OPT201/202
"Optimisation Différentiable : Théorie et Algorithmes"
Projet sur les réseaux de distribution d'eau

Khaoula BELAHSEN - Aicha BOUJANDAR

Février 2018

Table des matières

Présentation du problème

Dans notre projet, on cherche à décrire l'état d'équilibre d'un réseau de distribution d'eau potable. Ce réseau peut être modélisé par un graphe dont les noeuds sont soit des réservoirs, soit des demandes. Le but est donc de trouver les débits des arcs qui réalisent l'équilibre entre les soutitrages et les injections. Cela revient à trouver les débits qui minimisent l'énergie du réseau.

Variables du problème

Nom Math	Description physique de la variable
A	Matrice d'incidence noeuds-arcs
p	Pressions des noeuds
f	Flux des noeuds
q	Débits des arcs
r	Résistances des arcs

L'état d'équilibre se traduit par ailleurs par des équations de deux types.

1. La première loi de Kirchhoff qui exprime la non-accumulation d'eau aux noeuds du graphe. Elle a comme expression matricielle :

$$Aq - f = 0 \tag{1}$$

2. La seconde loi de Kirchhoff et La loi d'Ohm non linéaire qui exprime le fait que la perte de charge dérive d'un potentiel et dépend du débit de l'arc.

Le problème est modélisé par un graphe orienté à n arcs et m noeuds. Ce problème d'équilibre peut se voir comme un problème de minimisation de l'énergie du réseau, que l'on écrit :

$$min_{q\in\mathbb{R}^n; f_r\in\mathbb{R}^{m_r}} \frac{1}{3} < q, r \bullet q \bullet |q| > + < p_r, f_r > \tag{2}$$

Sous la contrainte $Aq - f = 0$

1 Résolution du problème primal

1.1 Mise sous forme d'un problème d'optimisation

Le problème d'équilibre décrit dans l'introduction est équivalent à un problème de minimisation sous contraintes. Notant $\Phi_\alpha(q_\alpha)$ la primitive de la fonction ϕ_α, ainsi :

$$\Phi_\alpha(q_\alpha) = \frac{1}{3} r_\alpha q_\alpha^2 |q_\alpha| \tag{3}$$

On définit la fonction $\bar{J}$ (qui peut s'interpréter comme l'énergie du réseau) par :

$$\bar{J}(q, f_r) = \sum_{\alpha \in A} \Phi_\alpha(q_\alpha) + \sum_{i \in N_r} p_i f_i \tag{4}$$

Notant $<.,.>$ le produit scalaire usuel (dans $\mathbb{R}^n$ ou $\mathbb{R}^{m_r}$), cette fonction se met sous la forme compacte :

$$\bar{J}(q, f_r) = \frac{1}{3} < q, r \bullet q \bullet |q| > + < p_r, f_r > \tag{5}$$

La détermination de l'équilibre du réseau revient alors à minimiser l'énergie du réseau tout en respectant la première loi de Kirchhoff, ce qui revient à résoudre le problème d'optimisation suivant :

$$min_{q \in \mathbb{R}^n ; f_r \in \mathbb{R}^{m_r}} \frac{1}{3} < q, r \bullet q \bullet |q| > + < p_r, f_r > \tag{6}$$

$$\text{Sous la contrainte } Aq - f = 0$$

En écrivant A sous la forme $A = \begin{pmatrix} A_r \\ A_d \end{pmatrix}$, la contrainte devient :

$$\begin{pmatrix} A_r q \\ A_d q \end{pmatrix} = \begin{pmatrix} f_r \\ f_d \end{pmatrix}$$

Ainsi, En remplaçant f_r par $A_r q$, le problème se met sous la forme :

$$min_{q \in \mathbb{R}^n} \frac{1}{3} < q, r \bullet q \bullet |q| > + < p_r, A_r q > \tag{7}$$

$$\text{Sous la contrainte } A_d q - f_d = 0$$

Réciproquement, si on cherche à résoudre :

$$min_{q \in \mathbb{R}^n} \frac{1}{3} < q, r \bullet q \bullet |q| > + < p_r, A_r q > \tag{8}$$

$$\text{Sous la contrainte } A_d q - f_d = 0$$

On pose $A_r q = f_r$ et $f = \begin{pmatrix} f_r \\ f_d \end{pmatrix}$

Le problème se réécrit ainsi sous la forme :

$$min_{q\in\mathbb{R}^n; f_r\in\mathbb{R}^{m_r}} \frac{1}{3} < q, r \bullet q \bullet |q| > + < p_r, f_r > \tag{9}$$

$$\text{Sous la contrainte } Aq - f = 0$$

D'où l'équivalence des deux problèmes.

