

# **Rapport TP d'optimisation**

**1 Séance 1 : optimisation continue et optimisation approchée**

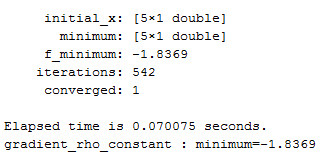
*L'executable de cette partie se trouve dans MAIN.m*

***1.1 Optimisation sans contraintes***

***1.1.1 Méthode du gradient***

a) La fonction GradResults permettant de minimiser une fonction par la méthode du gradient à pas fixe, nous la testons ici avec différentes valeurs de Rho. On obtient les résultats suivants :

* Rho = 0.01 : Minimum = -1.8369



* Rho = 0.1 : Pas de convergence
* Rho = 0.5 Pas de convergence

On s'apercoit donc que la convergence n'est pas toujours assurée en prenant une méthode du gradient à Rho constant.

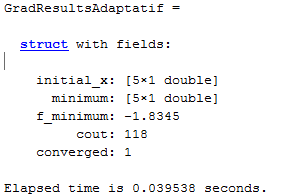
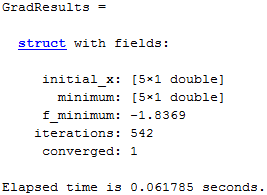
b) L'algorithme se trouve dans le fichier gradient\_rho\_adaptatif.m

c) Avec 'coût' représentant dans le cas adaptatif le nombre d'appel de la fonction, et en partant d'un pas initial Rho0 de 0.01, on a les résultats suivants :

**Algorithme Rho Constant = 0.01**

**Algorithme Rho Adaptatif : Rho0 = 0.01**

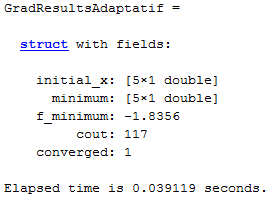
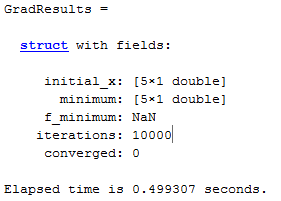




**Algorithme Rho Constant = 0.1**

**Algorithme Rho Adaptatif : Rho0 = 0.1**

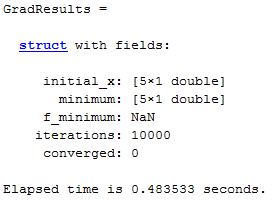
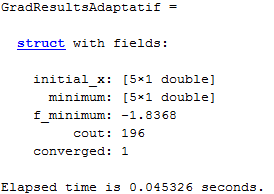
****

****

**Algorithme Rho Constant = 0.5**

**Algorithme Rho Adaptatif : Rho0 = 0.5**

****

** **

On remarque alors que la convergence est toujours assurée dans le cas de rho adaptatif, avec un temps de calcul plus faible..

**1.1.2 Utilisation des routines d’optimisation de la Toolbox Optimization. Mé- thode de Quasi-Newton (version BFGS).**

En utilisant la méthode de quasi Newton en laissant Matlab calculer un gradient approché (option par défaut), on obtient les résultats suivants:

|  |  |  |  |
| --- | --- | --- | --- |
| Méthode Rho constant  =0.01 | Méthode Rho Adaptatif  Rho0=0.01 | Quasi-Newton sans fournir de gradient | Quasi-Newton en fournissant le gradient |
| -1.8369 | -1.8345 | -1.8370 | -1.8370 |

|  |  |
| --- | --- |
| **Methodes de Quasi-Newton** | |
| Sans renseigner le gradient | En renseignant le gradient |
|  |  |

Les méthodes de quasi-Newton présentent moins d'itérations et d'appels à la fonction, mais présentent néanmoins un temps de calcul plus important, étant donné le calcul de l'inverse d'une matrice dans le processus d'optimisation.

