TP Optimisation - OMA

MEHDAOUI Abderrahim & ZEDDOUN Adnan

Octobre 2019
Compte rendu

Compte rendu

TP Optimisation - OMA

MEHDAOUI Abderrahim & ZEDDOUN Adnan

Octobre 2019

Table des matières

1 Optimisation sans contraintes

1.1 Méthode du gradient

Question 1 : Pour la méthode à ρ constant, on constate que pour des valeurs de ρ trop grandes ou trop petites, la convergence n'est plus assurée. Ainsi, il faut choisir une valeur de ρ convenable pour faire converger l'algorithme.

Question 2 : Pour permettre une meilleure convergence de l'algorithme, nous choisissons de programmer ce dernier avec un pas ρ adaptatif à chaque itération. Nous proposons 2 méthodes différentes : la méthode de relaxation et la méthode de Cauchy avec la section d'or.

Méthode de la relaxation
Initialement, on fixe ρ_0 et x_0. A l'étape n, si $f(x_n - \rho_n \nabla f(x_n)) < f(x_n)$, on prend $\rho_{n+1} = 2\rho_n$. Sinon, on prend $\rho_{n+1} = 0.5\rho_n$.

Méthode de Cauchy avec la section d'or
Initialement, on fixe ρ_0 et x_0. A l'étape n, on considère la fonction

$$h(\rho) = f(x_n - \rho \nabla f(x_n))$$

L'objectif est de minimiser h pour $\rho \in [a, b]$. Le problème de minimisation est cette fois à une dimension. Nous choisissons la méthode de la section d'or. Elle consiste à considérer des intervalles incluent dans $[a, b]$ de plus en plus petit et 4 points (2 étant les extrémités de l'intervalle et les 2 autres définis avec des ratios du nombre d'or). Par dichotomie, on converge vers le minimum global de h dans $[a, b]$.

Question 3 : Pour un ρ fixé, on converge avec beaucoup moins d'itérations via la méthode du gradient à pas adaptatif qu'avec la méthode du gradient à pas constant. En effet, on a par exemple plus de dix fois moins d'itérations pour $\rho = 10^{-3}$, 182 itérations en 0.00057s via la méthode de la relaxation contre 3753 itérations en 0.0047s via la méthode avec un pas constant.

La règle de Cauchy faisant appel à la section d'or est elle très efficace si l'on doit fixer un epsilon trop grand. Celle-ci converge vers un minimum moins précis (± 0.1 d'erreur) quand les deux autres méthodes diverge.

Enfin il est clair qu'utiliser la fonction **fminunc** fournie dans la toolbox optimization est bien plus efficace que toutes nos méthodes implémentées. En effet, il ne nous faut que 13 itérations pour converger vers un minimum via un algorithme implémenté utilisant une méthode de Quasi-Newton.

Pour f_2 on trouve que les méthode à pas constant est très peu performante. Par exemple, pour $\rho = 10^{-4}$, l'algorithme converge en 7226 itérations vers -0.1607 alors que pour la méthode de la relaxation il converge en 71 itérations. Pour la méthode de Cauchy avec la section d'or, il converge en 86 itérations. En utilisant **fminunc**, on converge en beaucoup moins d'itérations. On voit l'intérêt de choisir la bonne méthode pour ρ

1.2 Méthode de Quasi-Newton (version BFGS)

Question 1 : On peut trouver analytiquement le minimum en résolvant $\nabla f(U) = 0$ (car la fonction est convexe) ce qui nous donne $2SU - B = 0$ soit $U = \frac{1}{2}S^{-1}B$ et $f(U) = 1.8370$ on a donc une erreur de 10^{-3}.

2 Optimisation sous contraintes

2.1 Optimisation à l'aide de routines Matlab

Question 1 :

On utilise l'algorithme SQP et la fonction **fmincon** pour f_1 et f_2 et on obtient respectivement un minimum -0.1385 en 7 itérations et un minimum 0 en 9 itérations.

2.2 Optimisation sous contraintes et pénalisation

Question 1 : On choisit la fonction

$$\beta(u_1, ..., u_5) = \sum_{i=1}^{5}((u_i - 1)^+)^2 + ((-u_i)^+)^2$$

Question 2 : En prenant des valeurs très petites pour epsilon l'algorithme de pénalisation donne des résultats identiques à ceux trouver à la question 1 (algorithme SQP).

