peut alors obtenir la solution $\vec{y}(t)$, dans un intervalle t=[$t_{initial}$ $t_{final}$] et pour des conditions initiales définies dans y0 en utilisant `ode45`.

***Exemple simplicime: Refroidissement d'un corps***
Le corps à la température y0=100 est plongé dans un milieu de température s=10. On suppose le taux de refroidissement K constant = -0.1.

L'équation différentielle est : $dy/dt = K(y-s)$. Le script `myode.m` est le suivant:

```
% myode.m :
   function yprime = myode(t, y)
   yprime = -0.1 *(y-10);
```

et l'évolution dans le temps de la température y est obtenue en résolvant l'équation différentielle à l'aide de `ode45` : (fichier refroidissement.m)

```
>> tinitial=0 ;
>> tfinal=60 ;
>> y0=100 ;
>> [t y] = ode45('myode', [tinitial tfinal], y0)
>> plot(t,y); xlabel('time'); ylabel('temperature');
```

`ode45` (et les autres) permet de résoudre des systèmes de plusieurs ODE couplés. On va utiliser cette propriété dans l'exemple ci-dessous.

***Exemple: tir ballistique dans le vide***
Une transformation de variable permet de résoudre les équations du 2ème ordre.

composante horizontale du tir ; composante verticale du tir

Les équations du mouvement sont:

$\ddot{x} = 0$ ; $\ddot{z} = -g$

Pour ramener ces équations du 2ème ordre au premier ordre on effectue les transformations de variables suivantes:

$X(1) = x$ ; $X(2) = z$
$X(3) = \dot{x}$ ; $X(4) = \dot{z}$

En dérivant, on obtient alors les 4 ODE suivantes:

$\dot{X}(1) = X(3)$ ; $\dot{X}(2) = X(4)$
$\dot{X}(3) = 0$ ; $\dot{X}(4) = -g$

avec les conditions initiales suivantes:

$X_0(1) = 0$ ; $X_0(2) = h$
$X_0(3) = v_0 \cos\theta$ ; $X_0(4) = v_0 \sin\theta$

La fonction décrivant les ODE s'écrit alors:

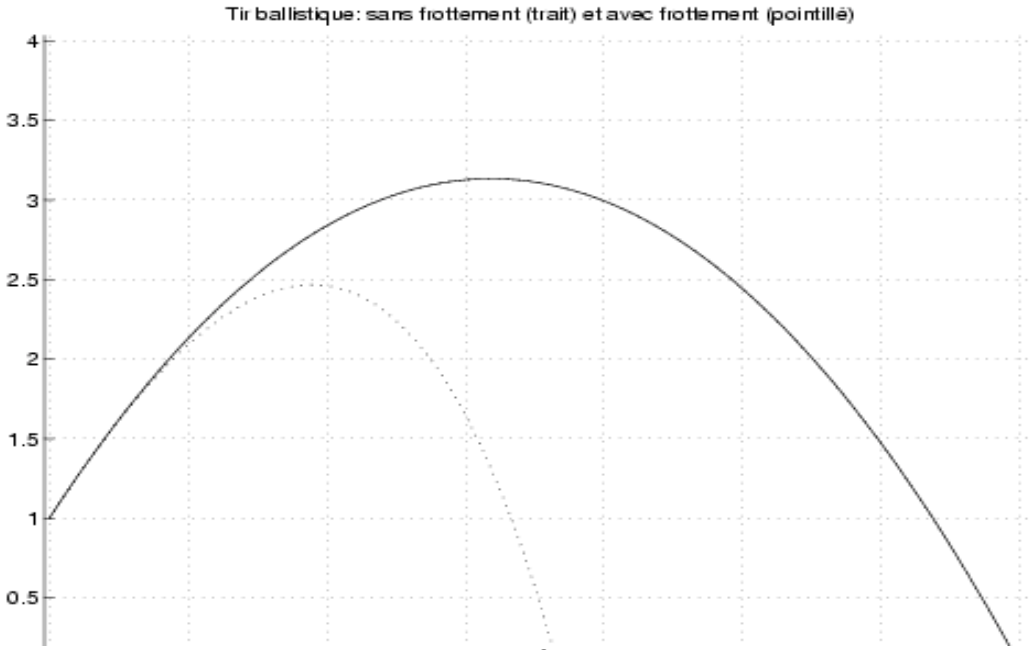
```
function Xdot= ballisticODE(t,X)
% BALLISTICODE.m   ODE pour le tir ballistique dans le vide
  global g
  Xdot = [ X(3)
           X(4)
```

```
            0
            -g   ];
```

et le M-file qui résoud le problème du tir ballistique et fait le dessin est:

```
% BALLISTIC.M tir ballistique simple dans le vide
%  exemple de résolution d'ODE avec MatLab
%  utilise la fonction ballisticODE.m qui définit les ODE à résoudre

clear; clf;
global g
g = 9.81; % const. de gravitation
% conditions initiales du tir:
h = 1; % altitude de départ
v0 = 25; % vitesse initiale
theta = pi/180*15; % angle de tir
% conditions initiales:
xin = [0, h, v0*cos(theta), v0*sin(theta)];
% temps de vol jusqu'à atteindre l'altitude 0
tmax = (xin(4) + sqrt(xin(4)^2 + 2*g*xin(2)))/g;
% solution numérique des ODE :
[t x] = ode23('ballisticODE', [0 tmax], xin);
%plot de la solution avec une interpolation faite avec des splines cubiques
N = max(x(:, 1)); xi = 0:N/100:N;
axis([0,max(x(:,1)), 0, max(x(:,2))])
grid on; hold on;
plot(xi, spline(x(:,1), x(:, 2), xi), ':r');
hold off;
```

***Exercice 6.5.1 :***
*Reprenez l'exemple du tir ballistique et introduisez une force de frottement proportionnelle à la vitesse* (voir MatLab1_fig11 ).



