# 6. L'Analyse de Signal

## 6.1 La Transformée de Fourier

| | |
|---|---|
| **fft** | transformée de Fourier |
| **ifft** | transformée de Fourier inverse |
| **fft2** | transformée de Fourier 2D |
| **ifft2** | transformée de Fourier inverse 2D |

Exemple de la fonction fft:
On génère un signal qui est la somme de deux sinus et on regarde la sortie de la fonction **fft**.
Regardez fft_sin.m.

```
fft_sin.m
```

```
%script pour montrer une analyse fft
Tinitial =0;      %d'abord on define le base du temps des données
Tfinal = 1;
NbEch = 300;      %Nombre d'échantillons

Tech = (Tfinal - Tinitial)/(NbEch-1);
Fech = 1/Tech;    %on détermine la période et la fréquence d'échantillonnage

Freq1 = 10;       % les fréquences des composants du signal
Freq2 = 25;

t = Tinitial : Tech: Tfinal;  % on génère le vecteur du temps et le signal
x = sin(2*pi*Freq1*t) + 0.5*sin(2*pi*Freq2*t);

y = fft(x);       % on applique l'analyse de Fourier

% la sortie est complex. On prend le modulus complex comme amplitude
% et on divise par Nb de donneées pour arriver aux 'vrais' amplitudes
m = abs(y)/NbEch;

p = unwrap(angle(y));   % la phase est calculée.

f = (0:NbEch-1)*Fech/NbEch;   % pour créer l'echelle de la fréquence

subplot(3,1,1),plot(t,x),xlabel('temps(s)')
subplot(3,1,2),plot(f,m),xlabel('fréquence(Hz)'),grid on
subplot(3,1,3),plot(f,p*180/pi),xlabel('fréquence(Hz)'),ylabel('phase
(deg)'),grid on
```

## 6.2 Interpolation

L'interpolation consiste à estimer les valeurs des points qui se trouvent entre des points donnés. MATLAB est capable de faire de l'interpolation sur des données mono-dimensionnelles ou plus généralement à n dimension. On trouve deux familles de calcul de l'interpolation dans MATLAB:

- L'interpolation polynomial
- L'interpolation basée sur FFT

| | |
|---|---|
| **interp1** | interpolation polynomial à une dimension |
| **interpft** | interpolation basée sur FFT à une dimension |
| **interp2** | interpolation polynomial deux dimension …. |

Syntaxe : yi = **interp**1 (x, y, xi, 'méthode')

x = vecteur d'abscisses monotones(croissantes ou décroissantes)
y = vecteur des points donnés correspondant aux valeurs de x
xi = vecteur des valeurs pour lesquelles on veut estimer y
'méthode' = nom de l'algorithme utilisé : 'nearest','linear','spline','cubic'
Si les valeurs de x sont régulières, l'exécution est accélérée en utilisant '*methode'

**Comparaison des différentes méthodes**
Pour cette comparaison, on utilise un fichier de données census.mat fourni par MATLAB

```
» clear
» load census
» whos
  Name       Size         Bytes  Class
  cdate      21x1           168  double array
  pop        21x1           168  double array

Grand total is 42 elements using 336 bytes

» plot(cdate,pop,'*')
» low = min(cdate)
low =
        1790

» high = max(cdate)
high =
        1990

» xi = low:high;
» yi = interp1(cdate,pop,xi,'nearest');
» figure
» plot(cdate,pop,'+',xi,yi,'r')
» figure
» yi = interp1(cdate,pop,xi,'linear');
» plot(cdate,pop,'+',xi,yi,'r')
» figure
» yi = interp1(cdate,pop,xi,'spline');
» plot(cdate,pop,'+',xi,yi,'r')
» figure
» yi = interp1(cdate,pop,xi,'cubic');
» plot(cdate,pop,'+',xi,yi,'r')
```

L'interpolation à deux dimensions est très importante dans le traitement de l'image. La syntaxe est la suivante