2.3 Méthodes duales pour l'optimisation sous contraintes

Question 1 : Le lagrangien associé à la fonction f_1 s'écrit

$$L_1(x, \lambda) = f_1(x) + \sum_{i=1}^{5} \lambda_i(x_i - 1) + \sum_{i=6}^{10} \lambda_i(-x_i)$$

où $\lambda_i \geq 0$ pour tout $i = 1, ..., 10$

Question 2 : On obtient bien résultat identique à celui trouvé en 1.2.1.

3

Optimisation non convexe - Recuit simulé

Question 1 : On invite le lecteur à consulter le fichier Matlab correspondant.

Question 2 : En prenant U_0 à 0; 1; 3.8 on obtient respectivement les minimums suivants : -7.66; 1.77 et 975.66. On constate bien qu'ils sont radicalement différents et qu'ils s'agit donc de minima locaux.

Question 3 : En prenant 100 valeurs U_0 et en appliquant la méthode du recuit simulé, on constate à l'aide du graphique ci-dessous que la répartition des coordonnées de l'argument minimum de f_3 sont constante à ± 1 près. On converge donc presque toujours vers le même point, quel que soit le point d'initialisation de l'algorithme.

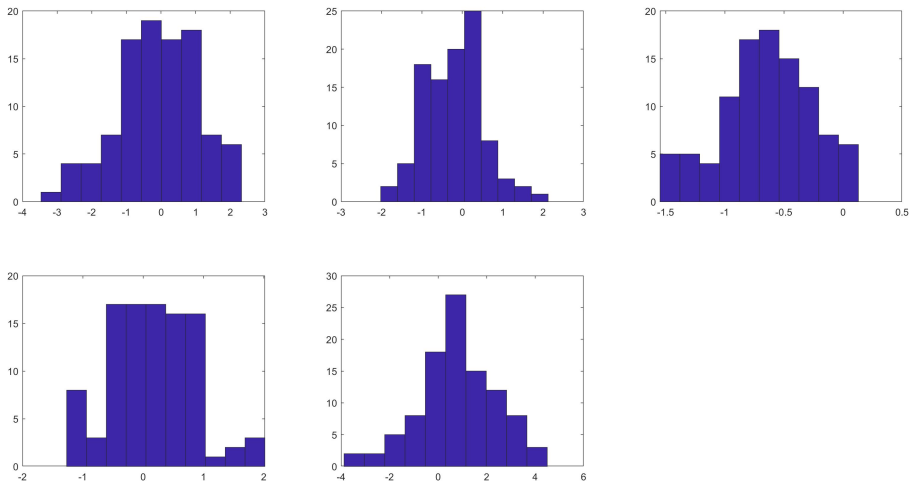

Fig. 3.1: Distribution des coordonnées du minimum global de f_3 trouvé à l'aide de la méthode du recuit simulé

4 Synthèse d'un filtre à réponse impulsionnelle finie

Question 1 : On résout ce problème d'optimisation avec la méthode quasi-Newton. On lance plusieurs fois l'algorithme pour s'assurer d'avoir un minima "le plus global".

On a alors le filtre avec les réponses impulsionnelle et fréquentielle suivantes :

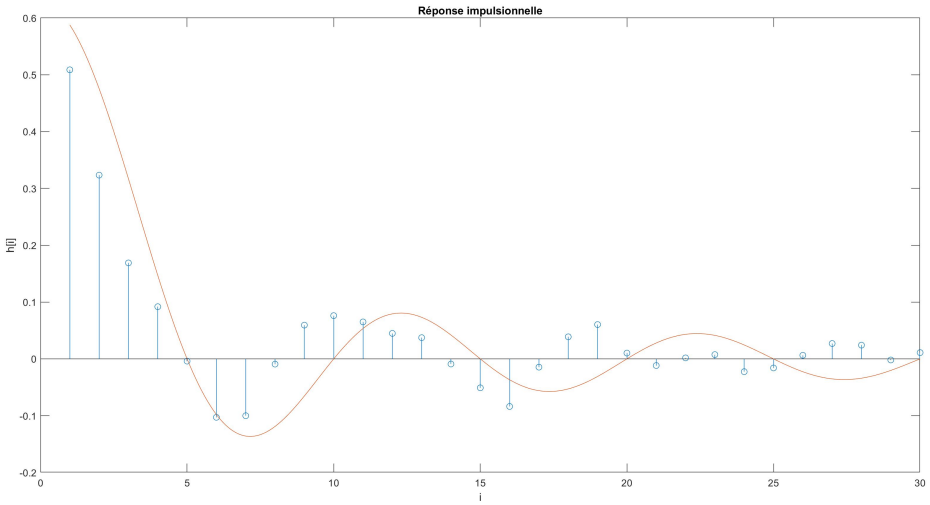

Fig. 4.1: Réponse impulsionnelle du filtre satisfaisant le problème d'optimisation

