X. Algorithmes Optimisation - Corrigé

1. Analyse points extrêmes

Pour la fonction f:

$$\Re^2 \longrightarrow \Re \text{ Définie } \quad f(x,y) = \sin(\frac{1}{2}x^2 - \frac{1}{4}y^2 + 3) * \cos(2x + 1 - e^y)$$

Pour trouver les points critiques, on utilise les variables symboliques syms de Matlab (Symbolic Toolbox) qui permettent de trouver facilement les dérivées partielles[1]

```
syms x y
f=2*x^3+2*y^3-9*x^2+3*y^2-12*y;
fx=diff(f,x)
fy=diff(f,y)
```

On utilise la commande `solve` pour trouver l'endroit où les dérives partielles sont simultanément égales à zero

```
S=solve(fx,fy)
```

Pour examiner les camps de S, on écrit…

```
[S.x,S.y]
```

Et on trouve les points critiques de la fonction f suivante

```
ans =

[0,  1]
[0, -2]
```

[1] Si vous utilisez des versions antérieures (Matlab 4):

```
f='sin(1/2*x^2-1/4*y^2+3)*cos(2*x+1-exp(y))';
fx=diff(f, 'x')
fy=diff(f, 'y')
```

```
[3,  1]
[3, -2]
```

Pour classifier les points on utilise le test de la seconde dérivée, qui consiste à evaluer le signe du déterminant de la matrice hessiène $|H| = f_{xx}(x,y) * f_{yy}(x,y) - f^2{}_{xy}(x,y)$

On définis $\partial_{xx} f$, $\partial_{yy} f$ et $\partial_{xy} f$.

```
fxx=diff(fx,x)
fyy=diff(fy,y)
fxy=diff(fx,y)
```

Et on les évalue à chaque point trouvé précédement. Comme f est définit comme une valeur symbolique, la commande a écrire est :

```
a=subs(f,[x,y],[0,1])
```

(x,y)	$f_{xx}(x,y)$	$f_{xx}(x,y) * f_{yy}(x,y) - f^2{}_{xy}(x,y)$	**Classification**
(0,1)	-7	-324	Point selle
(0,-2)	20	324	Minimum local
(3,1)	-34	324	Maximum local
(3,-2)	-7	-324	Point selle

Pour visualiser les solutions et s'assurer de la bonne classification des points, on crée une maille de points, et on défini la fonction :

```
[x,y]=meshgrid(-5:0.1:5);
z=2*x.^3+2*y.^3-9*x.^2+3*y.^2-12*y;
mesh(x,y,z)
xlabel('x')
ylabel('y')
zlabel(z='f(x,y)')
```

Pour mieux localiser les points on utilise une carte de contours et on y situe les points :

```
contour(x,y,z,50)
hold on
plot(0,1,'r*');
```