**Syntaxe** : zi = **interp2** (X, Y, Z, Xi, Yi,'méthode')

X,Y,Z = matrices des points mesurés
Xi, Yi = matrices des valeurs pour lesquelles on veut estimer l'altitude (les résultats du **meshgrid**)
'méthode' = nom de l'algorithme utilisé : 'nearest','bilinear','spline','bicubic'

**Exercice** :

1. Comparez les différentes méthodes de l'interpolation bi -dimensionelle. Vouz pouvez travailler avec la fonction de démonstration peaks de MATLAB.

| | |
|---|---|
| >> [X,Y] = meshgrid(-3:1:3,-3:1:3 ) | création d'un maillage cartsien du plan x,y |
| >> Z = peaks(X,Y) | calcul de la matrice de l'altitude |
| >> surf (X,Y,Z) | |

L'objective est d'affiner la maillage en utilisant l'interpolation. Donc …
>> [Xi ,Yi] = meshgrid (-3:0.25:3,-3:0.25:3);

Il existe aussi sous MATLAB des fontions de triangulation et d'interpolation des données dispersées (des nuages de points).
**Exemples** : convhull, delaunay, tsearch

## 6.3 L'ajustement des courbes

- **l'ajustement d'un polynôme par la méthode des moindres carrés**

| | |
|---|---|
| **polyfit** | trouve les coefficients d'un polynôme |
| **polyval** | calcule les valeurs d'un polynôme(ex. utilisant les résultats de polyfit**)** |

Les polynômes dans MATLAB sont représentés sous forme de vecteurs de lignes composés des coefficients classés en ordre décroissant. Le polynôme p(x) = x3 -6x2 +11x -6 est écrit dans MATLAB comme >> p = [1 -6 11 -6]

**Exemple** :

| | |
|---|---|
| » xm = 0:10; | vecteur d'abscisses(mesures) |
| » p = [1 2 0]; | polynôme x2 + 2x |
| » ym = polyval(p,xm) + rand(size(xm)); | évaluation de p et on ajoute du bruit |
| » plot(xm,ym,'x') | |
| » pf = polyfit(xm,ym,4) | interpolation par un polynôme d'ordre 4 |

```
p =
   0.0018   -0.0361    1.2284    1.5023    0.8249
```

| | |
|---|---|
| » y = polyval(pf,xm); | calcule les ordonnées de ce nouveau polynôme en fonction des abscisses |

```
» plot(xm,ym,'x',xm,y)                    affichage et comparaison
» res = ym - y;
» plot(xm,res,'x')
```

- **L'ajustement aux équations non-linéaires**

La boite à outils Optimisation contient la fonction **curvefit** qui permet l'ajustement à des fonctions non-linéaires

**Syntaxe** de base x = **curvefit**('fun',x0,xdata,ydata)

fun = nom d'une autre fonction (.m) qui contient la fonction
x0 = les conditions initiales
xdata,ydata = les données à ajuster
x = les coefficients de sortie

**Exemple /Exercice**
But : trouver la forme et les coefficients de l'équation qui représentent le mieux les données

- Dans le fichier fitting.mat il y a des données à ajuster. Charger ce fichier. Regarder l'allure des données.

- A votre avis quelle est la forme d'une équation qui peut représenter ces données ? Ecrivez l'equation dans un fichier.m( avec l'en-tête function f = fun(x,xdata) dans la forme suivante
  f = x(1) +x(2)**f1**(xdata) +x(3)**f2**(xdata) …….

- Créer un vecteur x0 des valeurs de départ
- Appeler la fonction **curvefit** avec les parametres 'fun', x0 et les données.
- Pour comparer les données avec votre résultat, faites appel à **fun** en lui passant les coefficients évalués par **curvefit**. Tracez les données et la courbe idéale. Vous pouvez également évaluer les résidus