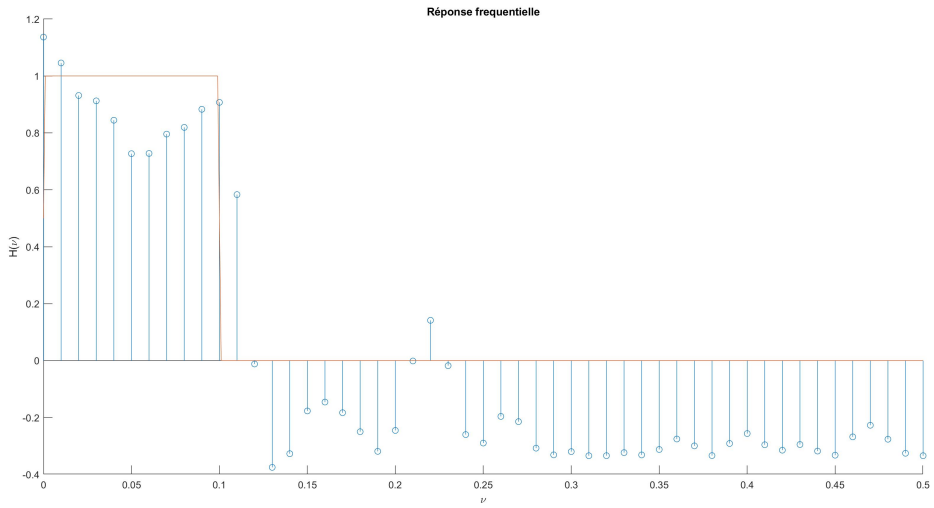

Fig. 4.2: Réponse fréquentielle du filtre satisfaisant le problème d'optimisation

5 Rangement d'objets

Question 1 :

On traduit que la boite i contient un objet et un seul par :

$$\sum_{j=1}^{N} x_{ij} = 1$$

Également, on traduit que l'objet j est dans un boite et une seule par :

$$\sum_{i=1}^{N} x_{ij} = 1$$

Question 2 :

On formule notre problème d'optimisation par un problème de programmation linéaire en nombre entiers :

$$\min_{x_{ij}} \sum_{i=1}^{N} \sum_{j=1}^{N} x_{ij} \|O_j - B_i\|$$

avec les contraintes :

- $\sum_{j=1}^{N} x_{ij} = 1$ pour tout i
- $\sum_{i=1}^{N} x_{ij} = 1$ pour tout j
- $x_{ij} \in \{0, 1\}$

On résout le problème de programmation linéaire en nombre entiers à l'aide de la fonction *intlinprog* de Matlab dans la bibliothèque *Optimization*.

On obtient la répartition de la figure ci dessous.

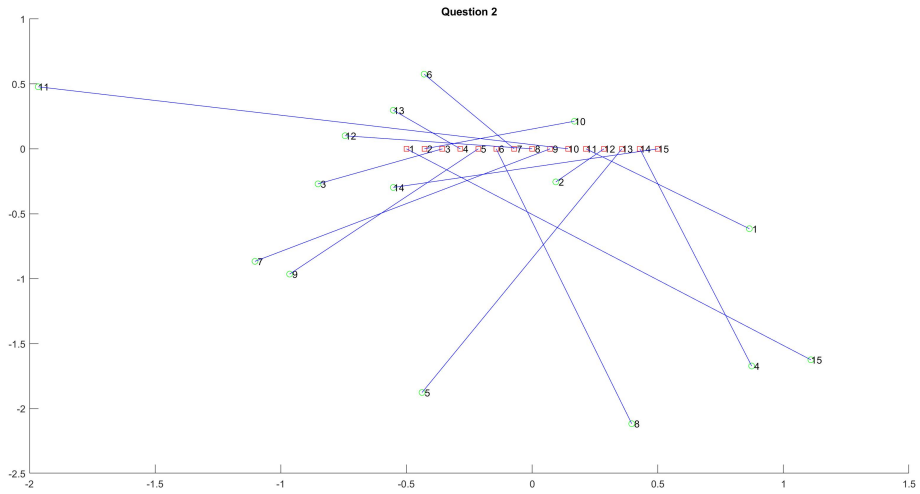

Fig. 5.1: Rangement des objets avec les contraintes définies à la question 2

La valeur de convergence donnée est **15.377628**.