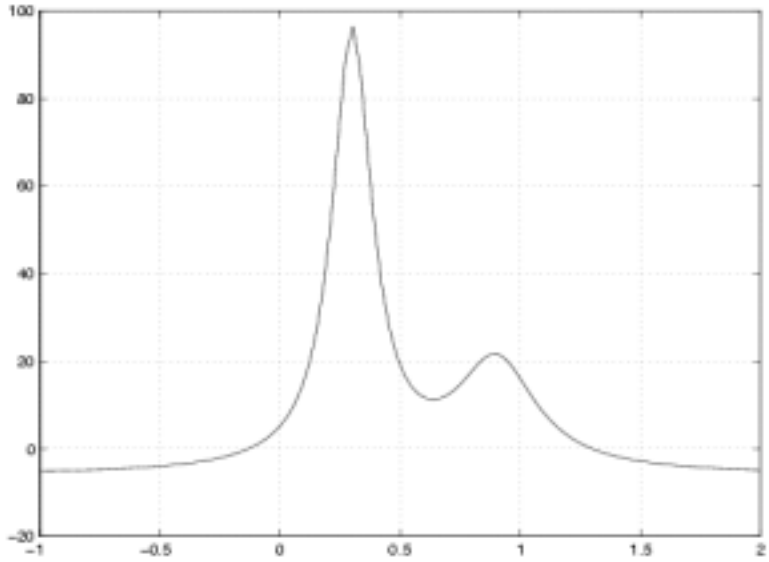

***MatLab1_fig11***

***Exercice 6.5.2 :***
*Résolvez les équations du mouvement pour les ressorts couplés de l'exercice 6.4.2*

# 6.6 Zéros et minima de fonctions; fits

**a) Recherche des zéros d'une fonction**

**hump** est une fonction prédéfinie dans MatLab (voir MatLab1_fig12 )




***MatLab1_fig12***

```
>> x = fzero('humps', 2.)%appelle fzero(Fonction_à_étudier, solution_estimée)
x =
   1.29954968258482
```

Attention, fzero ne retourne qu'un zéro:

```
% recherche les points où humps(x) = 60
>> fzero('humps(x)-60', [0.2 0.3]) % on peut donner un intervalle
ans =
   0.22497151787552
>> fzero('humps(x)-60', [0.3 0.4])
ans =
   0.37694647486211
>> fzero('humps(x)-60', [0.2 0.4])
??? Error using ==> fzero
The function values at the interval endpoints must differ in sign.
```

**b) Minimalisations**

Recherche des extrema d'une fonction.

Très utile quand vous voulez fitter une fonction paramétrique à une série de mesures afin de déterminer les paramètres de votre modèle.

***Fonctions à une dimension (modèle à 1 paramètre)*** :

```
>> fminbnd('humps', 0, 1) % cherche le minimum de la fonction humps entre 0
et 1
ans =
   0.63700873404437
```

Ici également, `fminbnd` ne retourne qu'un seul minimum:

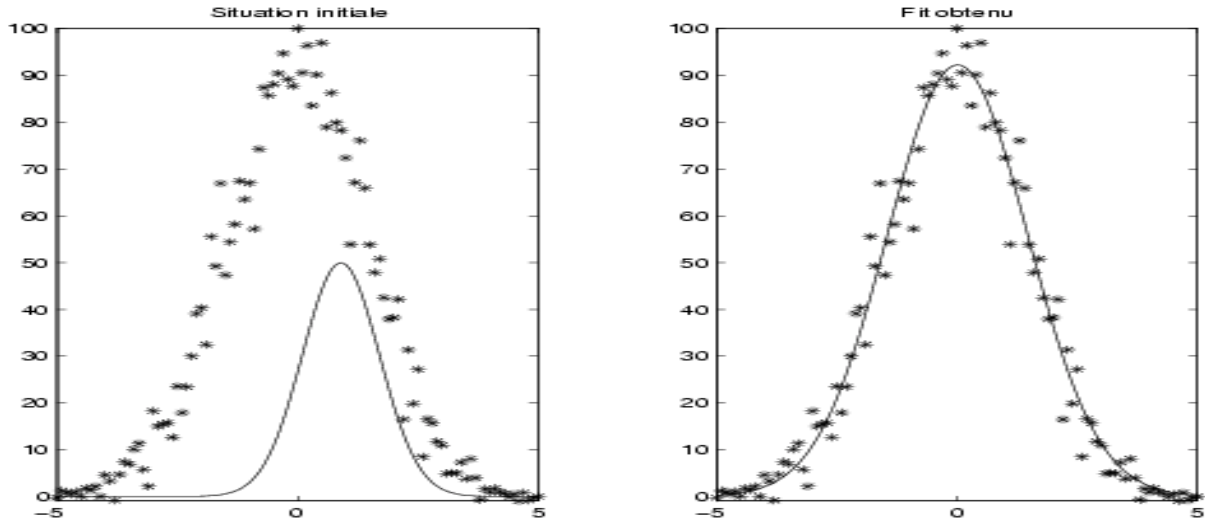
```
>> fminbnd('cos', 0, 6)
ans =
```

```
   3.14159268915185
>> fminbnd('cos', 0, 15)
ans =
   9.42477753165182
```

***Fonctions à plusieurs dimensions (modèles à plusieurs paramètres)*** :
Dans ce cas, on utilise `fminsearch`, de manière fort semblable à `fminbnd`.