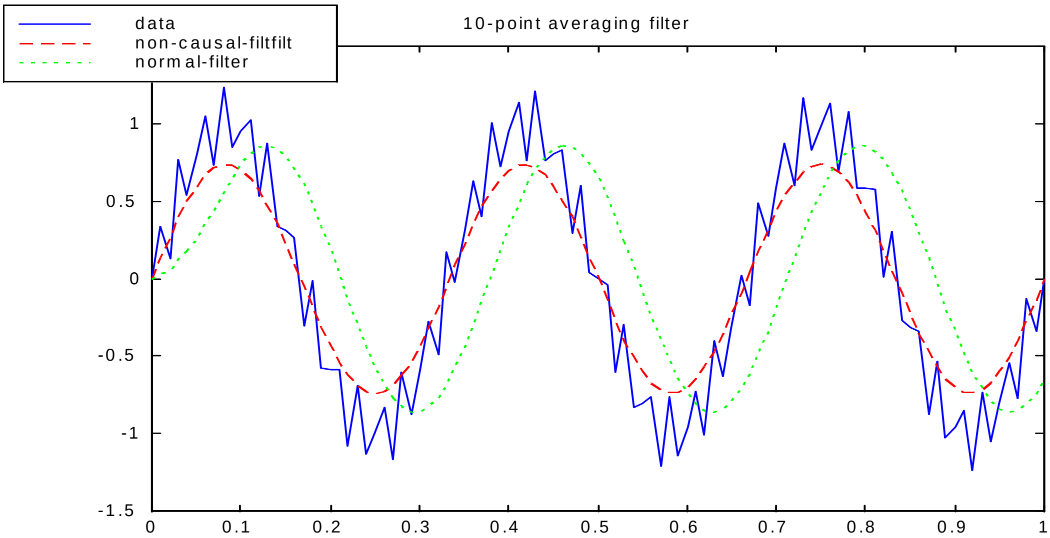
## 6.4 Le filtrage des donnees

Un filtre peut être caractérisé par sa fonction de transfert. Les coefficients de cette fonction de transfert se divisent en deux groupes, l'un pour le numérateur et l'autre pour le dénominateur. MATLAB stocke les coefficients dans deux vecteurs ligne, le vecteur **b** correspond au numérateur, et **a** au dénominateur.

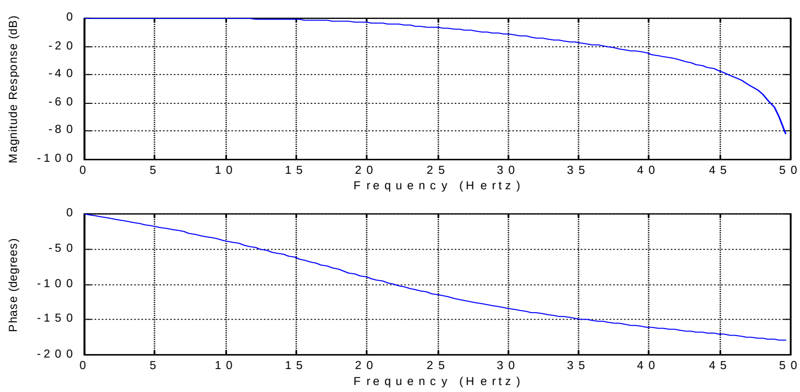
```
» Fs = 100;
» t = 0:1/Fs:1;
» x = sin(2*pi*t*3) + 0.25*sin(2*pi*t*40);      -génération des données
» b = ones(1,10)/10;                             -coefficients de filtrage pour un
                                                    filtre de 10 points moyennes
» y = filtfilt(b,1,x);                           -forward and reverse digital
                                                    filtering
» yy = filter(b,1,x);                            -one-dimensional digital filter

» plot(t,x,t,y,'r--',t,yy,'g:')
```