Question 3 :

On ajoute une nouvelle contrainte : l'objet 1 doit se situer dans la boite située juste à gauche de la boite contenant l'objet 2. Ceci se traduit mathématiquement par les contraintes suivantes :

$$x_{i,1} = x_{i+1,2} \text{ pour tout } i \text{ allant de } 1 \text{ à } n-1$$

.

On obtient la répartition ci-dessous :

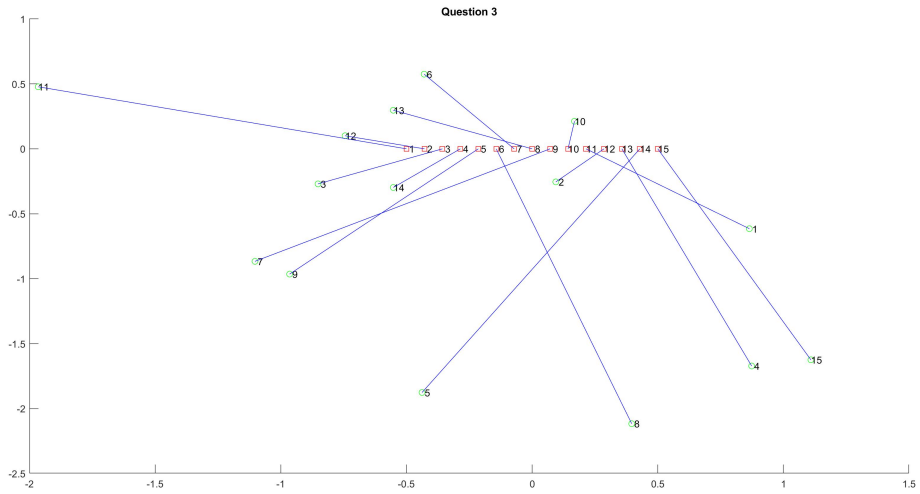

Fig. 5.2: Rangement des objets avec les contraintes définies à la question 2 et 3

La valeur de convergence donnée est **15.565123**.

Question 4 :

On ajoute une nouvelle contrainte : la boîte contenant l'objet n°3 se situe à droite de la boîte contenant l'objet n°4 (sans être forcément juste à côté). Ceci se traduit mathématiquement par les contraintes suivantes :

$$x_{i,3} + x_{i+k,4} \leq 1 \text{ pour tout } i \text{ allant de } 1 \text{ à } n \text{ et } k > 0$$

.

En effet,

- $x_{i,3} = 0$ et $x_{i+k,4} = 0$, les deux objets ne sont pas en i et $i + k$
- $x_{i,3} = 0$ et $x_{i+k,4} = 1$, l'objet 4 est dans la boite $i + k$ et l'objet 3 n'est dans aucune boite à gauche de l'objet 4
- $x_{i,3} = 1$ et $x_{i+k,4} = 0$, l'objet 3 est dans la boite i et l'objet 4 n'est dans aucune boite à droite de l'objet 3

On obtient la répartition ci-dessous :

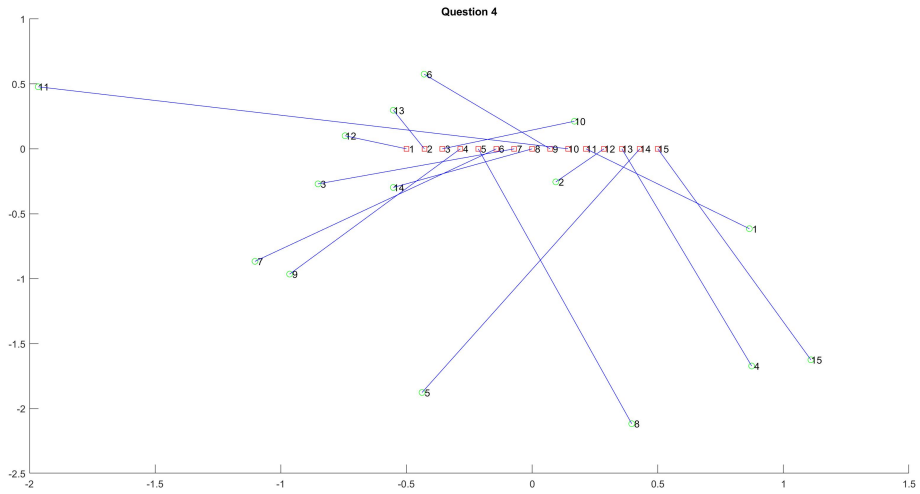

Fig. 5.3: Rangement des objets avec les contraintes définies à la question 2,3 et 4

La valeur de convergence donnée est **15.901379**.

