TP2: La méthode des faisceaux

Pirashanth RATNAMOGAN , Othmane SAYEM

ratnamogan@ensta.fr, sayem@ensta.fr

1 Description de la méthode des plans sécants

On cherche à résoudre le problème :

$$\min_x f(x)$$

Lorsque la fonction f qu'on cherche à minimiser est convexe et sous-différentiable, on a vu en cours qu'on peut construire une approximation tangentielle de f à partir des valeurs de f et des sous-gradients sur un ensemble fini de points $(x_i)_{i\in\mathbb{N}}$. Cette approximation tangentielle sera donnée pour p donné :

$$\hat{f}_p(x) = \max_{i<p}(f(x_i) + s_i(x - x_i)) \qquad s_i \in \partial f(x_i) \qquad \forall i \in [[p]]$$

On a pu voir en cours que plus p -le nombre de points sur lesquels est évaluée la fonction f- est grand, plus on s'approche du minimum de f en minorant $\hat{f}_p$. On a également pu voir que le problème de minimisation de $\hat{f}_p$ se ramène à un problème classique linéaire sous contraintes que l'on peut résoudre facilement en utilisant l'algorithme du simplexe par exemple.

L'algorithme des plans sécants s'écrit de la manière suivante, étant donné η_0, x_0, s_0 et $k = 0$:

(i) Calcul de $f(x_k)$, $s_k \in \partial f(x_k)$
(ii) Résoudre

$$[P_k] \begin{cases} & \min_{x,\eta} \eta \\ s.c. & \eta \geq f(x_i) + < s_i, x - x_i > \quad \forall k \in [[p]] \\ & x, \eta \geqslant 0 \end{cases} \tag{1.1}$$

On récupère les solutions du problème (x_{k+1}, η_{k+1}).
(iii) Si $f(x_{k+1}) - \hat{f}_k(x_{k+1}) < \epsilon$ STOP
(iv) Sinon $k \leftarrow k + 1$ et retour à l'étape (i)

On peut énumérer quelques problèmes liés à cet algorithme : on a pas d'information sur la vitesse de convergence, l'algorithme peut être par conséquent très lent. De plus cet algorithme, qui peut durer pendant plusieurs milliers d'itérations, on accumule à chaque itération des coupes ce qui peut conduire à des problèmes de mémoire.

1.1 Convergence de l'algorithme

Au cours des itérations de l'algorithme on obtient une borne supérieure à notre problème de minimisation (égale à l'évaluation de notre fonction objectif au point optimal donné par la résolution du problème à chaque itération $[P_k]$). La valeur optimale de η obtenue dans $[P_k]$ fournit une borne inférieure du problème à résoudre puisque c'est une approximation affine sous contrainte du vrai problème que l'on cherche à résoudre.
Le critère de convergence de l'algorithme est donné par l'écart que l'on a entre notre borne supérieure du problème (qui constitue une solution admissible du problème) et la borne inférieure du problème.

2 Test de la méthode des plans sécants de base

2.1 Le problème à résoudre et les paramètres

Le problème que l'on cherche à résoudre est un problème que l'on peut créer artificiellement à partir du code fourni pour ce TP. On cherche à minimiser une fonction convexe et linéaire par morceaux. On construit des problèmes de tailles et de difficultés variées en jouant sur des paramètres de cette fonction. On peut notamment chosir **le nombre de variables** de la fonction mais également **le nombre de cassures** de la fonction que l'on crée. Ainsi un problème avec une variable et une cassure est un problème simple de recherche de l'intersection de droite que l'algorithme aura à traiter. Par contre lorsque l'on a plus de 100 variables et une centaine de cassures on peut comprendre qu'alors, le problème sera plus difficile à résoudre.