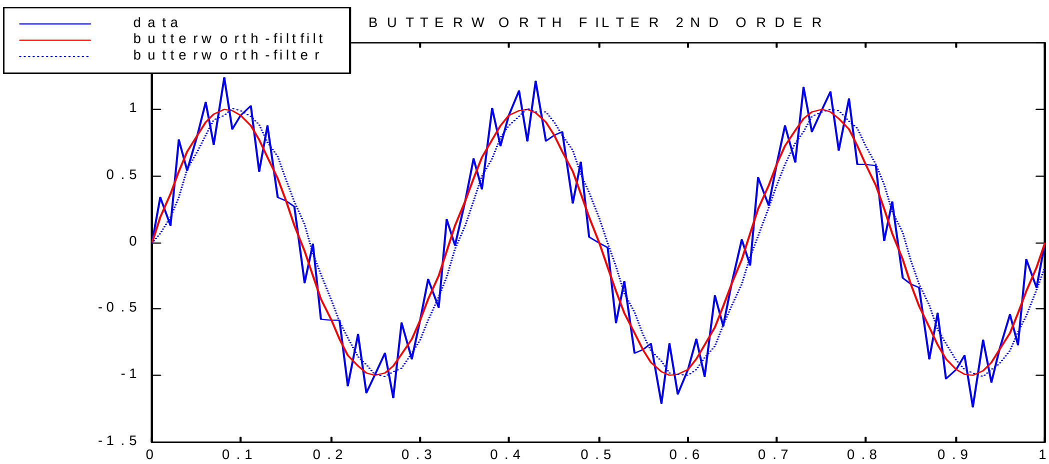
» legend('data','non-causal-filtfilt','normal-filter')
» title('10-point averaging filter')




» [b,a] = butter(2,20/50); - on genère les coefficients pour un filtre butterworth
» freqz(b,a,128,100) - affichage des caractéristiques du filtre




» butty = filter(b,a,x);
» buttyy = filtfilt(b,a,x);
» plot(t,x,t,buttyy,'r',t,butty,'g:')



# 7. Chaines de caractères

Les chaînes de caractères sont traitées comme des matrices. Chaque caractère correspond à un élément de la matrice.
La manipulation d'une matrice de caractères se fait de la même manière que ce qu'on a déjà vue.
Exemples
» a = 'azerty';
» a(1,2)
ans =
z
» e = [a,'abcd'] concaténation horizontale
e =
azertyabcd

» F = [a; 'toto']
??? All rows in the bracketed expression must have the same number of columns.

» F = [a; 'toto '] concaténation verticale. ATTENTION à la longueur de la chaîne
F =
azerty
toto

» G = str2mat(a,'toto') cette fonction fait une concaténation verticale en gérant les blancs automatiquement
G =
azerty
toto

## 7.1 Conversion

| | |
|---|---|
| **num2str** | valeur numérique → chaîne de caractères |
| **str2num** | chaîne de caractères →valeur numérique |
| **mat2str** | matrice → chaîne de caractères |
| **dec2hex** | entier en base 10 → chaîne hexadécimale |
| **real** | chaîne de caractères → codes ascii |
| **isstr** | egal 1 si c'est une chaine de caractères |
| **char** | codes ascii → chaîne de caractères |

Exemple
```
>> x = 45.678;
>>size(x)
ans =
        1        1
>>y = num2str(x);
>>size(y)
ans =
        1        6
```

## 7.2 Comparaison, recherche, remplacement

| | |
|---|---|
| **strcmp** | comparaison de 2 chaînes de caractères |
| **strncmp** | comparaison des n premiers caractères de 2 chaînes |
| **findstr** | Recherche d'une chaîne dans une autre |
| **strrep** | Remplace une chaîne par une autre |

Exemples:

```
>> c1 = 'bonjour'; c2 = 'bonne annee';
>>strcmp(c1,c2)
ans =
        0
>>strncmp(c1,c2)
ans =
        1
>> i = findstr(c2,'ne')
i =
        4       9
>> c3 = strrep(c2,'annee','fete')
c3 =
        bonne fete
```

## 7.3 Evaluation d'une chaîne de caractères

La commande **eval**(chaîne) évalue la chaîne de caractères comme une commande MATLAB
Exemple:

```
» pi1 = 4*atan(1)
pi1 =
   3.1416

» s = '4*atan(1)';
» pi2 = 2*eval(s)
pi2 =
   6.2832
```

Cette commande est très utile si, par exemple, on veut changer le nom d'une variable dans chaque itération d'une boucle