*Exercice 6.6.1:*
*Ecrivez un script fit.m qui*
*1) génère une mesure m(x) décrite par une gaussienne avec du bruit (utiliser* `randn`*).*
*2) fitte un modèle gaussien g(x) en utilisant* `fminsearch`*. Les paramètres du modèle seront:*
*la position $x_0$ du centre de la gaussienne,*
*la largeur de la gaussienne* $\sigma$
*l'amplitude de la gaussienne A*
*Pour utiliser* `fminsearch`*, il faudra écrire une fonction fitgaus qui retourne la somme des résidus entre la mesure et le modèle. Le résidu au point x est: $(m(x) - g(x))^2$ . Utiliser "sum".*
*3) Construire une figure en deux parties l'une montrant la mesure avec le modèle de départ et l'autre la mesure avec le fit obtenu. (*MatLab1_fig13 )
*Rappel : gaussienne :*

$$g(x) = A \exp\left(-\frac{(x-x_0)^2}{2\sigma^2}\right)$$




***MatLab1_fig13***

**c) Autres fonctions de fit:**
D'autres fonctions de fit sont disponibles dans toolbox/stats (cf Appendice):

`nlinfit` - Nonlinear least-squares data fitting. Fournit une estimation de l'erreur sur les paramètres du fit.
`polyfit` - Least-square polynomial fitting
`regress` - Multivariate linear regression
`optim` - Optimization toolbox (`help optim`) **Voir paragraphe 6.8**

# 6.7 Filtrage de signaux

Le filtrage des signaux est un très vaste domaine. On trouvera une série de filtres prêts à l'emploi dans divers toolbox. Citons les fonctions:

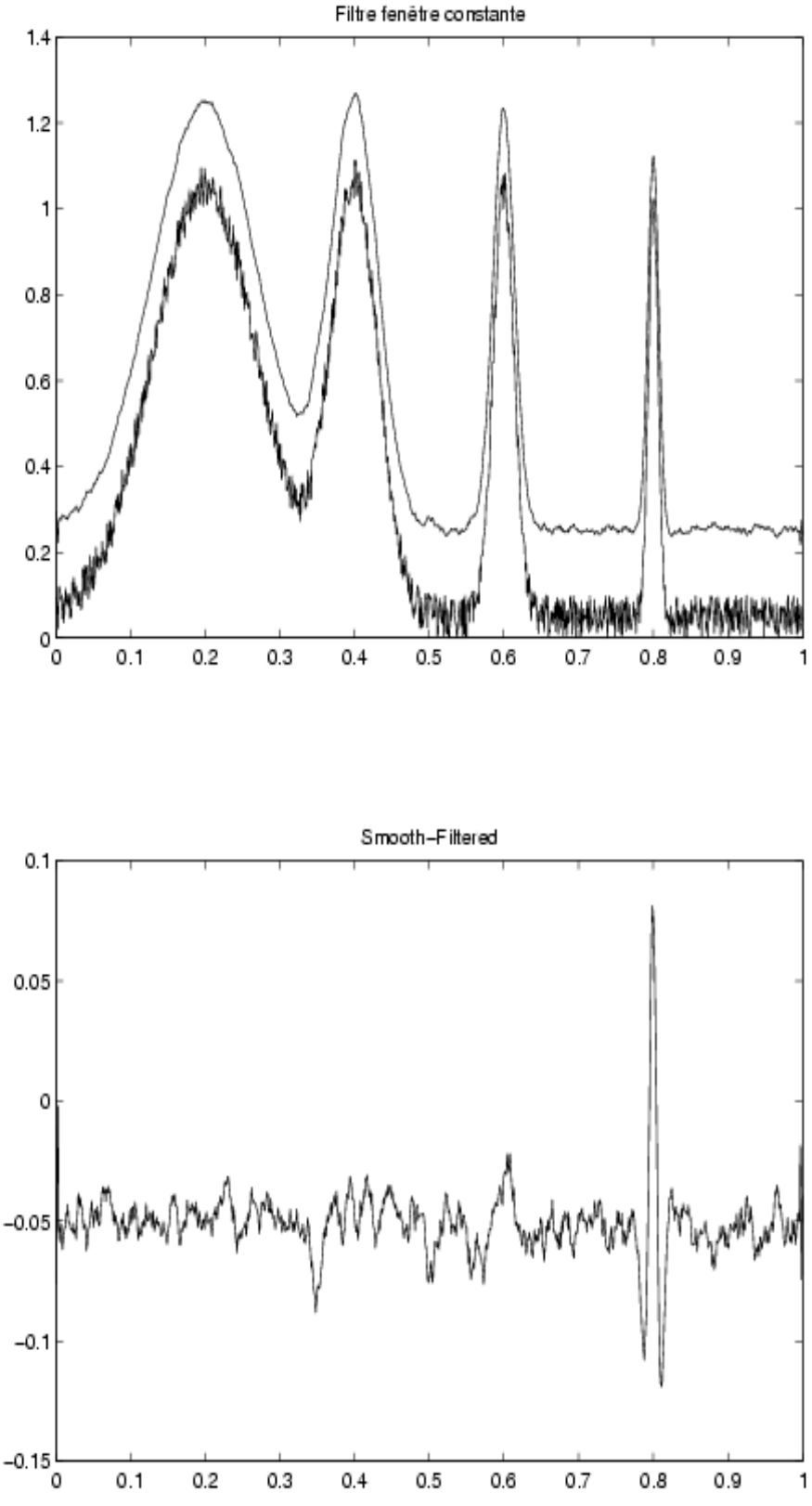
`conv` - convolution
`filter` - 1D Digital filter
`filter2` - 2D Digital filter
`latcfilt` - Lattice filter
`fir1, fir2` ...-Window based FIR filter design (Finite Impulse Response)
medfilt1 - 1D median filtering
besself - Bessel analog filter

*Exemple 1: Utilisation simple de `filter`*:
filtrage d'un signal bruyant avec une fonction carrée constante:

```
% TEST_FILTER.M   test de la fonction filter
%    Filtrage d'un signal bruyant avec une fonction
%    carrée de largeur 2*m + 1 .
%

clear
figure(1)
clf
subplot(2,1,1)
N = 1001;
bruit = 0.1
t = [0 : N-1]'/(N-1);
Smooth = exp( -100*(t-1/5).^2) + exp(  -500*(t-2/5).^2) + ...
    exp(-2500*(t-3/5).^2) + exp(-12500*(t-4/5).^2) ;
Noisy = Smooth + bruit* rand(size(t)); % avec bruit
plot(t, Noisy)
hold on
title('Filtre fenêtre constante')

m = 5; % Demi largeur de la fenêtre de filtrage
b = ones(2*m+1, 1) ./(2*m+1) ;
Filtered = filter(b, 1, Noisy);
% compensation du retard introduit par le filtre:
Filtered(1:m-1) = Noisy(1:m-1);
Filtered(m:N-m) = Filtered(m+m:N);
Filtered(N-m+1:N) = Noisy(N-m+1:N);
plot(t, Filtered+0.2,'-r')

subplot(2,1,2)
plot(t, Smooth-Filtered)
title('Smooth-Filtered')
```

Résultat: Voir figure MatLab1_fig14 .