2.2 Nombre d'itérations nécessaires à la convergence

Une implémentation de l'algorithme des plans sécant sur un problème créé artificiellement nous a été donnée. On peut faire quelques tests qui nous permettront de vérifier les différents défauts de l'algorithme vus en cours.
Pour ce test on fixe le critère de convergence à $\epsilon = 1e-3$

Nombre de variables	Nombre de cassures	Nb itérations	Solution optimale
10	10	103	302.172689
10	60	148	1 961.365028
20	30	432	1 824.533954
20	50	428	3 080.238964
50	30	1 796	4 586.339368
50	100	2 432	15 267.636512
100	50	7612	15 205.953427

On voit à travers ce tableau que le critère qui augmente le plus le nombre d'itérations avant de converger est le nombre de variables qui augmente drastiquement le nombre d'itérations avant la convergence de notre algorithme. Le nombre de cassures est également un critère important qui joue sur la difficulté du problème mais ce paramètre a moins d'influence que le nombre de variables.

2.3 Analyse de la convergence

On peut analyser l'évolution de nos bornes inférieures et supérieures du problème sur quelques cas fixés (critère de convergence à $\epsilon = 1e-3$).

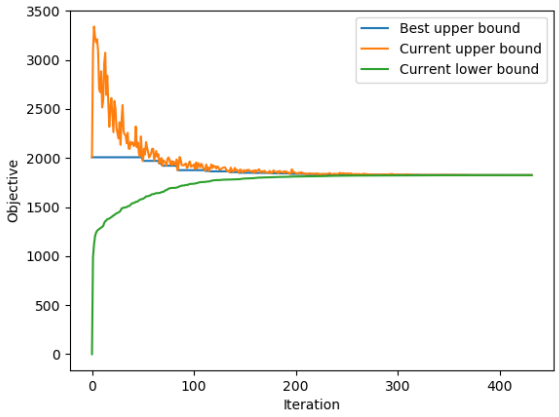


Figure 1: $N_{\text{cassure}} = 30, N_{\text{variable}} = 20$

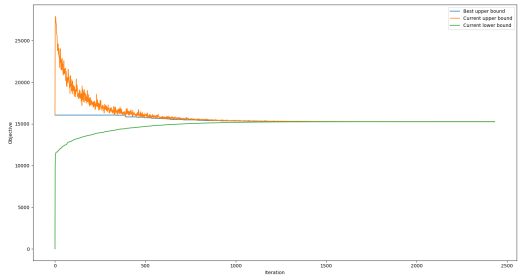


Figure 2: $N_{\text{cassure}} = 100, N_{\text{variable}} = 50$

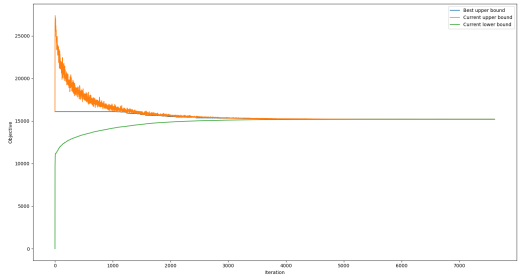


Figure 3: $N_{\text{cassure}} = 50, N_{\text{variable}} = 100$

Comme on peut le constater, la borne inférieure du problème est meilleure au fur et à mesure des itérations (on a prouvé en cours que cela était justifié théoriquement). On voit néanmoins que la borne supérieure obtenue est oscillante et on ne parvient à améliorer notre meilleure solution qu'après plus d'une centaine d'itérations (et cela s'empire lorsque notre nombre de variables augmente (plus de 2000 itérations pour améliorer notre solution admissible dans le cas avec 100 variables !)). On passe presque tout le début de l'algorithme en `Null Step`

3 L'algorithme des faisceaux

La régularisée de Moreau Yosida est donnée pour $c > 0$ par

$$f_c : x \to min_y \phi_{c,x}(y)$$

Où ϕ est donné par :

$$\phi_{c,x}(y) = f(y) + \frac{1}{2c}||x - y||^2$$

Dans le cours, on a pu prouver que f_c est une fonction différentiable même si f ne l'est pas. De plus on a vu que le minimum de f est atteind au même point que f_c et que le minimum de f est un point stationnaire de f_c. Utiliser une résolution de type gradient projeté conduirais à une résolution aussi complexe que celle de f. Ainsi on a introduit l'algorithme des faisceaux qui permet de résoudre ce problème :