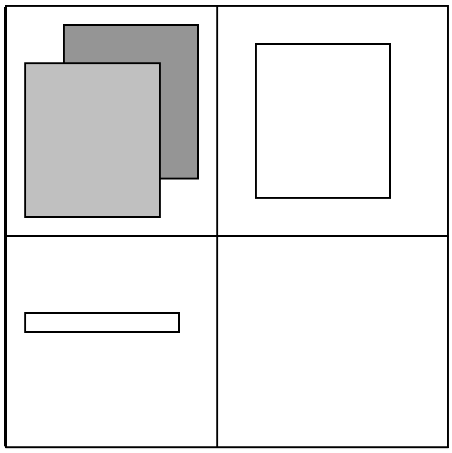
```
%carré magic
for k=5:2:11
   n= num2str(k);
   nom= ['M',n,'= magic(k)'];
   eval(nom)
end
```

# 8. Les Tableaux de Cellules et les Structures

Les tableaux de cellules et les structures dans MATLAB offrent un format de stockage optimisé.

## 8.1 Les Tableaux de Cellules

Un tableau de cellules est une variable constituée de cellules, celles-ci peuvent contenir n'importe quel type de données: matrice, structure, cellule..



### 8.1.1 Construction d'un tableau de cellules

Les accolades { } sont utilisées pour représenter des cellules.  Trois syntaxes existent pour la construction d'un tableau de cellules

**Syntaxe 1** :

```
» A{1,1}  = rand(3,3,3);
» A{1,2}  = magic(4);
» A{2,1}  = 'sujet 1';
```

**Syntaxe 2**

```
» A(1,1) = {rand(3,3,3)};
» A(1,2) = {magic(4)};
» A(2,1) = {Sujet 1};
```

**Syntaxe 3**

```
» A = {rand(3,3,3),  magic(4); 'Sujet 1', [ ] };
```

Pour afficher, on a aussi une choix à faire selon les détails voulues

a) >> A donne la forme du tableau et le type du contenu de chaque cellule

b) celldisp(A)  affiche le contenu détailé de A

**c)** cellplot(A) donne une représentation graphique de A

### 8.1.2 Accès aux éléments d'une cellule.

Les indices, les accolades et les parenthèses donnent accès aux éléments

```
>> Sub = A{2,1};
>> Donnees = A{1,2}
>> cond = A{1,2}(1,:)
```

### 8.1.3 Manipulation des cellules

Voici quelques exemples de manipulation des cellules

```
» b = A(1,:)
b =
   [3x3x3 double]    [4x4 double]
```

b est construit avec la première ligne de A

```
» c= reshape(A,4,1)
c =
   [3x3x3 double]
   'Sujet 1'
   [4x4   double]
             []
```

A est transformé en un tableau de cellules de 4 lignes et de 1 colonne.

```
» A(:,2) = []
A =
   [3x3x3 double]
   'Sujet 1'
```

on supprime la deuxième colonne

```
» M = max(max(A{1,1}))
M(:,:,1) =
   0.9318

M(:,:,2) =
   0.8462

M(:,:,3) =
   0.7095
```

calcul du max de la matrice contenue dans la première cellule de A

### 8.1.4 Cellules et chaînes de caractères

Les chaîne de caractères peuvent être stockées dans des cellules. Cela évite de compléter les chaînes par des blancs lorsqu'on crée un tableau.

```
>> Mois = {'Janvier';'Fevrier';'Mars';'Avril'}
```

| | |
|---|---|
| >> Mois2 = char(Mois) | transforme les 4 cellules en tableau de chaîne de caractères. Les blancs sont insérés automatiquement |
| >> Mois3 = cellstr(Mois2) | transforme le tableau en 4 cellules - les blancs sont enlevés |

On peut créer des tableaux multidimensionnels de cellules. On n'est pas obligé  a avoir une correspondance entre les pages des cellules

## 8.2 <u>Les Structures</u>

Les structures sont composées de champs, ceux-ci peut contenir  n'importe quel type de données. Comme les tableau de cellules les structures peuvent être multidimensionnels.