Question 5 :

On ajoute une nouvelle contrainte : la boîte contenant l'objet n°7 se situe à côté de la boîte contenant l'objet n°9. Ceci se traduit mathématiquement par les contraintes suivantes :

- $x_{1,9} - x_{2,7} \leq 0$
- $x_{N,9} - x_{N-1,7} \leq 0$
- $x_{i,9} - x_{i-1,7} - x_{i+1,7} \leq 0$ pour tout i allant de 2 à $n-1$.

On obtient la répartition ci-dessous :

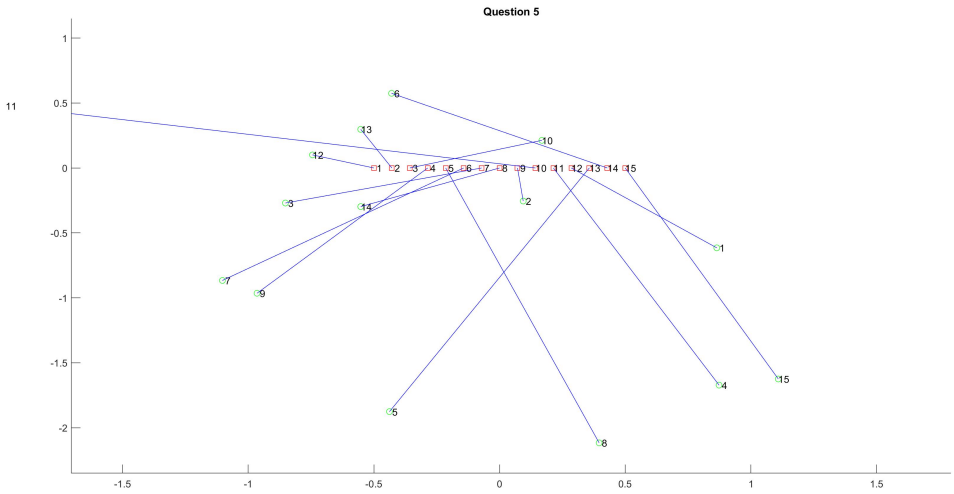

Fig. 5.4: Rangement des objets avec les contraintes définies à la question 2,3,4 et 5

La valeur de convergence donnée est **15.982158**.

6 Communication entre espions

Question 1 :

On peut modéliser le problème comme suit : considérons un graphe où les sommets représentent les espions et les arêtes correspondent aux liens entre les espions qui peuvent communiquer directement avec des poids p_{ij}. Dans notre exercice, le graphe est connexe c'est à dire qu'il existe toujours un chemin reliant deux espions. On suppose l'indépendance des interceptions entre 2 communications différentes. Ainsi, notre problème est un problème d'arbre recouvrant de poids minimal où il faut minimiser le critère suivant :

$$\prod_{i=1}^{n} \delta_{ij} p_{ij}$$

où δ_{ij} vaut 0 s'il n'a pas de lien entre les espions i et j, 1 sinon.

Nous mettons notre critère sous forme standard (avec une somme) :

$$\sum_{i=1}^{n} \delta_{ij} \log(p_{ij})$$

Ce problème se resout à l'aide des algorithmes de Kruskal ou de Prim (en temps polynomial avec n).

Question 2 :

La probabilité d'interception est **0.5809**.

```
(2,4)        0.0066
(3,5)        0.1156
(1,6)        0.0983
(3,7)        0.0981
(4,8)        0.0631
(4,9)        0.0768
(5,10)       0.0486
(9,11)       0.1074
(1,12)       0.0395
(5,13)       0.0520
(3,14)       0.0517
(6,14)       0.0098
(11,14)      0.0272
(6,15)       0.0400
```

Fig. 6.1: Chemin permettant la transmission d'un message entre espions minimisant la probabilité d'interception