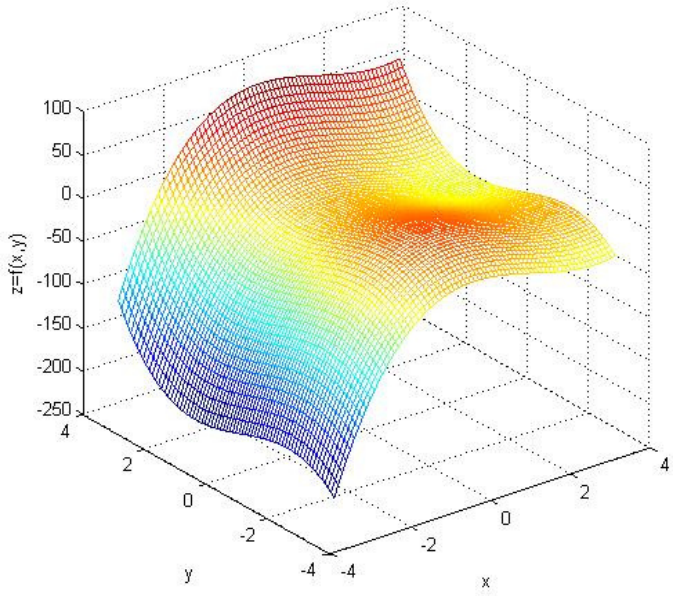

Figure 1- Mesh de la fonction

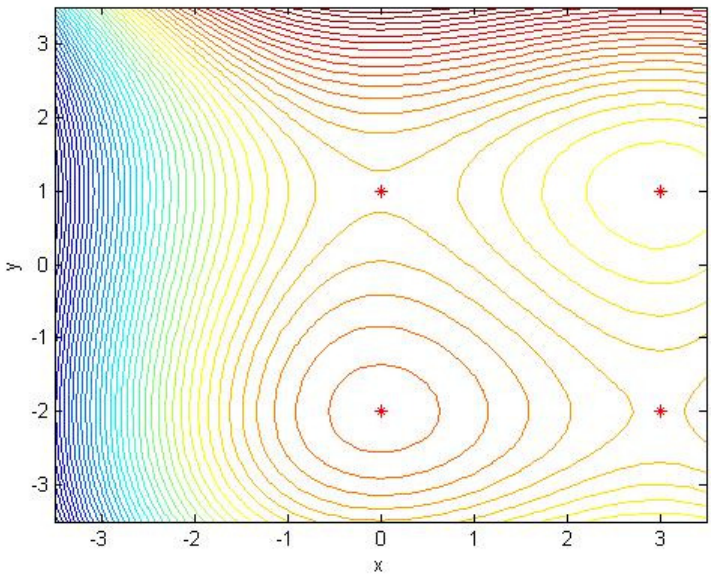

Figure 2- carte de contours de la fonction f(x,y)

2. Minimisation : Méthode de 'Golden Search'

Pour visualiser la fonction

```
title('Optimisation: Visualisation fonction')
xlabel('x')
ylabel('f(x)')
```

On observe que le minimum se trouve dans l'interval [0,1].

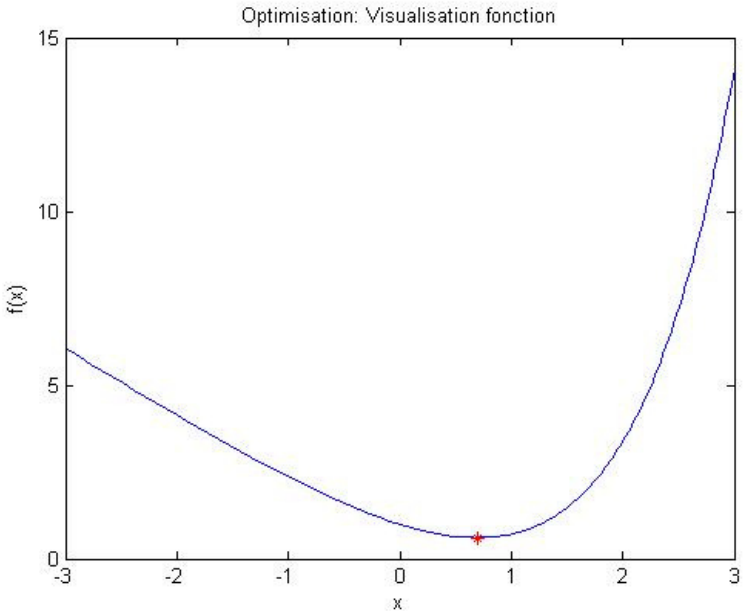

Figure 3- Visualisation graphique f(x)

Pour trouver le minimum avec Matlab de façon automatique, comme f(x) est une fonction continue et dérivable on peut utiliser :

```
min=fminbnd('fonction',0,2)
feval('fonction',min)
```

On obtient un minimum pour x=0.6932, y= 0.6137. Pour programmer l'algorithme, on va créer deux archives, une qui va contenir la fonction objet (fonction.m) et l'autre qui va nous permettre de définir la méthode golden.m. Une structure en plusieurs fichiers permet l'utilisation d'un même algorithme pour évaluer différents fonctions objets facilement.

fonction.m

```
% ALGORITHMES OPTIMISATION: "Golden Search"
% Fonction y=f(x) à minimiser
function y = f(x)
y = exp(x) - 2*x;
```