Chaque champ est construit un par un. Dans un premier temps; la structure n'a qu'une dimension

**Exemple** : >>

» sujet.nom = 'Jean';
» sujet.poid = 120;
» sujet.data = rand(5);

La variable sujet est agrandie comme suit

» sujet(2).nom = 'Jeanne';
» sujet(2).poid = 140;
» sujet(2).data = rand(5);

Pour accéder au champs d'une structure il suffit de taper le nom du structure, un point suivi par le nom du champs

>> subjet(2).data(:,3) donne accès à la troisième colonne dans le champs data

Les commandes suivantes permettent aussi d'accéder au contenu d'une structure

| | |
|---|---|
| **getfield** | Lecture d'un champ |
| **setfield** | Ecriture d'un champ |
| **fieldnames** | Lecture de la liste des champs d'une structure |
| **rmfield** | Suppression d'un champ |

La fonction ex-struct est utilisée pour afficher les informations contenues dans la structure Sujet créée toute à l'heure

ex_struct.m

```
function ex_struct(S)
%Affichage de la structure "Sujet"

for i=1:length(S)
   figure
   plot(S(i).data)
   title(S(i).nom,'fontsize',18)
end
```

**Exemple** d’une analyse en batch

batch.m

```
%script pour gerer une analyse en batch
clear all;


% on cree une liste des fichiers a traiter
File = dir('nuage*.txt'); % on determine la liste des fichiers a traiter

num_fichiers = size(File,1);

for count = 1: num_fichiers
   id_fich_in = fopen(File(count).name,'r');
   File(count).Data = fscanf(id_fich_in, '%f',[2,inf])';
   fclose(id_fich_in);

   % ici on appelle les fonctions d'analyse .....
   File(count).Norm_data = normalisation(File(count).Data);

   File(count).Moy_donnees = mean(File(count).Data);
   File(count).Ecart_T = std(File(count).Data);

   %..... etc etc

   % creation du nom du fichier de sortie - la meme racine que le fichier
   % d'entree mais avec une extension differente
   nom_sortie = File(count).name(1:(end-3));
   nom_sortie = [nom_sortie, 'out'];

   % ouverture et ecriture des donnees
   id_fich_out = fopen(nom_sortie,'w');
   fprintf(id_fich_out,' donnees moyennes\n');
   fprintf (id_fich_out,'%f',File(count).Moy_donnees);
   fprintf(id_fich_out,' \n ecart-types\n');
   fprintf (id_fich_out,'%f',File(count).Ecart_T);
   fprintf(id_fich_out,' \n donnees normalisees\n');
   fprintf (id_fich_out,'%f %f\n',File(count).Norm_data');
   fclose(id_fich_out);

   %affichage
   figure
   plot(File(count).Norm_data)
   title([File(count).name(1:(end-4)),' averages =
',num2str(File(count).Moy_donnees)])

end

% analyse inter sujet
% ex. on calcule la moyenne pour tous les fichier pour col 1 et 2
av_data = [File.Moy_donnees];
av_col1 = mean(av_data(1:2:end))
av_col2 = mean(av_data(2:2:end))


% ....et si on avait la boite a outils 'Statistics, on serait pret
% à ecrire l'article !
```

## 9. Fonctions de Fonctions

MATLAB peut travailler en termes de fonctions mathématiques qu'il représente sous la forme de M-files. Ces fonctions mathématiques peuvent être utilisées comme arguments d'entrée d'un ensemble de routines MATLAB qui permettent de faire :

- de l'intégration numérique
- de l'optimisation
- de l'ajustement des courbes
- la résolution d'équations non-linéaires.

L'ensemble de ces routines MATLAB porte le nom générique de fonctions de fonctions et la plupart se trouve dans le répertoire funfun dans MATLAB/Toolbox.

Considérons la fonction $f(x) = \dfrac{1}{(x - 0.3)^2 + 0.01} + \dfrac{1}{(x - 0.9)^2 + 0.06} - 6$

Cette fonction peut être utilisée comme une entrée pour une fonction de fonction. Cette fonction s'ecrit dans humps.m

```
function [out1,out2] = humps(x)
%HUMPS  A function used by QUADDEMO, ZERODEMO and FPLOTDEMO.
%   Y = HUMPS(X) is a function with strong maxima near x = .3
%   and x = .9.
%
%   [X,Y] = HUMPS(X) also returns X.  With no input arguments,
%   HUMPS uses X = 0:.05:1.
%
%   Example:
%      plot(humps)
%
%   See QUADDEMO, ZERODEMO and FPLOTDEMO.

if nargin==0, x = 0:.05:1; end

y = 1 ./ ((x-.3).^2 + .01) + 1 ./ ((x-.9).^2 + .04) - 6;

if nargout==2,
  out1 = x; out2 = y;
else
  out1 = y;
end
```