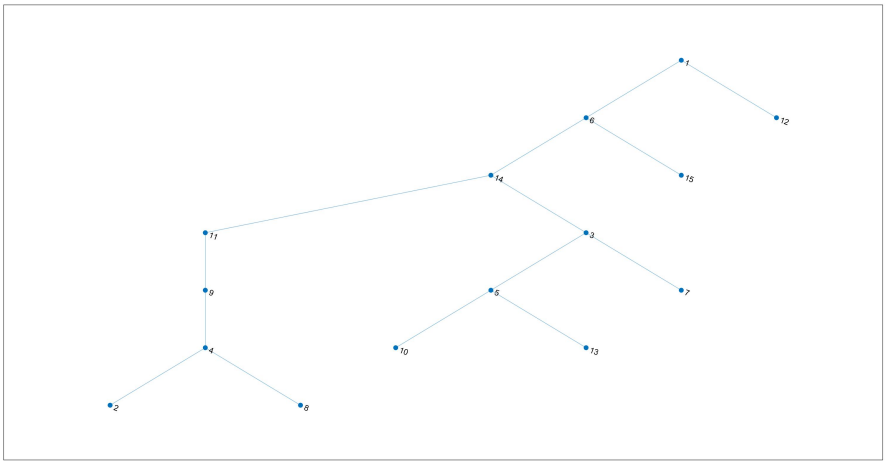

Fig. 6.2: Arbre représentant la transmission d'un message entre les espions qui minimise la probabilité d'interception

7 Dimensionnement d'une poutre

Méthode gloutonne

On génère N couples satisfaisant les contraintes et on approxime le front de Pareto avec les points non dominés.

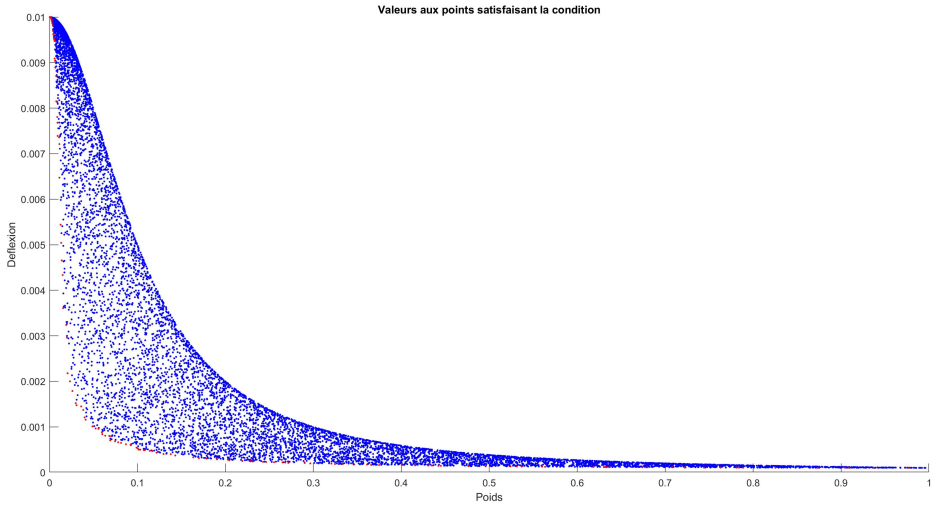

Fig. 7.1: Déformation en fonction du poids. En rouge, front de Pareto

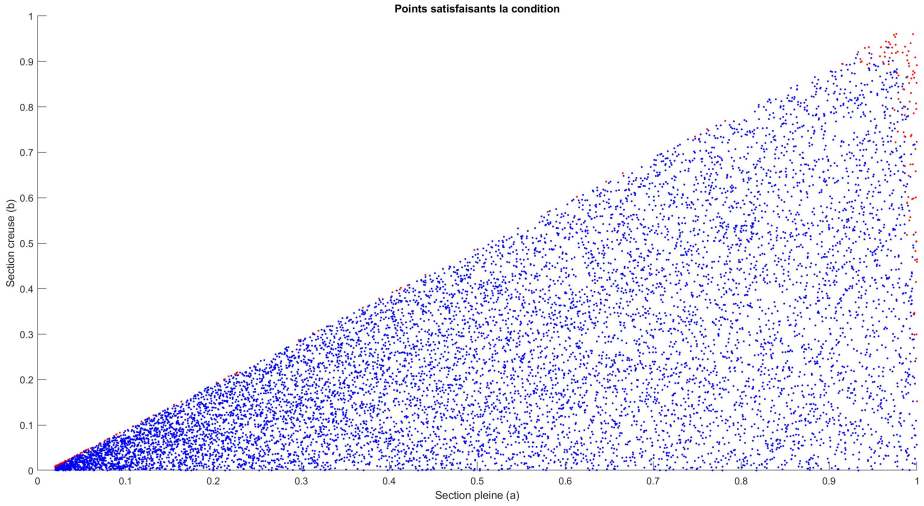

Fig. 7.2: Points satisfaisants les contraintes. En rouge, points satisfaisant les contraintes et non dominés

Méthode sophistiquée

L'idée est de se ramener à un problème d'optimisation mono-objectif. On considère

$$J(a,b) = d(a,b) + \alpha p(a,b)$$

où α est un paramètre variant de 0 à $+\infty$. Nous utilisons la fonction *fmincon* de Matlab pour résoudre ce problème d'optimisation. L'avantage de cette technique est de réduire la complexité de l'algorithme permettant d'avoir le front de Pareto de notre problème.