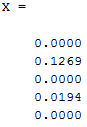
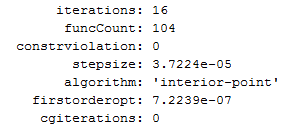
La fonction f1 est quadratique : il serait possible de calculer explicitement l'expression dans R5 et à partir de là se ramener à un problème de minimisation où on peut expliciter la solution par Fritz-Jones.

**1.2 Optimisation sous contraintes**

**1.2.1 Optimisation à l’aide de routines Matlab**

En utilisant l'algorithme SQP de Matlab, nous obtenons les résultats suivants:

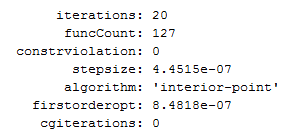
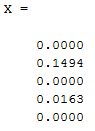
**F1**

****



**Valeur\_Min = -0.1385**

**F2**

****

Elapsed time is 4.294305 seconds.

**Valeur\_Min = 2 x 10^-6**

**1.2.2 Optimisation sous contraintes et pénalisation**

1) On prend une fonction C1 pour être sûr qu'elle soit semi continue inférieur.

Cette fonction de pénalisation est positive et telle que B(U)=0 si U appartient à Uad.

La fonction est implémentée dans le fichier Beta.m, et le script lançant l'optimisation de la fonction de pénalisation pour f1 et f2 se trouve dans MAIN2.m .

**1.2.3 Méthodes duales pour l’optimisation sous contraintes**

1) Le lagrangien de la fonction est défini dans le fichier lagrangien.m.

qui peut s'écrire sous forme matricielle par :

avec

2) L'algorithme d'Uzawa est implémenté dans le fichier MAIN2.m .

On trouve avec cette méthode

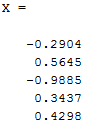
**F1min =-0.1381**

**1.3 Optimisation non convexe - Recuit simulé**

a) La fonction f4 est implémentée dans le fichier f4.m .

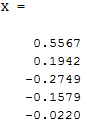
b) On utilise ici BFGC avec fminunc, en prenant différentes valeurs initiales.

* U0 = [0,0,0,0,0]



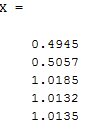
FVAL = -7.6563

* U0 = [0.5;0.5;0.5;0.5;0.5];



FVAL = -1.3731

* U0 = [0.5;0.5;1;1;1];



FVAL = 105.4410

c) Etant donné que plus T est faible, plus la probabilité p = exp(−(J(y)−J(xi))/T) de choisir un point "Moins bon" est faible, il convient de choisir un T petit. On fixe ici T = 50.

Par ailleurs, on diminue le nombre d'itérations à la même température afin de converger plus rapidement : On fixe L = 10.

Avec ces paramètres, nous verifions aisément que nous convergeons toujours vers le même point, à savoir le minimum global

**X = -1.5041; 0.4583; -0.6201; 0.9569; -0.8089**

**FVAL = -10.7979**

**1.4 Application Synthèse d’un filtre à réponse impulsionnelle finie**

1) En minimisant le critère J comme problème d'optimisation sans contrainte par la méthode de Quasi Newton, et en prenant comme H initial le vecteur 2\*ones(30,1), on obtient :

Jmin = 0.7669

Ce résultat dépend de la valeur initial de H: en prenant Hinitial = ones(30,1), on trouve Jmin = 1.

2) Le problème peut être reformulé de la manière suivante :

Pour

**2 Séances 2 et 3 : optimisation discrète et optimisation multi-objectif**

**2.1 Rangement d’objets (optimisation combinatoire)**

On considère ici le vecteur Xi,j= 1 si l'objet j est dans la boîte i.

1.

La condition pour que la boite i contient un objet et un seul s'écrit :

Par ailleurs, la condition pour que l'objet j ne soit que dans une seule boite s'écrit:

2. Le problème s'écrit ici comme un problème linéaire en nombre entier qui s'écrit de manière à minimiser la fonction de déplacement total:

constante définie du problème, notée .