Cet exemple montre clairement l'effet de retard introduit par `filter`.

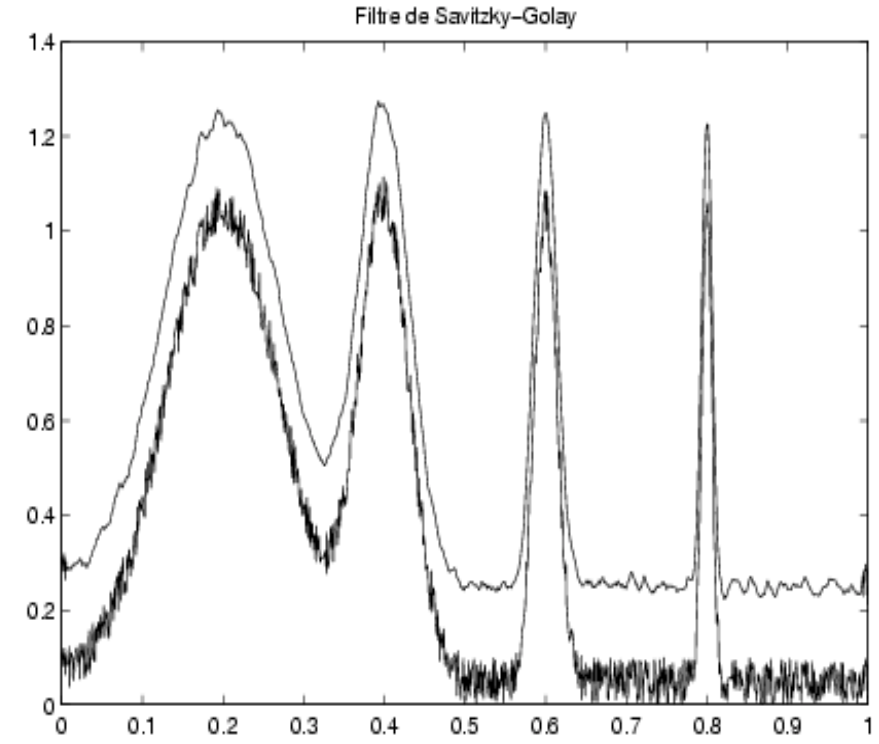


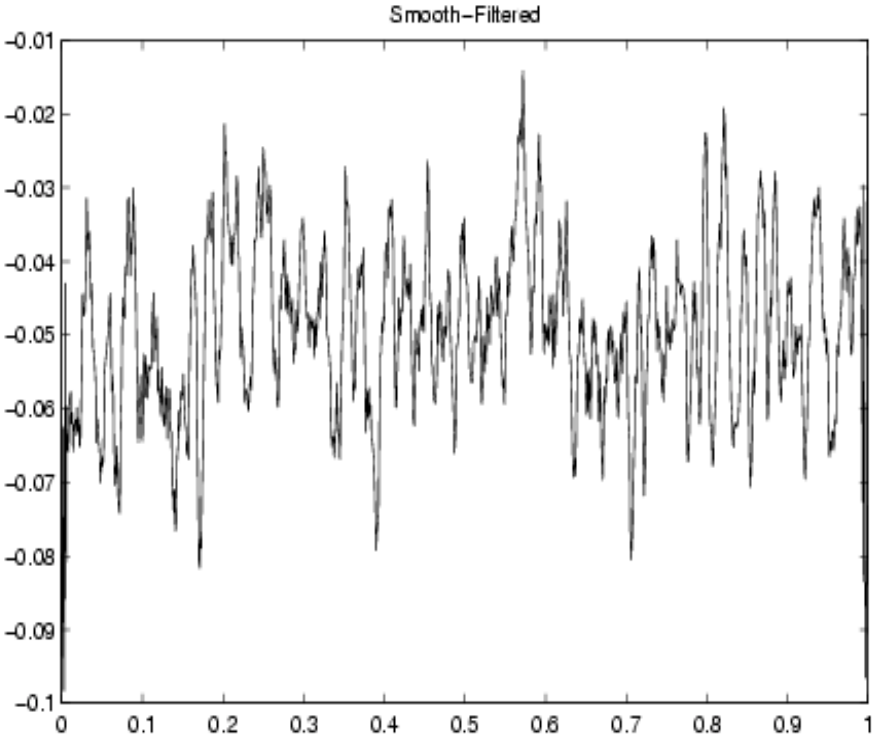

***MatLab1_fig14***

*Exemple 2 : Utilisation plus subtile de* `filter`:
Filtre de Savitzky-Golay : ce filtre revient à fitter localement un polynôme.

```
% TEST_FILTER2.M   test de la fonction filter
%    Filtrage d'un signal bruyant avec un polynôme de degré M
%    fitté localement au signal.
%   (méthode de Savitzky-Golay, d'aprés Gander & Hrebicek, chap. 9)
%

clear
figure(2)
clf
subplot(2,1,1)
N = 1001;
bruit = 0.1
t = [0 : N-1]'/(N-1);
Smooth = exp( -100*(t-1/5).^2) + exp(  -500*(t-2/5).^2) + ...
    exp(-2500*(t-3/5).^2) + exp(-12500*(t-4/5).^2) ;
Noisy = Smooth + bruit* rand(size(t)); % avec bruit
plot(t, Noisy)
hold on
title('Filtre de Savitzky-Golay')

M = 3 % 4 degré du polynome
nl= 8 % 5 nombres de points à gauche
nr= 8 % 5 nombres de points à droite
A = ones(nl+nr+1, M+1);
for j = M:-1:1
  A(:,j) = [-nl:nr]' .* A(:,j+1);
  end
[Q, R] = qr(A) ; % Orthogonal-triangular decomposition
c = Q(:, M+1) / R(M+1, M+1);
Filtered = filter(c(nl+nr+1:-1:1), 1, Noisy);
% compensation du retard introduit par le filtre :
Filtered(1:nl) = Noisy(1:nl);
Filtered(nl+1:N-nr) = Filtered(nl+nr+1:N);
Filtered(N-nr+1:N) = Noisy(N-nr+1:N);
plot(t, Filtered+0.2, '-r')

subplot(2,1,2)
plot(t, Smooth-Filtered)
title('Smooth-Filtered')
```

Résultat: Voir figure MatLab1_fig15




***MatLab1_fig15***

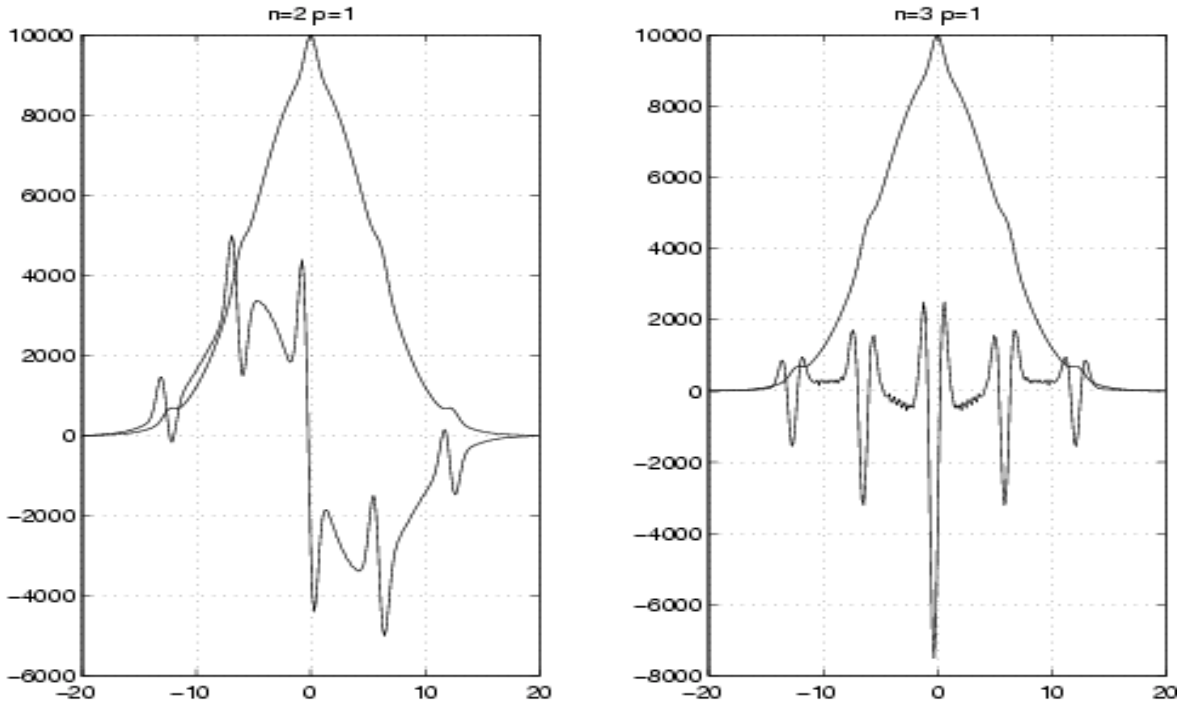
*Exercice 6.7.1:*
*a) Ecrivez un script TEST_CONV.M qui filtre le signal de TEST_FILTER.M par convolution avec une fonction gaussienne. Utiliser conv.*
*b) Déconvoluez le résultat pour tenter de retrouver le signal de départ. Utiliser deconv.*