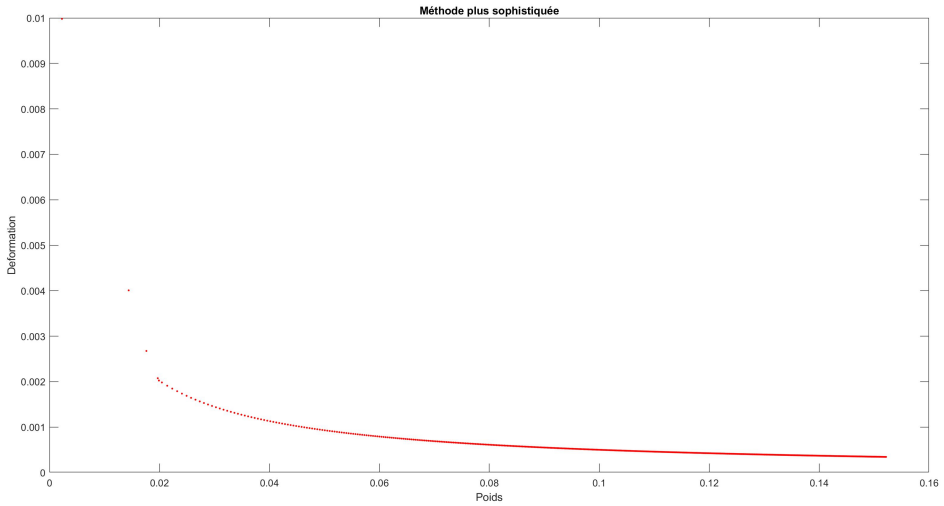

Fig. 7.3: Front de Pareto par optimisation mono-objectif

Méthode génétique

Pour obtenir un front de Pareto avec une méthode génétique, nous avons utiliser la fonction *gamultiobj*. Nous avons très rapidement un front de Pareto respectant les contraintes du problème.

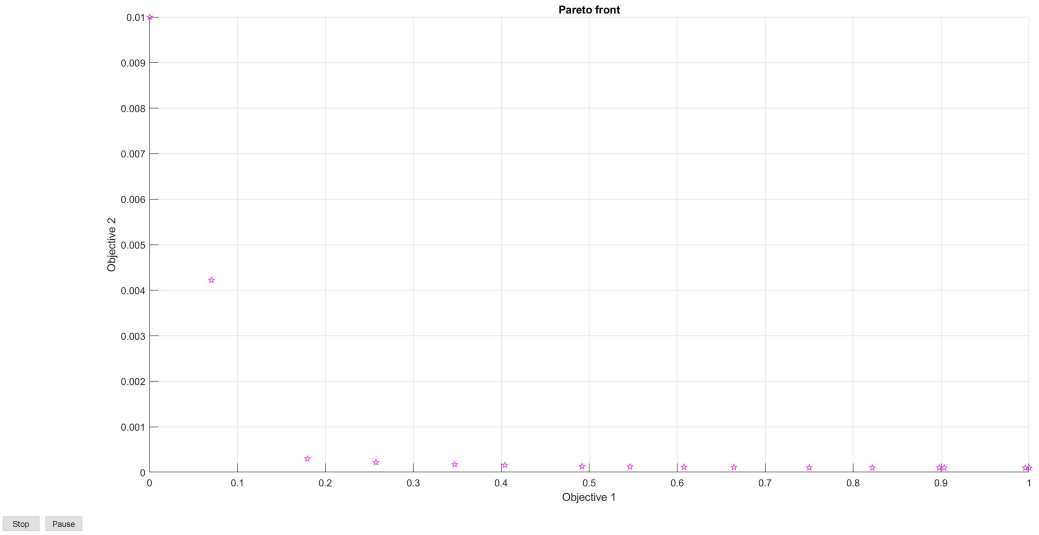

Fig. 7.4: Front de Pareto par méthode génétique

8 Approvisionnement d'un chantier

Une entreprise de construction a un chantier à accomplir sur N semaines. Elle établit semaine par semaine, ses besoins en machines pour ce chantier. L'entreprise fait appel à une société de location qui lui fournit les tarifs suivants :

- L'acheminement d'une machine sur le chantier coûte 800
- La location d'une machine coûte 200 par semaine
- La récupération d'une machine ainsi que sa remise en état coutent 1200

