31/03/2017

**TP 6 : Optimisation numérique**

**Points de STEINER**

Bilal ALAUDDIN

Imane HAOUDI

Table des matières

[I- Introduction 2](#_Toc478773077)

[II- Optimisation du problème de Steiner avec 3 points et calcul de performance : 2](#_Toc478773078)

[1- Fonction Calgrad(fct,x) : 2](#_Toc478773079)

[2- Programme Gradient à pas constant : 3](#_Toc478773080)

[3- Test sur les fonctions données J1, J2 et J3 avec la méthode à pas constant : 5](#_Toc478773081)

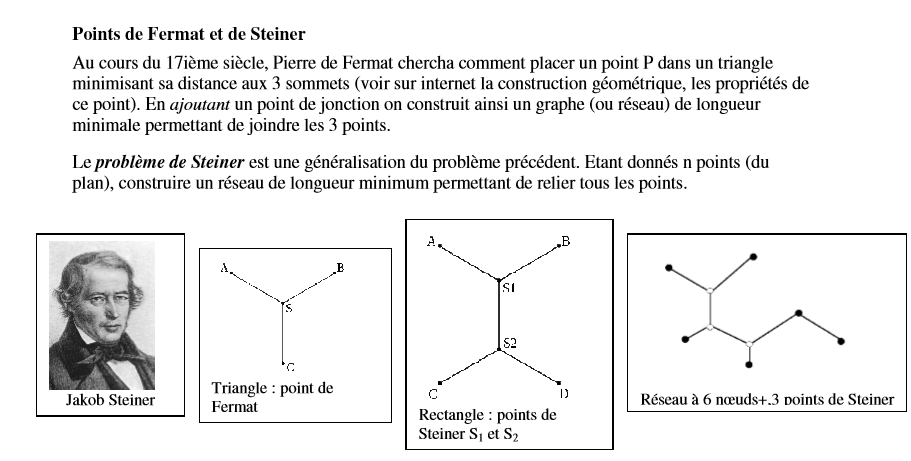
[4- Courbes de performances sur le problème de Steiner : 6](#_Toc478773082)

[5- Programme Gradient à pas optimal : 7](#_Toc478773083)

[III- Conclusion : 9](#_Toc478773084)

# Introduction

Dans le cadre de notre formation à l’EMSE et dans le cadre du module des méthodes numériques, nous avons pu confronter un modèle d’optimisation assez classique et complexe mais basique pour nous aider à maitriser au mieux les bases des méthodes numérique. Le problème présenté dans ce TP est le problème de Steiner. Il consiste à trouver le nombre et la position de points de jonctions permettant de relier n points d’un réseau tout en minimisant sa longueur.

Durant ce TP on cherche à résoudre le problème de Steiner à travers plusieurs méthodes d’optimisation. Nous allons analyser ces méthodes suivant la précision et l’efficacité de chacune d’elles à travers le nombre d’itérations et comparer entre elle.

# Optimisation du problème de Steiner avec 3 points et calcul de performance :

# Fonction Calgrad(fct,x) :

|  |  |
| --- | --- |
| **Fonction Calgrad(fct,x)** | **Test sur la fonction : f(x)=x^2 +7x** |
| function grad=Calgrad(fun,x)  E=zeros(length(x),1);  grad=zeros(length(x),1);  for i=1:length(x)  eup=10^(-5);  E(i)=eup;  grad(i)=(fun(x+E)-fun(x))/eup;  E(i)=0;  end  end  E est un vecteur unitaire change dans chaque itération sivant la variable i. | Solution analytique du Gradient :  Grad(f,x)=2x  Notre fonction :  function z=f(x)  z=x^2+7;  Notre script de comparaison:  x=0:0.1:5;  for i=1:length(x)  GradAnal(i)=(2\*x(i));    A(i)=Calgrad(@f,x(i));  end  plot(x,A,'r')  hold on  plot(x,GradAnal,'b')  title('courbes du gradien analytique et de ma fonction Calgrad')  xlabel('x')  ylabel('gradient')  les courbes qu’il donne sont très similaires qu’on n’arrive pas à distinguer entre les deux fonctions.la bleu étant celle du fct analytique, la rouge étant celle de notre fct Calgrad |

# Programme Gradient à pas constant :

elseif strcmp(methode,'Pas Constant') % gradient Ã  pas constant

rho=0.2;

x1=x0;

x2=x0-rho\*Calgrad(critere,x0);