golden.m

```
function [a,b] = golden(f,a,b,eps,N)
% ALGORITHMES OPTIMISATION: "Golden Search"
% Minimisation de la fonction f en [a,b]
% f est continue on [a,b]
% f est unimodal (juste un minimum en [a,b])
% N est le nombre maxime d'itérations
% Critère arret: b-a < eps

%Nombre d'or
c = (-1+sqrt(5))/2;
inia=a;
inib=b;

%Calcul x1,x2,f(x1),f(x2)
x1 = c*a + (1-c)*b;
fx1 = feval(f,x1);
x2 = (1-c)*a + c*b;
fx2 = feval(f,x2);

%Sorties graphiques
fprintf('-----------------------------------------------------
\n');
fprintf(' x1 x2 f(x1) f(x2) b - a\n');
fprintf('-----------------------------------------------------
\n');
fprintf('%.4e %.4e %.4e %.4e %.4e\n', x1, x2, fx1, fx2, b-a);

for i = 1:N-2

%Cas 1: f(x1)<f(x2)
if fx1 < fx2
b = x2;
x2 = x1;
fx2 = fx1;
x1 = c*a + (1-c)*b;
fx1 = feval(f,x1);

%Cas 2: f(x1)>f(x2)
else
a = x1;
x1 = x2;
```

```
fx1 = fx2;
x2 = (1-c)*a + c*b;
fx2 = feval(f,x2);
end;

fprintf('%.4e %.4e %.4e %.4e %.4e\n', x1, x2, fx1, fx2, b-a);

%Critère d'arrêt
if (abs(b-a) < eps)
fprintf('Convergence après %d iterations\n', i);
return;
end;

%Graphiques erreur
axisx(i)=i+1;
axisy(i)=abs(b-a);
figure (2)
plot(axisx,axisy,'-or')
title('Optimisation: Algorithme Golden Search')
xlabel('Nombre Iterations')
ylabel('Erreur')

end;
%Message non convergence
fprintf('Non convergence après %d iterations\n', N);
```

Pour lancer l'algorithme on écrit la commande

```
[a,b] = golden('fonction',0,1,0.001,20)
```

On obtient que notre minimum se trouve entre (0.6927, 0.6934). Le minimum obtenu avec la commande fminbnd est 0.6932 $\in$ (0.6927, 0.6934).

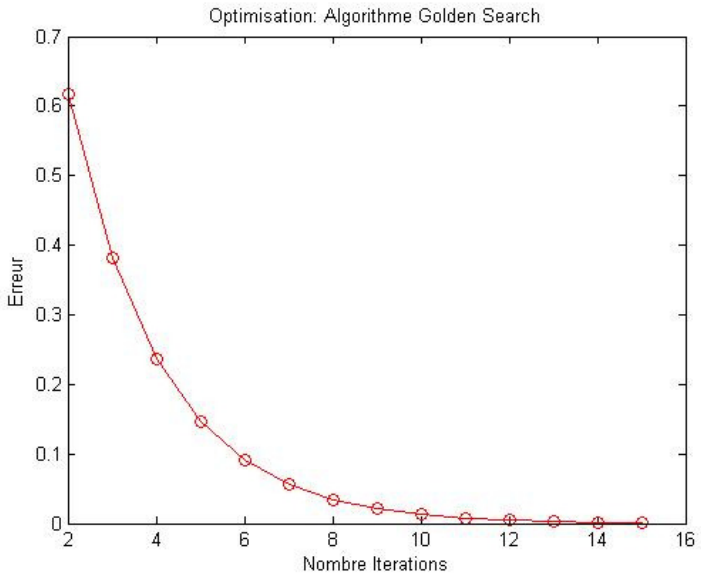

Figure 4- Evolution de l'erreur avec les itérations. Algorithme Golden Search

3. Minimisation : Méthode de Newton

```
function [xvect,xdif,fx,it] = newton(f,x0,N,eps)

%INPUT
% f : Fonction objectif
% x0:approximation initiale
% N: Nombre maxime itérations
% eps=1e-10:tolérance
%
% OUTPUT
% xvect: Stockage valeurs
% xdif : Différence deux valeurs consécutives
% fx: Valeur fonction a chaque itération
% nit: Numéro itérations requerees
```

Le program s'initialise avec

```
nit = 0;
xvect = x0;
x = x0;
```

et évalue ddf(x) et df(x)

Pour arrêter le processus on a deux conditions : que le nombre d'itération ne surpasse pas le nombre d'itérations maximal défini et que la tolérance ne soit pas atteinte. Pourtant :

```
%Algorithme Newton
while(err > eps & nit <= N)
    nit=nit+1;
    x=xvect(nit);
    f=eval(fun,x);
    df=feval(dfun,x);
    ddf=feval(ddfun,x);

    if (ddf==0), err=eps*1e-10; %erreur
      disp('Impossible diviser par zero');
    else,
        xn=x-dfx/ddfx;
        err=abs(xn-x);
        xdif=xn-x;
        x=xn; %actualiser valeur
        xvect=[xvect;x];
        f=feval(f,x)];
        dfx=feval(df,x);
        ddfx=feval(ddf,x);
end
end
```

Il est important de définir un arrêt du calcul par exemple lorsque la seconde dérivée est zéro (dans ce cas la méthode ne peut être utilisée).

4. Références

[1] MATLAB Help

[2] Wikipedia

[3] Amos, Gilat, 2007. *Matlab, an introduction with application,* John Willey and Sohn, Inc.