A titre d'exemple, nous appliquons un filtre non linéaire simple à une mesure de comptage simulée. Le but du filtre est de mettre en évidence un petit signal gaussien superposé à un grand bruit de fond et noyé dans le bruit statistique.
Le principe de ce filtre est de prendre la dérivée d'ordre n du signal et de l'élever à une puissance p. On peut choisir interactivement m et n, ainsi que la statistique du signal (en donnant le taux de comptage accumulé à son pic). Le bruit est réparti selon la racine carée du signal, comme c'est le cas pour toute mesure de comptage.

```
%REX.M  démonstration de filtrage non-linéaire
%     - lit un signal propre comprenant des petits pics à localiser
%     - met ce signal à l'échelle du comptage désiré au pic
%     - ajoute du bruit selon la répartition statistique
%     - évalue le signal filtré en
%       prenant la N- ème dérivée du signal
%       élevée à la puissance p
% typical run:
%   peak count > 100000
%   with noise=1, without noise=0 [1/0]> 1
%   order of differentiation > 2
%   power > 2
%
%Lecture du signal y=f(x) dans le fichier test.dat
clear; clf
load test.dat
x=[test(:,1)];
y=[test(:,2)];
%
% scale the signal to the amplitude an at the peak
maxy=max(y);
y=y./maxy;
n=max(size(x));
an=input('peak count >');
y=y.*an;
%
% add noise following the sqrt(signal) hypothesis
noise=randn(n,1);
noise=noise.*sqrt(y);
sw=input('with noise=1, without noise=0 [1/0]>');
noise=noise.*sw;
y=y+noise;
plot(x,y);
%
% non-linear filtering
i=input(' order of differentiation >');
xdi=[x(1:n-i)];
ydi=diff(y,i);
p=input('power >');
ydp=ydi.^p;
maxydp=max(ydp);
ydp=ydp./maxydp;
fin=cumsum(ydp);
```

```
maxfin=max(fin);
minfin=min(fin);
fin=fin./(maxfin-minfin);
fin=fin.*an;
plot(x,y,'-',xdi,fin,'-')
grid
```