On souhaite optimiser la stratégie à adopter par l'entreprise de construction si elle souhaite minimiser ses dépenses. Pour résoudre ce problème, nous allons le modéliser sous la forme d'un graphe orienté. On identifie ce problème à un problème d'approvisionnement vu en cours, on peut donc modéliser ce problème par un graphe. Un noeud représente le nombre de machines que l'entreprise peut stocker sur une semaine donnée (elle peut stocker plus que nécessaire mais jamais moins) et un arc entre deux noeuds représenterait le passage de d_i machine à la semaine i à d_{i+1} machine à la semaine $i+1$. Le poids de l'arête entre ces deux noeuds est le coût d'entretien (location, mise en service, évacuation) des machines possédées à la semaine $i+1$. Une fois ce graphe implémenté, il nous suffira d'appliquer un algorithme (type Dijkstra) pour trouver le plus court chemin entre la semaine 1 et la semaine N.

On pose pour chaque semaine $i \in \{1, N\}$

- d_i le besoin en machine pour la semaine i
- $(n_{i,1}, n_{i,2})$ les indices extrêmes des noeuds à la semaine i (on veut stocker l'indice des noeuds pour sauvegarder le plus court chemin)
- $(d_{i,1}, d_{i,2})$ les valeurs extrêmes possibles du nombre d'engin que l'entreprise peut stocker à la semaine i

On trouve $(d_{i,1}, d_{i,2})$ de la manière suivante :

- Si $d_{i,1} > d_{i,2}$) le besoin en machine est supérieur au stock maximum de la semaine i et il faut donc nécessairement réapprovisionner jusqu'à d_{i+1} (pas plus car nous ne voulons pas de coûts supplémentaire). Donc $d_{i,1} = d_{i,2} = d_{i+1}$ et $n_{i+1,1} = n_{i+1,2} = n_{i,2} + 1$ (un noeud possible).

- Sinon le besoin est inférieur et c'est à ce moment-là qu'il est judicieux de se demander s'il ne vaut pas mieux garder quelques machines. Donc :

  - Soit on rend des engins pour avoir $d_{i+1,1} = d_{i+1}$ et $n_{i+1,1} = n_{i,2} + 1$

  - Soit on garde certaines machines (ou toutes) et on a $n_{i+1,2} = n_{i+1,1} + (d_{i+1,2} - d_{i+1,1} + 1)$

Le poids d'un arc entre deux semaines se déduit en connaissant $\bar{d_i} \in [d_{i,1}, d_{i,2}]$ et $\bar{d}_{i+1} \in [d_{i+1,1}, d_{i+1,2}]$ comme suit. $p_{i,i+1} = \bar{d}_{i+1} * loc + max(\bar{d}_{i+1} - \bar{d}_i, 0) * init + max(\bar{d}_i - \bar{d}_{i+1}, 0) * fin$

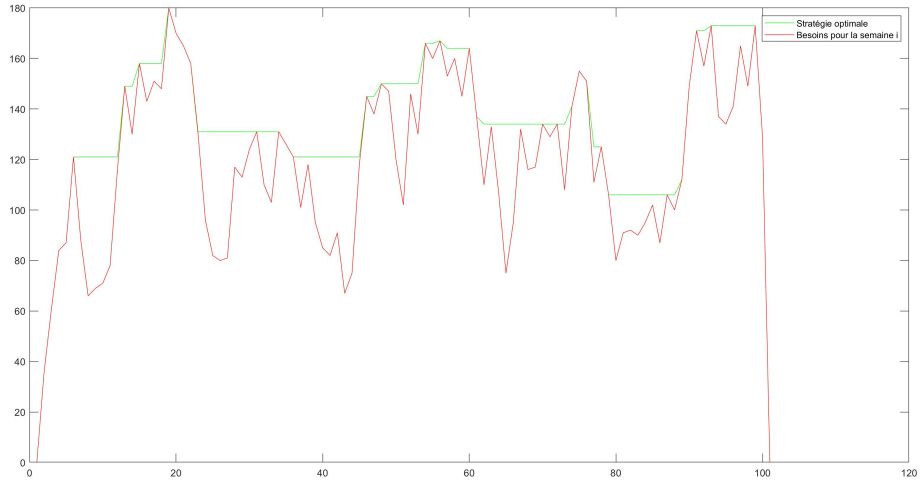

Fig. 8.1: Stratégie optimale (en vert) en fonction de la semaine

En implémentant ce graphe on trouve un coût minimale de **3 311 200**. Il est également possible de résoudre ce problème via d'autres méthodes comme la programmation linéaire en nombre entier via *intlinprog*.