Dans un premier temps, il convient de transformer la matrice X en vecteur de taille n². Celui ci ///s'écrit :

X =

n

n

n

n

n

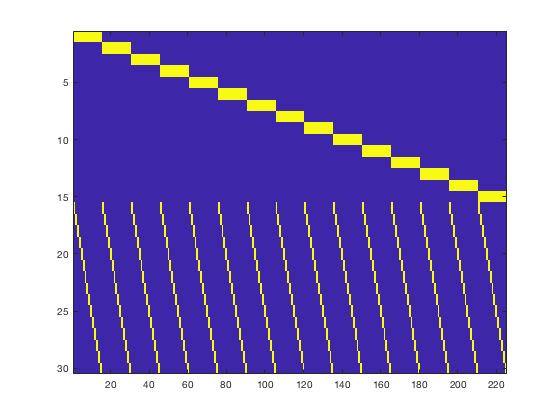
Les contraintes s'écrivent de la forme AX = B avec A qui s'écrit de la manière suivante dans le cas 3x3

n²&²

A=

2n

La matrice des contraintes d'égalité est donc de la forme:



Et B vecteur ne contenant que des 1.

En transformant la matrice ( ) i,j en vecteur, le problème devient :

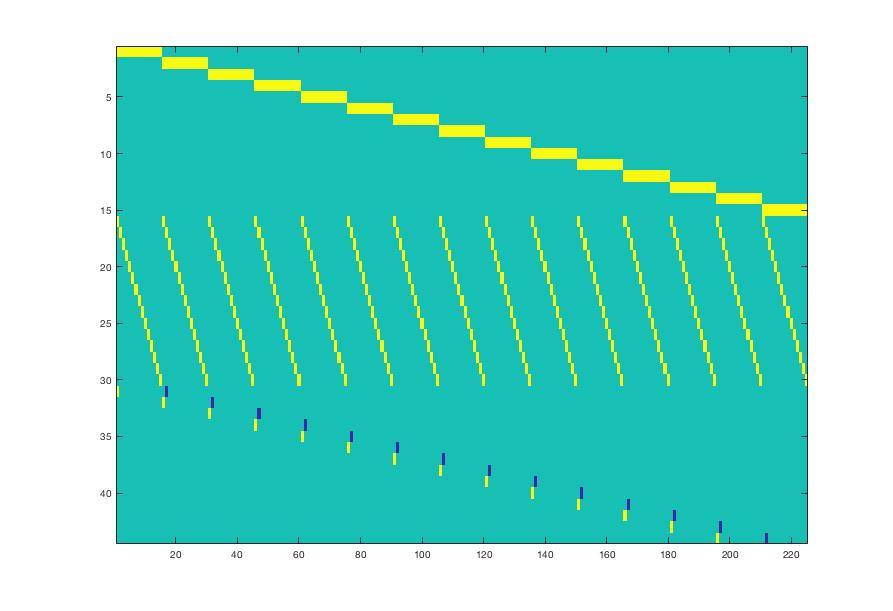
sous contrainte AX = B.

**15.3776**

3) La condition pour que l’objet n°1 se situe dans la boîte située juste à gauche de la boîte contenant l’objet n°2.

Ajouter cette contrainte revient à enrichir la matrice de contraintes afin de prendre en compte la condition que

On ajoute donc à la matrice construite précédemment la matrice suivante dans le cas n=3:



Le programme est implémenté dans le fichier xxx.m

On trouve Dmin =15.56

4) cela signifie que l'objet 3 se trouve dans une boite i et l'objet 4 ne se trouve pas dans une boite à droite de l'objet 3: l'objet 3 est donc à droite de l'objet 4.

cela signifie que l'objet 3 ne se trouve pas dans une boite à gauche de l'objet 4: l'objet 3 est donc à droite de l'objet 4.