Maintenant, à partir de la relation (5) qui dans notre problème est notée (14), on va démontrer qu'il s'écrit également sous la forme :

$$min_{q_c\in\mathbb{R}^{n-m_d}} \frac{1}{3} < q^{(0)} + Bq_c, r \bullet (q^{(0)} + Bq_c) \bullet |q^{(0)} + Bq_c| > + < p_r, A_r(q^{(0)} + Bq_c) > \tag{10}$$

Où :

$$q = q^{(0)} + Bq_c \text{ avec } q^{(0)} = \begin{pmatrix} A_{d,T}^{-1} f_d \\ 0_{n-m_d} \end{pmatrix} \text{ et } B = \begin{pmatrix} -A_{d,T}^{-1} A_{d,C} \\ I_{n-m_d} \end{pmatrix}$$

qui dans notre problème est noté (19).

En effet, en partant de la contrainte de l'équation (5), et en notant $A_d = \begin{pmatrix} A_{d,T} & A_{d,C} \end{pmatrix}$ et $q = \begin{pmatrix} q_T \\ q_C \end{pmatrix}$, on obtient : $q = \begin{pmatrix} A_{d,T}^{-1}(f_d - A_{d,C} q_C) \\ q_C \end{pmatrix} = \begin{pmatrix} A_{d,T}^{-1} f_d - A_{d,T}^{-1} A_{d,C} q_C \\ 0_{n-m_d} + I_{n-m_d} q_C \end{pmatrix} = \begin{pmatrix} A_{d,T}^{-1} f_d \\ 0_{n-m_d} \end{pmatrix} + \begin{pmatrix} -A_{d,T}^{-1} A_{d,C} \\ I_{n-m_d} \end{pmatrix} q_C$
D'où, en prenant :

$$q = q^{(0)} + Bq_c \text{ avec } q^{(0)} = \begin{pmatrix} A_{d,T}^{-1} f_d \\ 0_{n-m_d} \end{pmatrix} \text{ et } B = \begin{pmatrix} -A_{d,T}^{-1} A_{d,C} \\ I_{n-m_d} \end{pmatrix}$$

On a bien l'écriture (7).

Réciproquement, en prenant les valeurs de q, B et $q^{(0)}$ on obtient bien la contrainte de l'équation (5), d'où l'équivalence entre les problèmes (14) et (19) est bien établie.

1.2 Calculs analytiques

On note pour $q_c \in \mathbb{R}^{n-m_d}$:

$$\begin{aligned} J(q_c) &= \frac{1}{3} < q^{(0)} + Bq_c, r \bullet (q^{(0)} + Bq_c) \bullet |q^{(0)} + Bq_c| > + < p_r, A_r(q^{(0)} + Bq_c) > \\ &= \frac{1}{3} < q, r \bullet q \bullet |q| > + < p_r, A_r q > \end{aligned}$$

Avec la notation : $q = q^{(0)} + Bq_c$

1.2.1 Calcul du gradient de J :

Soient $q_c, h \in \mathbb{R}^{n-m_d}$, on a :

$$\begin{aligned} J(q_c+h) &= \frac{1}{3} < q+Bh, r \bullet q + Bh \bullet |q+Bh| > + < p_r, A_r q + A_r Bh > \\ &= \frac{1}{3}\Sigma_{i=1}^{n-m_d} signe((q+Bh)_i)(q+Bh)_i r_i (q+Bh)_i (q+Bh)_i + < p_r, A_r q + A_r Bh > \end{aligned}$$

Or, pour h assez petit : $signe((q+Bh)_i) \simeq signe(q_i)$, donc :

$$\begin{aligned} J(q_c+h) &= \frac{1}{3}\Sigma_{i=1}^{n-m_d} signe(q_i) r_i [q_i^3 + 3q_i^2 (Bh)_i + 3q_i (Bh)_i^2 + (Bh)_i^3] + < p_r, A_r q > + < p_r, A_r Bh > \\ &= J(q_c) + \Sigma_{i=1}^{n-m_d} signe(q_i) r_i q_i^2 (Bh)_i + < B^T A_r^T p_r, h > + o(||h||) \\ &= J(q_c) + < B^T r \bullet q \bullet |q|, h > + < B^T A_r^T p_r, h > + o(||h||) \end{aligned}$$

D'où :

$$\nabla J(q_c) = B^T (r \bullet q \bullet |q| + A_r^T p_r) \tag{11}$$

1.3 Calcul du hessien

On a :

$$\nabla J(q_c+h) = B^T (r \bullet q + Bh \bullet |q+Bh| + A_r^T p_r)$$

Pour $j \in 1, ..., n-m_d$

$$\nabla J(q_c+h)_j = \Sigma_{i=1}^{n-m_d} B_{ji}^T signe((q+Bh)_i) r_i (q+bh)_i^2 + (B^T A_r^T p_r)_j$$