%eup=10^(-10);

while(critere(x1)>critere(x2)+eup)

x1=x2;

x2=x1-rho\*Calgrad(critere,x1);

nb\_pas\_opt=nb\_pas\_opt+1;

end

xopt=x2;

|  |  |  |
| --- | --- | --- |
|  | **Test sur point de Fermat (triangle) puis Steiner (carré)** | **Comparaison avec les fcts d’optimisation déjà données** |
| **3 points** | Pour epsilon=10^(-2)    Longueur minimale obtenue : 1.734065  nombre d'appels à la fonction objectif : 25  coordonnées des points solution  X =  0.5002  0.3287 | Méthode de Newton Matlab    longueur minimale obtenue : 1.732051  nombre d'appels à la fonction objectif : 19  coordonnées des points solution  X =  0.4999  0.2883 |
| **4 points** | Avec le même epsilon.  Longueur minimale obtenue : 2.733313  nombre d'appels à la fonction objectif : 31  coordonnées des points solution  X =  0.4801  0.2756  0.4807  0.7348 | Méthode de Newton Matlab    longueur minimale obtenue : 2.732215  nombre d'appels à la fonction objectif : 31  coordonnées des points solution  X =  0.4860  0.2880  0.4865  0.7114 |

On a presque les mêmes résultats pour les points optimaux cependant il y a une différence au niveau des nombre d’itération (coût de calcul) qui fait qu’un programme est évalué mieux que l’autre et qu’il est plus adapté.

# Test sur les fonctions données J1, J2 et J3 avec la méthode à pas constant :

Le tableau suivant montre la difference entre la méthode à pas constantet les méthodes de Matlab pour les fonctions J1, J2 et J3.

|  |  |  |
| --- | --- | --- |
|  | **Méthodes à pas constant** | **Méthodes de Newton Matlab** |
| J1  Y en fonction de X | La fonction en 3D :    xopt =  0.0152  -0.0022  nb\_appel =  37 | On a la même fonction en 3D.  xopt =  1.0e-04 \*  0.0296  -0.1063  nb\_appel =  22  On remarque ne pas avoir vraiment le même point optimal c’est dû au fait que le point optimal est (0,0) et cette méthode est plus précise que celle du gradient à pas constant. |
| J2  Y en fonction de X | Avec un rho=0.2 on n’obtient pas de solution convergente le programme passe beaucoup de temps avant qu’on le force à s’arrêter. Le shéma qu’on obtient est celui là :    On divise notre rho par deux pour voir le changement qui peut donner (si la solution converge), on obtient :    On voit bien que cette fois-ci la solution converge et on a les résultats suivants :  xopt =  4.7913  4.9065  nb\_appel =  28  La représentation en 3D est la suivante : | xopt =  4.7885  4.9082  nb\_appel =  25  Si on essaie avec un autre X0 qui est plus proche d’un min local que du min globla :    xopt =  -9.4583  4.9858  nb\_appel =  25  On remarque bien que notre solution et programme dépend fortement du point de départ. |

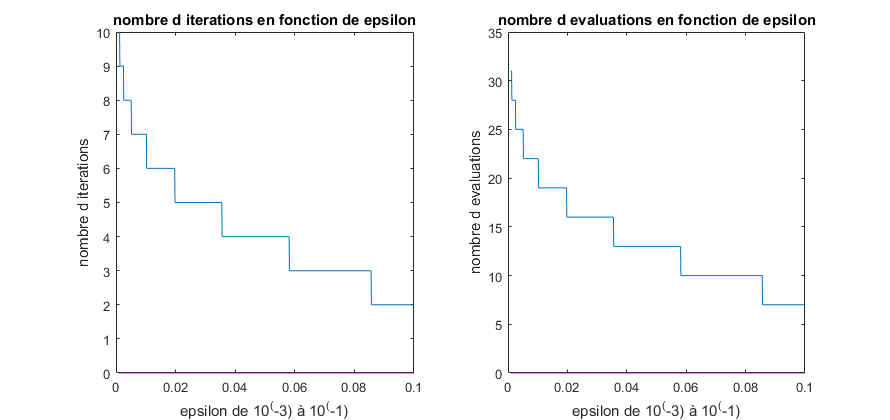
Ce tableau nous montre bien que notre méthode est adapté aux problèmes de l’optimisation on arrive à avoir des solutions optimales proches de celles codées avec Matlab. Le seul souci c’est le pas et le point de début qu’on doit choisir avec rigueur pour ne pas avoir un grand nombre d’itération ni une solution non conforme ou non convergente.