Alors ni l'objet 3 ni l'objet 4 ne sont dans les boites i et (i+k): Si l'objet 3 n'est pas dans la boite i, alors l'objet 4 n'est pas a droite du 3.

Donc la condition traduisant la contrainte que la boîte contenant l’objet n°3 se situe à droite de la boîte contenant l’objet n°4 est bien xi,3 + xi+k,4 ≤ 1∀i, ∀k > 0.

Ainsi, la condition s'écrit :

5) Modélisation pour traduire la contrainte que la boîte contenant l’objet n°7 se situe à côté de la boîte contenant l’objet n°9:

ou

Celle ci s'écrit :

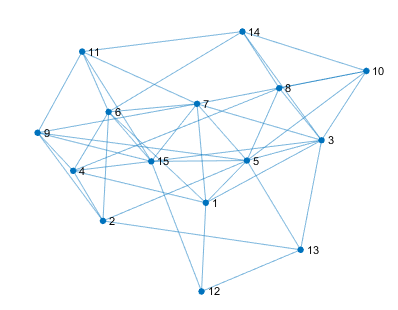
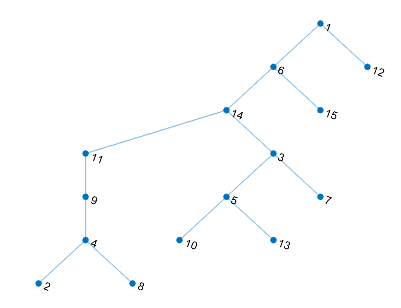
**2.2 Communication entre espions (optimisation combinatoire)**

On modélise le lien entre les espions par de taille n² tel que si les espions i et j communiquent, et 0 sinon.

Notons que .

La fonction a minimiser est donc .

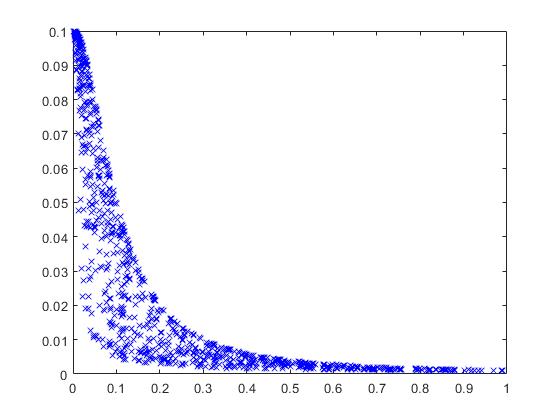
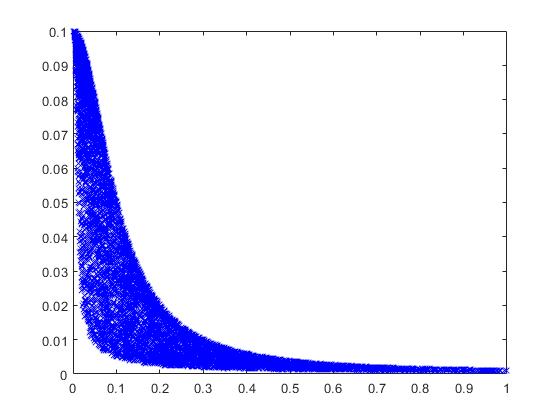
Afin d'avoir un problème linéaire, nous passons au logarithme: la fonction à minimiser devient :



En raisonnant par événement contraire, on obtient comme valeur minimale :

**2.3 Dimensionnement d’une poutre (optimisation multiobjectif)**

Il convient ici de générer N couples (a,b) de solutions réalisables, puis d'approximer le front de pareto par la courbe obtenue à partir des solutions de rang 1. Dès lors, une fois les solutions réalisables obtenues, on détermine le front de pareto par domination des solutions.

 **N=1000 N=10 000**