L'algorithme des faisceaux s'écrit de la manière suivante, étant donné η_0, x_0, y_0 et $i = 0, k = 0$:

(i) Calcul de $f(y_i)$, $r_i \in \partial f(x_i)$

(ii) Résoudre

$$[P_k] \begin{cases} & \min_{y \in X, \eta} \eta + \frac{1}{2c}||x_k - y||^2 \\ s.c. & \eta \geq f(y_j) + < r_j, y - y_j > \quad \forall j \in [[i]] \\ & x, \eta \geqslant 0 \end{cases} \tag{3.1}$$

On récupère les solutions du problème (y_{i+1}, η_{i+1}).

(iii) a) Si $f(y_{i+1}) - \hat{f}_i(y_{i+1}) < \frac{1}{2c}||x_k - y_{i+1}||$. `Serious Step`

(i) Si $||y_{i+1} - x_k|| \leq \epsilon$ STOP

(ii) Sinon $x_{k+1} = y_{i+1}$, $k \leftarrow k + 1$et retour en (i)

(iv) b) Sinon `Null Step`

(i) Si $f(y_{i+1}) - \hat{f}_i(y_{i+1}) < \epsilon$ STOP

(ii) Sinon $i \leftarrow i + 1$

D'après le cours cet algorithme converge en un nombre fini d'itérations avec un test d'arrêt de même qualité que pour un algorithme de gradient. Cet algorithme permet d'introduire un centre de stabilié autour duquel on va chercher à minimiser la fonction au lieu d'essayer de minimiser la fonction sur tout l'espace. On notera $t = \frac{1}{2c}$, la tension (pénalisation) qui controle le fait de rechercher une solution localement ou non.

3.1 Test et résultats de la version de l'algorithme des faisceaux classique

Nous allons tester l'algorithme des faisceaux sur le même problème que celui qu'on a résolu précedemment grâce à l'algorithme des plans sécants. Nous écrirons donc le code en calquant le modèle donné par le fichier AMPL résolvant le problème des plans sécants. En un premier lieu, on considère que 't' sera constant tout au long de l'algorithme. Ainsi, la vitesse de notre algorithme va dépendre grandement de ce paramètre. Comme le montrent le tableau et la figure ci dessous :

Faisceaux - t fixe

	Iterations	Solution Optimale
10	667	1962.040388
1	124	1961.365263
0,1	27	1961.364957
0,01	62	1961.365029
0,0001	137	1961.365028
0	150	1961.365028

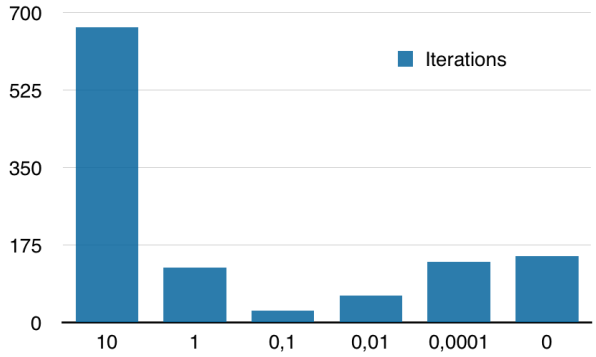
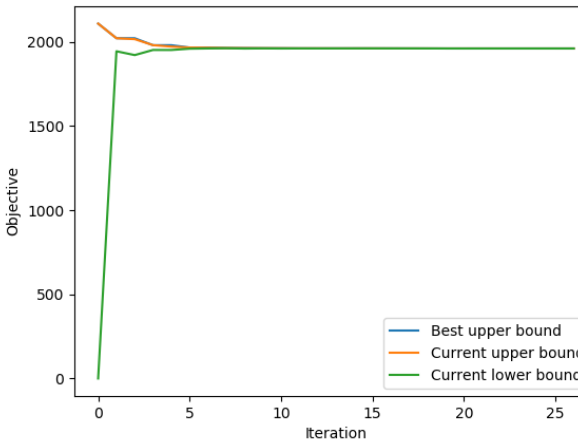
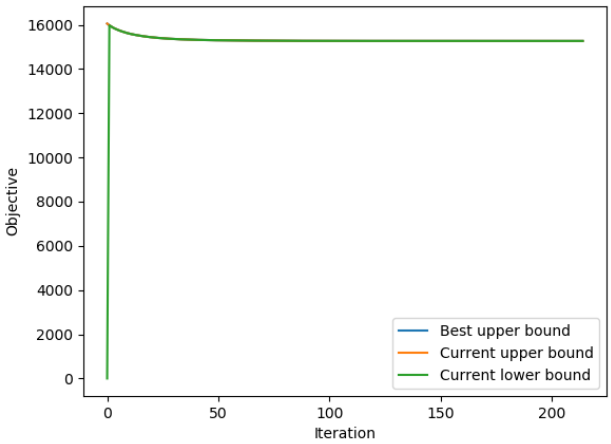