*Exercice 6.7.2:*
*Observez comment se comporte le filtre rex.m*
*- a) sans bruit (dans ce cas, le taux de comptage dans le pic ne joue pas de rôle). Essayez notamment n=2 p=1; n=3 p=1 et n=3 p=3. (voir* MatLab1_fig16 )
*- b) avec du bruit, en fonction du taux comptage accumulé dans votre signal*




***MatLab1_fig16***

*Exercice 6.7.3: REX2.M*
*Ajoutez dans rex.m un lissage du signal avant d'appliquer le filtre non-linéaire.*
*Observez l'effet du lissage. Un lissage simple peut être implémenté en ajoutant à un point, les valeurs de ses voisins (convolution; voir fonction "conv").*

*Exercice 6.7.4:*
*Construisez un filtre plus performant que celui implémenté dans rex.m et rex2.m*

# 6.8 Optimisation

Au paragraphe 6.6 nous avons abordé la minimalisation. On appelle généralement **optimisation** une minimalisation sous contraintes. Pour traiter ce type de problème avec Matlab, il faut avoir à disposition la "*Optimization Toolbox*" qui ne fait pas partie de Matlab lui-même. Cette toolbox est à disposition sur le réseau de l'Université, mais le nombre de licences est assez restreint.
`help optim` donne les grandes lignes des outils de cette toolbox
`helpdesk` fournit une description plus détaillée.
Au travers d'un exemple et d'un exercice, nous allons étudier deux des fonctions d'optimisation disponibles.

***Exemple d'optimisation :***

```
% LinearProg -- Programmation linéaire en Matlab: fonction LINPROG
% (D'après l'exemple du helpdesk)
%
% Trouver x=(x1, x2, x3) qui minimise f(x) = -5x1 -4x2 -6x3
% avec les contraintes:
%    x1 -  x2 +  x3 <= 20,
%   3x1 + 2x2 + 4x3 <= 42,
%   3x1 + 2x2       <= 30,
%   x1 >= 0,  x2 >= 0,  x3 >= 0

clear
disp('Problème: minimiser cx, sous les contraintes Ax <= b')
disp('avec 3 constraintes sur x')
disp(' ')
disp('Coefficients c de la fonction objectif f(x)')
c = [-5 -4 -6]
disp('Constraintes : membres de gauche')
A = [1 -1 1; 3 2 4; 3 2 0]
disp('Contraintes: membres de droite')
b = [20 42 30]
%
% Limites inférieures et supérieures de x :
n = max(size(c));
L = zeros(n,1);
U = 10^10*ones(n,1);
%
% Optimisation :
[x,lambda,exitflag] = linprog(c,A,b,[],[],L,U);
if exitflag<= 0
  disp('ERROR, exitflag =')
  exitflag
else
  disp('Solution optimale pour x')
  x
  disp('Valeur optimale de f(x)')
  z = c*x
  disp('Vérification des contraintes :')
  for i=1:n
    A(i,:)*x
  end
end
```

***Exercice 6.8.1:***
***Déterminer un acier aux propriétés données, à partir de relations empiriques.***
Le problème est exposé ci-dessous:

```
% Steels -- Synthèse d'un alliage aux propriétés spécifiques
%
% P. Comte (Institut Straumann) a établi expérimentalement des relations
% empiriques qui permettent de déterminer les propriétés d'un acier.
% On voudrait, en utilisant ces relations, trouver des aciers qui auraient
% des propriétés données.
%
% Ennoncé du problème :
%
% Les éléments composant l'alliage sont : Fe Ni C Mn N Mo Si Nb  et le Cr.
%
% Les propriétés considérées sont :
%    Sigma02 : le coefficient d'allongement à 0.2%
%    Uts     : Ultimate tensile strenght - tension de rupture
%    Gamma   : module d'élasticité
%    Cor1    : indice de corrosion
%    Prix
%
% Si on dénote par Fe, Ni, ... la concentration (%) de Fer, de Nickel, ...
% dans l'alliage, Comte a décrit les propriétés de l'alliage
% par les relations suivantes :
%
% Sigma02 = 1*Fe + 2*Ni + 36*C + 3*Mn + 50*N + 1*Mo + 2*Si + 4*Nb + 1*Cr + 7
% Uts     = 2*Fe + 1*Ni + 55*C + 3*Mn + 86*N + 2*Mo + 4*Si + 8*Nb + 0*Cr + 46
% Gamma   = 3*Fe + 1*Ni + 30*C + 1*Mn + 30*N + 2*Mo + 4*Si + 1*Nb + 3*Cr
% Cor1*10 = 1*Fe + 3*Ni +  2*C + 1*Mn +  2*N +30*Mo + 1*Si + 2*Nb + 10*Cr
% Prix    = 1*Fe + 3*Ni +  1*C + 1*Mn +  0*N + 1*Mo + 1*Si + 2*Nb + 6*Cr
%
% Ces relations sont valables dans le domaine de concentrations suivantes :
%    25   <= Fe <= 40
%    25   <= Ni <= 40
%    0.01 <= C  <= 0.1
%    1    <= Mn <= 7
%    0.1  <= N  <= 1
%    1    <= Mo <= 10
%    0.2  <= Si <= 2
%    0.3  <= Nb <= 3
%    15   <= Cr <= 35
%
% Finalement, il faut imposer que la somme des concentrations égal 100%
%
% Résolution du problème :
%
% En examinant les relations, on voit que le problème est mal déterminé,
% donc numériquement instable. Un algorithme de minimalisation tel que
% fminsearch n'est donc pas adéquat.
%
% P. Comte a développé une stratégie "Monte-Carlo" pour trouver les aciers
% aux propriétés recherchées : on tire au hasard un grand nombres d'aciers
% et on sélectionnne celui qui a les propriétés les plus proches de celles
% désirées. Cette approche nécessite beaucoup de temps CPU si on veut être
% sûr de ne pas être trop loin de la bonne solution. L'industriel nous a alors
```

```
% demandé s’il était possible d’utiliser une stratégie plus adéquate.
%
% La méthode présentée ici est basée sur la théorie de l’optimisation
% (minimalisation sous contraintes) qui consiste, à partir d’un acier de
% départ, à descendre dans l’espace de minimalisation jusqu’au minimum qui
% donnera l’acier aux propriétés optimum. Un bémol : la solution peut toujours
% n’être que un minimum local. Il faut donc la vérifier avec soin.
%  Une difficulté résolue par cette méthode : le respect de contraintes, qui
% peuvent être de plusieurs types : inégalités, égalités, limites inférieures
% et/ou supérieures.
%
% En Matlab, les problèmes d’optimisation  peuvent se résoudre aisément avec
% une des fonctions du "toolbox" d’optimisation. Nous utiliserons fminimax.
%
% Pour fminimax, il faut fournir une fonction de "résidus", ce que l’on peut
% faire en évaluant:
%
%      residu d’une propriété de l’acier x =
%                (propriété désirée - propriété obtenue pour l’acier x)^2
%
% La fonction (ResiduProprietes dans l’exemple ci-dessous) doit fournir un
% vecteur contenant le résidu pour chaque propriété. Comme les paramètres
% de la fonction sont définis, on devra passer les autres quantités
% nécessaires par l’intermédiaire de variables globales.
```