# Courbes de performances sur le problème de Steiner :

|  |  |
| --- | --- |
| On a utilisé la fonction suivante pour avoir les graphes de performances qui présentent le nb d’évaluations en fonction d’epsilon et le nombre d’itérations en fonction d’epsilon. | On a utilisé le script suivant pour obtenir les graphes. |
| function [nbIter,nappel]=Performance(TolX)  % global xvil lvil lstein  global nb\_pas\_opt  global nappel  nappel=0;  X0=[0.8;0.8 ];  %[nb\_pas\_opt, X]=Methoptim (@Objectif,X0,'Nelder Matlab',eup);  %[nb\_pas\_opt, X]=Methoptim (@Objectif,X0,'Newton Matlab',eup);  [nb\_pas\_opt, X]=Methoptim (@Objectif,X0,'Pas Constant',TolX);  % [nb\_pas\_opt, X]=Methoptim (@Objectif,X0,'Rech Lin',eup);  nbIter=nb\_pas\_opt; | eup=10^(-3):0.0001:10^(-1);  nbIterVector=zeros(length(eup));  nbappelVector=zeros(length(eup));  for i=1:length(eup)  [nbIterVector(i),nbappelVector(i)]=Performance(eup(i));    end  figure()  subplot(1,2,1)  plot(eup,nbIterVector)  title('nombre d iterations en fonction de epsilon')  xlabel('epsilon de 10^(-3) à 10^(-1)')  ylabel('nombre d iterations')  subplot(1,2,2)  plot(eup,nbappelVector)  title('nombre d evaluations en fonction de epsilon')  xlabel('epsilon de 10^(-3) à 10^(-1)')  ylabel('nombre d evaluations') |

On a dû changer un peu la fonction methodesopti pour qu’elle puisse nous rendre le nombre d’itérations aussi.

Le graphe ci-dessous nous montre le nombre d’itérations nécessaires pour la résolution du problème en fonction du critère de convergence epsilon.

On constate que la courbe est décroissante ie le nombre d’itérations pour un rho donnée décroît lorsque que l’on augmente le critère de convergence. Ce qui est cohérent.

Pour la méthode de rech lin on a le même résultat presque, c’est toujours décroissant en fonction de epsilon : moins est la précision moins est le nombre d’évaluation et d’itérations.

# Programme Gradient à pas optimal :

Le script pour la méthode du gradient à pas optimal est le suivant :

x=x0;

val0=critere(x0);

ecart=1e10; % pour entrer dans la boucle while une première fois

nb\_pas\_opt=0;

TolX=1e-2;

elseif strcmp(methode,'Rech Lin') % gradient avec recherche linéaire

val=critere(x);

while ecart>TolX

Grad=Calgrad(critere,x);

Jx=critere(x);

x1=Armijo(critere, x,Jx,-Grad,Grad); % recherche linéaire assez grossière

%x1=Goldstein2(critere, x,-Grad)

ecart=norm(x1-x);

x=x1;

nb\_pas\_opt=nb\_pas\_opt+1;

end

xopt=x;

On fait la comparaison entre la méthode du gradient à pas optimal et celle à pas constant en utilisant la fonction Steiner. Le tableau suivant est un récapitulatif des résultats :

|  |  |  |
| --- | --- | --- |
|  | **Pas optimal** | **Pas constant** |
| **3 points** | Pour epsilon=10^(-2)    Longueur minimale obtenue : 1.732051  nombre d'appels à la fonction objectif : 89  coordonnées des points solution  X =  0.5000  0.2888 | Pour epsilon=10^(-2)    Longueur minimale obtenue : 1.734065  nombre d'appels à la fonction objectif : 25  coordonnées des points solution  X =  0.5002  0.3287 |
| **4 points** | Avec le même epsilon.  Longueur minimale obtenue : 2.732064  nombre d'appels à la fonction objectif : 182  coordonnées des points solution  X =  0.4969  0.2878  0.4968  0.7129 | Avec le même epsilon.  Longueur minimale obtenue : 2.733313  nombre d'appels à la fonction objectif : 31  coordonnées des points solution  X =  0.4801  0.2756  0.4807  0.7348 |

On remarque qu’avec la méthode du pas optimal on a beaucoup plus d’itérations qu’avec la méthode à point fixe mais reste à voir si le temps n’est pas meilleur

# Conclusion :

Pour conclure, on aura pu voir à travers ces différentes méthodes qu’on a des complexités différentes. Ainsi, le critère de convergence permet de réduire le nombre d’itérations pour résoudre le problème s’il on prend de la marge. De plus, choisir un pas rho adéquat permet de converger plus rapidement vers la solution s’il est bien choisi.

On a pu observer que pour certaines valeurs, le programme ne fonctionnait pas justement à cause de la valeur trop faible (ou trop élevée) du pas. De même on a pu constater que la convergence dépendait aussi du point initial, qui, s’il est mal choisi peut mener à un minimum local non voulu ou à un cycle sans fin. Le gradient à pas optimal ne fonctionnait pas toujours lorsque le gradient était trop faible, et donc il était préférable d’utiliser une autre méthode pour trouver le résultat.

Ainsi, le fait d’utiliser toutes ces méthodes nous a permis de nous rendre compte que dans certains cas il fallait choisir une méthode plutôt qu’une autre pour résoudre le problème, étant donné les différences remarquables qu’il peut y avoir dans la complexité et notamment dans le nombre d’itérations de chacune d’elles.