Figure 4: $N_{\text{cassure}} = 60, N_{\text{variable}} = 10$

En effet, pour tout choix de t, l'algorithme des faisceaux converge vers la même solution optimale. Toutefois, on remarque pour des grandes valeurs de t, l'algorithme mets beaucoup de temps à converger. Même remarque quand t tend vers 0. Ces résultats sont logiques puisque dans ce dernier cas, on revient à l'algorithme des plans sécants, qui est plus lent que l'algorithme des faisceaux. Dans le cas où t est grand, on donne trop d'importance à la recherche locale de la solution, ce qui engendre un temps de convergence important. Pour avoir un temps de convergence minimal, il faut choisir un bon compromis entre la recherche locale de la solution, et l'algorithme des plans sécants. Le choix optimal dans notre cas par exemple est t=0.1, qui converge très rapidement comparé aux autres valeurs. On trace donc l'évolution des bornes inf et sup de la fonction objective pour t=0.1, pour deux tailles différentes :



(a) $N_{\text{cassure}} = 60, N_{\text{variable}} = 10$



(b) $N_{\text{cassure}} = 50, N_{\text{variable}} = 100$

Figure 5: Algorithme des faisceaux, pour t = 0.1

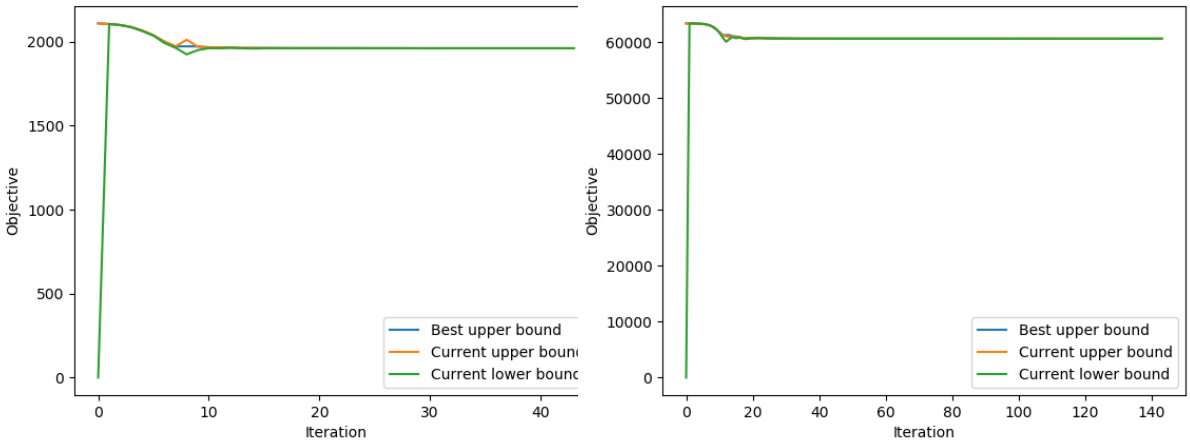
Afin d'éviter les cas où le choix initial de t modifie grandement la qualité de l'algorithme, une gestion dynamique s'impose afin d'obtenir la meilleure vitesse de convergence, même quand le choix initial de t n'est pas optimal.