### 9.1 Tracé

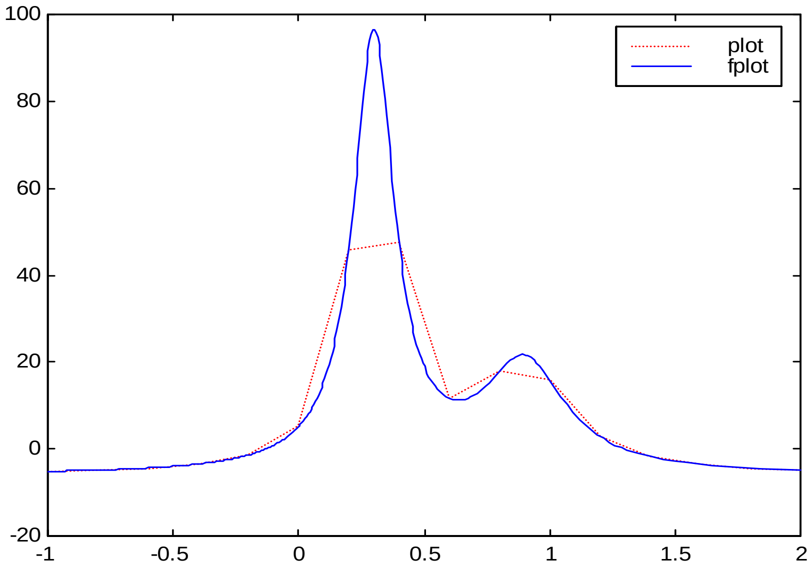
Deux possibilités se présentent pour tracer un fonction :

- définir soi-même le vecteur d'abscisse et utiliser **plo**t
- utiliser la commande **fplot** qui ne nécessite que les bornes de l'intervalle du tracé.

» x = -1:0.2:2;    définition des abscisses
» plot(x,humps(x),'r:')
» hold on

La fonction **fplot** permet de représenter graphiquement des fonctions en ne donnant que les bornes des abscisse. Le pas d'échantillonnage est défini automatiquement en prenant en compte la pente.

» **fplot**('humps',[-1 2])




## 9.2 Intégration Numérique

| | |
|---|---|
| **quad**<br>**quad8**<br>**dblquad** | Intégration par la méthode de Simpson à pas adaptatif<br>Intégration par la méthode de Newton-Cotes à pas adaptatif<br>Double intégration |

**Exemple :**

» q = quad('humps',0,1)
q =
  29.8583

## 9.3 Optimisation

| | |
|---|---|
| **fmin**<br>**fmins**<br>**fzero** | recherche du minimum de fonctions monovariables<br>recherche du minimum de fonctions multivariables<br>recherche du zéro de fonctions monovariables |

**Exemple:**

» xmin = fmin('humps',0.5,1) recherche d'un minimum entre les abscisses 0.5 et 1
xmin =
   0.6370

» ymin = humps(xmin) calcul de l'ordonnée de ce minimum
ymin =
  11.2528

» plot(xmin,ymin,'+r') affichage du minimum par une croix rouge

» xz=fzero('humps',0)
xz =
   -0.1316

» plot(xz,0,'ob')

La **Toolbox d'Optimisation** complète ces fonctions

## 9.4 Résolution de systèmes d'équations différentielles

MATLAB permet de résoudre les systèmes d'équations différentielles de la forme

$$dX/dt = f(t,X)$$

connaissant la valeur initiale de la solution $X(t_o) = X_o$
Les solvers d'équations différentielles sous MATLAB utilisent des méthodes numériques d'intégration. Ce sont des solvers à pas variable.
Les algorithmes de résolution d'un système d'équations différentielles sont classés en deux catégories. La première regroupe les algorithmes qui s'appliquent aux systèmes d'équations non-raides, la seconde contient ceux pour les systèmes raides.