**Appendice 1 :**

**Les principales “toolbox” disponibles sur le serveur de MatLab à l’Université**

| | | |
|---|---|---|
| calfem | - | Finite Element Toolbox |
| communications | - | Communication Toolbox |
| compiler | - | MATLAB Compiler |
| control | - | Control System Toolbox |
| ident | - | System Identification Toolbox |
| images | - | Image Processing Toolbox |
| local | - | Preferences |
| nnet | - | Neural Network Toolbox |
| optim | - | Optimisation Toolbox |
| signal | - | Signal Processing Toolbox |
| simulink | - | model/system-based design |
| stateflow | - | Logic and Command Toolbox |
| stateflow/sfdemos | - | Stateflow Demonstrations |
| stats | - | Statistics Toolbox |
| symbolic | - | Symbolic Math Toolbox |

La liste détaillée dépend de l’environnement de travail. Elle est fournie par la commande `ver`.

## Appendice 2 : Tableaux de fonctions

Extrait de : *Mastering Matlab 5*, D. Hanselman, B. Littlefield, Prentice Hall, 1998.

### *Operations*

| Element-by-element Operation | Representative Data<br>a = [$a_1$ $a_2$ . . . $a_n$], b = [$b_1$ $b_2$ . . . $b_n$], c= <*a scalar*> |
|---|---|
| Scalar addition | a+c = [$a_1$+c $a_2$+c . . . $a_n$+c] |
| Scalar multiplication | a*c = [$a_1$*c $a_2$*c . . . $a_n$*c] |
| Array addition | a+b = [$a_1$+$b_1$ $a_2$+$b_2$ . . . $a_n$+$b_n$] |
| Array multiplication | a.*b = [$a_1$*$b_1$ $a_2$*$b_2$ . . . $a_n$*$b_n$] |
| Array right division | a./b = [$a_1$/$b_1$ $a_2$/$b_2$ . . . $a_n$/$b_n$] |
| Array left division | a.\b = [$a_1$\$b_1$ $a_2$\$b_2$ . . . $a_n$\$b_n$] |
| Array exponentiation | a.^c = [$a_1$^c $a_2$^c . . . $a_n$^c] |
| | c.^a = [c^$a_1$ c^$a_2$ . . . c^$a_n$] |
| | a.^b = [$a_1$^$b_1$ $a_2$^$b_2$ . . . $a_n$^$b_n$] |

### *Syntaxe d'adressage*

| Array Addressing | Description |
|---|---|
| A(r,c) | Addresses a subarray within A defined by the index vector of desired rows in r and index vector of desired columns in c. |
| A(r,:) | Addresses a subarray within A defined by the index vector of desired rows in r and all columns. |
| A(:,c) | Addresses a subarray within A defined by all rows and the index vector of desired columns in c. |
| A(:) | Addresses all elements of A as a column vector taken column by column. If A(:) appears on the left-hand side of the equal sign, it means fill A with elements from the right side of the equal sign without changing its shape. |
| A(i) | Addresses a subarray within A defined by the single index vector of desired elements in i, as if A was the column vector, A(:). |
| A(x) | Addresses a subarray within A defined by the logical array x. x must be the same size as A. |

## *Matrices spéciales*

| Matrix | Description |
|---|---|
| `[]` | The empty matrix. |
| `compan` | Companion matrix. |
| `eye` | Identity matrix. |
| `gallery` | Over 50 test matrices. |
| `hadamard` | Hadamard matrix. |
| `hankel` | Hankel matrix. |
| `hilb` | Hilbert matrix. |
| `invhilb` | Inverse Hilbert matrix. |
| `magic` | Magic square. |
| `ones` | Matrix containing all ones. |
| `pascal` | Pascal triangle matrix. |
| `rand` | Uniformly distributed random matrix with elements between 0 and 1. |
| `randn` | Normally distributed random matrix with elements having zero mean and unit variance. |
| `rosser` | Symmetric eigenvalue test matrix. |
| `toeplitz` | Toeplitz matrix. |
| `vander` | Vandermonde matrix. |
| `wilkinson` | Wilkinson eigenvalue test matrix. |
| `zeros` | Matrix containing all zero elements. |