3.2 Gérer dynamiquement la tension t

On observe que le paramètre t de convergence est déterminant dans la vitesse de convergence de l'algorithme et que ce paramètre est très dépendant de la taille du problème. On voudrait implémenter une version de l'algorithme qui gère dynamiquement ce paramètre en fonction des `Null Step` (pas d'amélioration de notre borne supérieure significative) et des `Serious Step` (amélioration de notre borne supérieure significative)
On va donc adapter notre paramètre selon l'évolution de notre borne supéreure, en augmentant ou en diminuant le paramètre t. En effet, dans le cas où notre borne supérieure s'améliore, on a choisit qu'on devrait relacher la contrainte de recherche locale un peu plus. Pour cette raison, dans la serious step, on diminue donc le terme de tension t. Dans le cas contraire, la Null Step, où on améliore pas la borne supérieure, on augmente le terme t afin de donner de l'importance à la recherche locale de solution. Après avoir essayé plusieurs formules pour l'incrémentation de t dans les deux steps, on a finalement opté pour l'incrémentation : $t = \frac{1}{2}t$ pour la Serious Step et $t = 2 \times t$ pour la Null Step.

3.3 Test et résultats de l'algorithme des faisceaux avec gestion dynamique de la tension

On applique la gestion dynamique décrite dans le chapitre précédant sur le même problème, pour des tailles différentes afin de mesurer l'impact et la rapidité que ça aura sur l'algorithme des faisceaux. On trace tout d'abord, l'évolution des bornes supérieures et inférieures, pour deux tailles différentes, et on observe que l'algorithme converge rapidement vers une valeur optimale.



(a) $N_{\text{cassure}} = 60, N_{\text{variable}} = 10$ **(b)** $N_{\text{cassure}} = 200, N_{\text{variable}} = 100$

Figure 6: Algorithme des faisceaux avec gestion dynamique

Pour bien mesurer l'importance de la gestion dynamique, on compare, comme pour la gestion fixe, le nombre d'itération nécéssaires quand on change le point de départ. Les résultats sont présentés sur le graphe en dessous, pour 50 variables et 100 cassures :

Faisceaux avec gestion dynamique

	Iterations	Solution Optimale
100	108	1961.365028
10	110	15267.787030
1	100	15267.775032
0,1	97	15267.752218
0,01	94	15267.766321
0,001	101	15267.766321

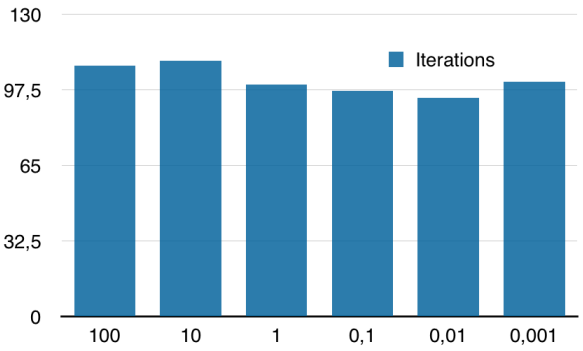


Figure 7: $N_{\text{cassure}} = 100, N_{\text{variable}} = 50$

On remarque, qu'en effet, le choix initial de la tension n'est pas très important et que le nombre d'itérations nécessaires pour la convergence est à peu près similaire pour tous les points initiaux.

Mis à part le fait de permettre à l'algorithme de converger pour tout t initial, cette gestion dynamique améliore de façon très significative la vitesse de l'algorithme. Pour mesurer cet impact, on calcule pour différentes tailles, les itérations nécessaires pour la convergence et on compare les trois algorithmes : plans sécants, faisceaux avec tension fixe (t=0.1) et faisceaux avec gestion dynamique de la tension :