### 9.4.1 Algorithmes de résolution de systèmes d'équations différentielles non raides

| | |
|---|---|
| **ode45** | basé sur l'algorithme de Runge-Kutta et utilisé généralement pour un premier essai dans la résolution d'un système. |
| **ode23** | basé sur l'algorithme de Runge-Kutta . L'ordre est simplement inférieur, ce qui fait qu'il est plus efficace lorsque les tolérances sont moins exigeante. |
| **ode113** | est une adaptation de l'algorithme d'Adams. Cet algorithme est d'ordre variable(son pas d'intégration varie). |

### 9.4.2 Algorithmes de résolution de systèmes d'équations différentielles raide

| | |
|---|---|
| **ode15s** | basé sur les formules de différentiation numérique. Il est d'ordre variables. Si l'équation est raide ou si ode45 est inefficace, il faut essayer ode15s |
| **ode23s** | Cet algorithme est à pas unique, il est plus efficace que ode15s lorsque les tolérances sont plus larges…. |

### 9.4.3 Appel général d'un solver

L'écriture syntaxique employée par les algorithmes ODE est la suivante:

**[t,X]** = `solver` **('fichier', tspan, Xo options, p1, p2…..)**

où `solver` est un des algorithmes précédents
Les arguments d'entrée sont :

'fichier' chaîne représentant le nom du fichier décrivant le système ODE- le ODEfile
**tspan** vecteur de temps. Pour un vecteur de deux éléments tspan = $[t_o\ t_{final}]$
**Xo** vecteur avec les conditions initiales du problème.
**options** argument qui peut être créé par **odeset** pour contrôler l'algorithme

**p1, p2**… paramètres de la fonction

Les arguments de sortie sont :

**t** vecteur colonne des points temporels
**X** tableau des solutions. Chaque ligne de X correspond à une solution pour un point temporel retourné dans t

Il est également possible d'obtenir des statistiques sur les performance de l'algorithme

**[t,X,s]** = `solver` **('fichier', tspan, Xo, options, p1, p2…..)**

où s est un vecteur colonne de six éléments

- nombre de pas réussis
- nombre d'accès ayant échoue
- …….

La fonction **odeset**, sans paramètre d'entrée permet d'obtenir la liste des propriétés d'un algorithme de résolution. Donc en tapant **odeset** on peut voir les options de défaut. Cette commande aussi permet de personnaliser les options……

**9.4.4. Exemple de résolution d'equations.**

On va comparer l'efficacité des différents algorithmes pour résoudre l'équation suivante

dx/dt = -1000(x - sin(t)) + cos(t)

D'abord on programme cette équation sous forme d'une fonction MATLAB dans un ODEfile

equation.m

```
function dx = equation(t,x)
% example de résolution d'une équation différentielle

dx = -1000*(x - sin(t)) + cos(t);
```

- **algorithmes pour problèmes non-raides**

» [t,x,s] = ode45('equation',[0 10],0);
» s(1)
ans =
 3013

» plot(t,x)
» zoom on

» [t,x,s] = ode23('equation',[0 10],0);
» s(1)
ans =
 3976

» [t,x,s] = ode113('equation',[0 10],0);

```
» s(1)
ans =
        6200
» plot(t,x)
```

- **algorithmes pour équations raides**

```
» [t,x,s] = ode15s('equation',[0 10],0);
» s(1)
ans =
    40
» plot(t,x)

» [t,x,s] = ode23s('equation',[0 10],0);
» s(1)
ans =
   224
» plot(t,x)
```

L'argument s(1) indique le nombre de pas réussis, c'est à dire le nombre de dates par lesquelles est passé le solver pour résoudre l'équation différentielle tout en satisfaisant les critères d'erreurs définis (odeset). Plus ce nombre est faible, plus grande est la rapidité de résolution.  On note que dans le cas particulier d'une équation raide (equation.m) les solvers raides permettent de gagner un facteur 10 en temps de résolution.