## *Fonctions matricielles*

| Function | Description |
|---|---|
| `A^n` | Exponentiation, e.g., `A^3` = A · A · A. |
| `balance(A)` | Scale to improve eigenvalue accuracy. |
| `cdf2rdf(A)` | Complex diagonal form to real block diagonal form. |
| `chol(A)` | Cholesky factorization. |
| `cholinc(A,DropTol)` | Incomplete Cholesky factorization. |
| `cond(A)` | Matrix condition number. |
| `condest(A)` | 1-norm matrix condition number estimate. |
| `condeig(A)` | Condition number with respect to repeated eigenvalues. |
| `det(A)` | Determinant. |
| `eig(A)` | Vector of eigenvalues. |
| `[V,D]=eig(A)` | Matrix of eigenvectors `V`, and diagonal matrix of eigenvalues `D`. |
| `[V,D]=eigs(A)` | A few eigenvectors and eigenvalues. |
| `expm(A)` | Matrix exponential (preferred method). |
| `expm1(A)` | M-file implementation of `expm(A)`. |
| `expm2(A)` | Matrix exponential via Taylor series. |
| `expm3(A)` | Matrix exponential via eigenvalues and eigenvectors. |
| `funm(A,'fun')` | Compute general matrix function. |

## *Fonctions matriceilles (suite)*

| Function | Description |
|---|---|
| `inv(A)` | Matrix inverse. |
| `logm(A)` | Matrix logarithm. |
| `lscov(A,b,V)` | Least squares with known covariance. |
| `lu(A)` | Factors from Gaussian elimination. |
| `luinc(A,DropTol)` | Incomplete LU factorization. |
| `nnls(A,b)` | Nonnegative least squares. |
| `norm(A)`<br>`norm(A,1)`<br>`norm(A,2)`<br>`norm(A,inf)`<br>`norm(A,p)`<br>`norm(A,'fro')` | Matrix and vector 2-norm,<br>1-norm,<br>2-norm<br>infinity norm,<br>P-norm (vectors only),<br>Frobenius norm, i.e., `norm(A(:))`. |
| `normest(A)` | Estimate of matrix 2-norm. |
| `null(A)` | Null space. |
| `orth(A)` | Orthogonalization. |
| `pinv(A)` | Pseudoinverse. |
| `planerot(x)` | Given's plane rotation. |
| `poly(A)` | Characteristic polynomial. |
| `polyeig(A0,A1,...AP)` | Solve polynomial eigenvalue problem. |
| `polyvalm(A)` | Evaluate matrix polynomial. |
| `qr(A)` | Orthogonal-triangular decomposition. |
| `qrdelete(Q,R,j)` | Delete column from `qr` factorization. |
| `qrinsert(Q,R,j,x)` | Insert column in `qr` factorization. |
| `qz(A,B)` | Generalized eigenvalues. |
| `rank(A)` | Number of linearly independent rows or columns. |
| `rref(A)` | Reduced row echelon form. |
| `rsf2csf(U,T)` | Real schur form to complex schur form. |
| `schur(A)` | Schur decomposition. |
| `sqrtm(A)` | Matrix square root. |
| `subspace(A,B)` | Angle between two subspaces. |
| `svd(A)` | Singular value decomposition. |
| `svds(A,k)` | A few singular values. |
| `trace(A)` | Sum of diagonal elements. |

| Data Analysis Function | Description |
|---|---|
| `corrcoef(x)` | Correlation coefficients. |
| `cov(x)` | Covariance matrix. |
| `cplxpair(x)` | Sort vector into complex conjugate pairs. |
| `cumprod(x)` | Cumulative product. |
| `cumsum(x)` | Cumulative sum. |
| `cumtrapz(x,y)` | Cumulative trapezoidal integration. |
| `del2(A)` | Discrete Laplacian. |
| `diff(x)` | Differences between elements. |
| `gradient(Z,dx,dy)` | Approximate gradient. |
| `histogram(x)` | Histogram or bar chart. |
| `max(x), max(x,y)` | Maximum component. |
| `mean(x)` | Mean or average value. |
| `median(x)` | Median value. |
| `min(x), min(x,y)` | Minimum component. |
| `prod(x)` | Product of elements. |
| `rand(x)` | Uniformly distributed random numbers. |

| Data Analysis Function | Description |
|---|---|
| `randn(x)` | Normally distributed random numbers. |
| `sort(x)` | Sort in ascending order. |
| `sortrows(A)` | Sort rows in a ascending order. |
| `std(x)` | Standard deviation of columns normalized by $N - 1$. |
| `subspace(A,B)` | Angle between two subspaces. |
| `sum(x)` | Sum of elements in each column. |
| `trapz(x,y)` | Trapezoidal integration of $y = f(x)$. |

## *Transformées de Fourier*

| Function | Description |
|---|---|
| `conv` | Convolution. |
| `conv2` | 2-D convolution. |
| `fft` | Discrete Fourier transform. |
| `fft2` | 2-D discrete Fourier transform. |
| `fftn` | *N*-dimensional discrete Fourier transform. |
| `ifft` | Inverse discrete Fourier transform. |
| `ifft2` | 2-D Inverse discrete Fourier transform. |
| `ifftn` | *N*-dimensional inverse discrete Fourier transform. |
| `filter` | Discrete time filter. |
| `filter2` | 2-D discrete time filter. |
| `abs` | Magnitude. |
| `angle` | Four-quadrant phase angle. |
| `unwrap` | Remove phase angle jumps at 360-degree boundaries. |
| `cplxpair` | Sort array into complex conjugate pairs. |
| `fftshift` | Shift FFT results so negative frequencies appear first. |
| `nextpow2` | Next higher power of 2. |

# Table des Matières

# 6. Quelques applications de Matlab



# Appendices