Nombre de variables	Nombre de cassures	Algo	Nb itérations	Solution optimale
10	60	Plans sécants	148	1 961.365028
		Faisceaux (t=0.1)	27	1961.364957
		Faisceaux dynamiques	44	1961.383580
20	50	Plans sécants	428	3 080.238964
		Faisceaux (t=0.1)	82	3080.248085
		Faisceaux dynamiques	70	3080.255043
50	100	Plans sécants	2 432	15 267.636512
		Faisceaux (t=0.1)	215	15267.779812
		Faisceaux dynamiques	110	15267.787030
100	100	Plans sécants	>10 000	-
		Faisceaux (t=0.1)	557	30281.796866
		Faisceaux dynamiques	137	30280.104311

On remarque qu'effectivement, notre gestion dynamique de la tension t a permis d'améliorer la vitesse de convergence de l'algorithme des faisceaux de manière considérable, spécialement pour les problèmes de grande taille, où les deux autres algorithmes ont besoin de plusieurs itérations pour converger. On réduit significativement le nombre de `Null Step` grâce à l'algorithme des faisceaux.

4 Remplacer la contrainte quadratique de l'algorithme des faisceaux par une contrainte linéaire par morceaux à n branches

4.1 L'algorithme

On a pu voir précedemment que l'algorithme des faisceaux permet de largement accélerer l'algorithme des plans sécants classique. Lors de la résolution avec l'algorithme des faisceaux on a arbitrairement un terme de pénalité quadratique pour obliger l'algorithme à la recherche de solutions proches de la dernière solution qui aura été trouvée. Le choix de cette fonction était arbitraire, ce que l'on cherche en fait c'est une fonction qui va pénaliser l'écart de la nouvelle solution recherchée à l'ancienne solution X_h^*. On peut très simplement

imaginer par exemple remplacer la fonction quadratique par exemple une fonction très simple qui aurait cette forme :

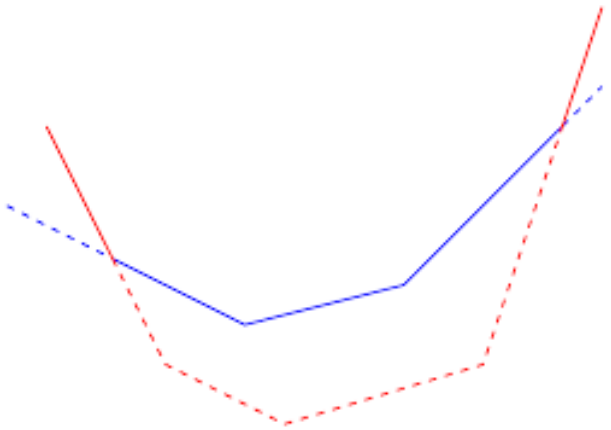


Figure 8: Exemple de fonctions linéaire par morceaux convexe

Il faut néanmoins réfléchir à la forme de la fonction linéaire par morceaux qui serait utile à la tâche qu'on veut réaliser. On voit par exemple que les fonctions de la figure 8 ne sont pas symétriques par rapport à leurs minimum ce qui n'est pas conforme au besoin que l'on a besoin pour notre pénalisation. On veut créer une fonction linéaire par morceaux assez proche de la fonction quadratique : plus on s'éloigne du zéro plus le coût de l'éloignement est élevé.
Ainsi, nous avons utilisé la fonction suivante pour remplacer la contrainte quadratique du faisceau :
Dans les explications qui suivent on se place en dimension 1 pour faciliter les explications, on généralise facilement à des dimensions plus élévés en considérant la fonction de pénalisation créé indépendemment suivant chaque variable.

On a découpé l'espace admissible X pour la solution x en n tranches de taille égale noté $(X_i)_{i\in[[1;n]]}$ avec $\cup_{i\in[[1;n]]}X_i = X \text{et } \forall i,j, i\neq j \quad X_i \cap X_j = \varnothing$. Notre fonction linéaire par morceaux agit sur un élément y (égale à l'écart en x et l'ancienne solution du Serious Stepp x_h^* dans notre cas). On a également n pentes $t_1, t_2, ..., t_n$ que l'on a défini comme étant proche de zéro lorsque l'on est à coté de l'origine (présent au centre de l'espace admissible) mais de plus en plus élevé lorsqu'on s'y éloigne ainsi on aura $t_1 > t_2 > t_3 > ... > t_{\frac{n}{2}} \simeq 0$ (partie négative), $t_n > t_{n-1} > t_{n-2} > ... > t_{\frac{n}{2}+1} \simeq 0$ (partie positive).
Avec les notations introduites notre nouvelle fonction de pénalisation linéaire par morceaux à n branches sera donnée pour chaque dimension de la variable considéré par

