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 $f(x) = \frac{1}{(x - 0.3)^2 + 0.01} + \frac{1}{(x - 0.9)^2 + 0.06} - 6$

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

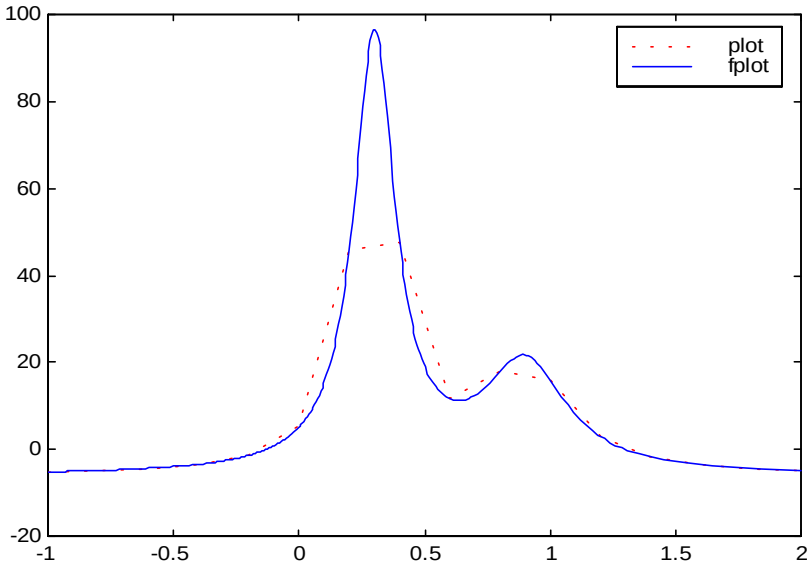
Deux possibilités se présentent pour tracer un fonction :

- 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	Intégration par la méthode de Simpson à pas adaptatif
quad8	Intégration par la méthode de Newton-Cotes à pas adaptatif
dblquad	Double intégration

Exemple :
» q = quad('humps',0,1)
q =
29.8583

9.3 Optimisation

fmin	recherche du minimum de fonctions monovariables
fmins	recherche du minimum de fonctions multivariables
fzero	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.