$$f(x) = \sum_{i\in[[1;n]]} \mathbb{1}_{x\in X_i}(x) * (t_i.x_i)$$

On a ainsi créé une fonction linéaire par morceaux qui pourrait parfaitement approximer la pénalisation quadratique pour un n très grand. La fonction créée est ainsi bien plus riche et très flexible. On peut par exemple décider de ne pas pénaliser le fait d'être proche de zéro en posant $t_{\frac{n}{2}+1} = t_{\frac{n}{2}} = 0$. On peut aussi fortement pénaliser d'être dans des segments éloignés de 0 en posant $t_1 = t_n = +\infty$. On pourrait également imaginer une gestion dynamique des pentes en fonction de l'endroit où l'on se trouve (qui joue le même rôle que la tension mais que l'on peut maintenant ajuster selon le segment sur lequel on se trouve).

Finalement, on peut donc implémenter l'algorithme de faisceaux par une contrainte linéaire par morceaux à n branches en modifiant dans 3.1 le terme quadratique par une somme sur chaque dimension de la variable x avec notre fonction linéaire par morceaux qui s'applique sur l'écart avec la dernière solution dans le `Serious Step`

4.2 Test de l'algorithme des faisceaux avec contrainte linéaire par morceaux à n branches

La encore comme dans la résolution sans gestion dynamique de la tension l'algorithme est assez dépendant des hyperparamètres à fixer. On fixe le nombre de branches à $n = 4$. Les pentes sont égales à l'écart au centre, pour les pentes des segments pas directement en contact avec 0 et 0.5 sinon. On fixe $eta = 0.5$, on obtient les résultats suivants :

Nombre de variables	Nombre de cassures	Algo	Nb itérations	Solution optimale
10	60	Plans sécants	148	1 961.365028
		Faisceaux	27	1961.364957
		Faisceaux dynamiques	44	1961.383580
		Faisceaux linéaire par morceaux	152	1961.366122
20	50	Plans sécants	428	3 080.238964
		Faisceaux	82	3080.248085
		Faisceaux dynamiques	70	3080.255043
		Faisceaux linéaire par morceaux	343	3081.791879

On voit sur ces exemples que les tests ne sont pas très concluants. Dans notre nouveau modèle, il y a beaucoup plus d'hyperparamètres. La bonne convergence de l'algorithme est très dépendant de ces hyperparamètres et il est difficile d'obtenir des bons résultats sur les tests.
Afin de rendre cette méthode performante il faudrait étudier comment plus précisement fixer les pentes et les cassures de notre modèle.

5 Conclusion

Dans le cadre de ce TP nous avons pu tester un des algorithmes que l'on a pu voir pendant le cours Optimisation non différentiable et méthodes proximales. On a pu constater que la méthode des plans sécants était adaptée pour résoudre des problèmes de minimisation de fonctions convexes et sous-différentiables. Néanmoins, le temps de convergence de l'algorithme peut-être extrêmement long, en particulier en grande dimension. L'algorithme des faisceaux permet d'obtenir les résultats de l'algorithme précédent avec un gain de temps assez important. Les performances de cet algorithme avaient le défaut de dépendre d'un paramètre de tension contrôlant l'importance du terme de pénalité. Une gestion dynamique de ce terme nous a permis d'obtenir des résultats concluants avec peu de supervision. Enfin on a essayé de proposer une variante de l'algorithme des faisceaux classique où le terme de pénalité est quadratique en utilisant une fonction linéaire par morceaux avec n branches. On parvient à obtenir des résultats corrects mais les résultats sont très dépendents des hyperparamètres : il faudrait trouver une manière de les ajuster automatiquement pour rendre cet algorithme bien plus performant.