Or, pour h assez petit : $signe((q+Bh)_i) \simeq signe(q_i)$, donc :

$$\begin{aligned} \nabla J(q_c+h)_j &= \Sigma_{i=1}^{n-m_d} B_{ji}^T signe(q_i) r_i [q_i^2 + 2q_i (Bh)_i + (Bh)_i^2] + (B^T A_r^T p_r)_j \\ &= \nabla J(q_c)_j + \Sigma_{i=1}^{n-m_d} B_{ji}^T signe(q_i) 2 r_i q_i (Bh)_i + o(||h||) \\ &= \nabla J(q_c)_j + \Sigma_{i=1}^{n-m_d} \Sigma_{k=1}^{n-m_d} B_{ji}^T signe(q_i) 2 r_i q_i B_{ik} h_k + o(||h||) \\ &= \nabla J(q_c)_j + \Sigma_{i=1}^{n-m_d} \Sigma_{k=1}^{n-m_d} B_{ji}^T [diag(2 \bullet r \bullet |q|) B]_{ik} h_k + o(||h||) \\ &= \nabla J(q_c)_j + \Sigma_{k=1}^{n-m_d} [B^T [diag(2 \bullet r \bullet |q|) B]]_{jk} h_k \\ &= \nabla J(q_c)_j + [B^T diag(2 \bullet r \bullet |q|) Bh]_j \end{aligned}$$

Ainsi :

$$\nabla^2 J(q_c) = B^T diag(2 \bullet r \bullet |q|) B \tag{12}$$

1.4 Existence et unicité des solutions

Dans ce qui suit, on s'intéresse à l'existence et unicité des solutions du problème (19).

1.4.1 Existence d'une solution

On a :

— $\mathbb{R}^{n-m_d}$ est **fermé**.

— J est continue car composition de sommes et produits de composantes et valeur absolue, elle est donc **semi-continue inférieurement (s.c.i)**.

— J est **coercive**, en effet :
Si $||q_C||_\infty \to +\infty$ on a par équivalence des normes en dimension finie : $||J(q_C)||_2 \underset{||q_C||_2 \to +\infty}{\longrightarrow} +\infty$

Donc (19) admet au moins une solution.

1.4.2 Unicité de la solution

— L'ensemble $\mathbb{R}^{n-m_d}$ est convexe.

D'après les calculs précédents :

$$\nabla J(q) = r \bullet q \bullet |q| + A_r^T p_r$$

$$\nabla^2 J(q) = diag(2r \bullet |q|)$$

— J est strictement convexe en effet :

Soit $q, h \in \mathbb{R}$ avec h non nul.
On a

$$\nabla^2 J(q).h^2 = \sum_i 2r_i|q_i|h_i^2 > 0$$

Donc J est strictement convexe, donc la solution est unique.

Comme les problèmes (14) et (19) sont équivalents, l'existence d'une unique solution est aussi vérifiée pour (14).

1.5 Implémentation en Matlab

Le code Matlab est composé de deux oracles, OraclePH et OraclePG qui définissent la fonction à minimiser J qui est appelée F avec son gradient et son hessien. La fonction Main.m lance les calculs.

1.5.1 Recherche du minimum : démarche générale

Nous implémentons différents algorithmes de recherche de minimum. Ces algorithmes sont tous à direction de descente, ils construisent une suite (x_k) dans laquelle chaque itéré x_k est calculé par la relation $x_{k+1} = x_k + \alpha_k d_k$ Le calcul des α_k se fait par recherche linéaire. Ici tous les algorithmes d'optimisation utilisent la recherche linéaire de Wolfe.

1.5.2 Recherche linéaire de Wolfe

La recherche linéaire de Wolfe est l'une des méthodes possibles pour déterminer le pas α_k utilisé dans les algorithmes à direction de descente. On utilise cette méthode pour déterminer un pas α_k qui vérifie deux conditions :

La première est :

$$f(x_k + \alpha_k d_k) \leq f(x_k) + \omega_1 \alpha_k g_k^T d_k$$

avec f la fonction à minimiser, x_k le point courant, d_k la direction de descente, g_k le gradient en le point courant et ω_1 un paramètre qu'on choisit entre 0 et 1. La deuxième condition est ;

$$(\nabla f(x_k + \alpha_k d_k))^T d_k \geq \omega_2 g_k^T d_k$$

Les ω_1 et ω_2 sont choisis de manière arbitraire.

Pour la mise en oeuvre de l'algorithme de Flechter-Lemaréchal, on choisit deux bornes de α : $\underline{\alpha} = 0$ et $\bar{\alpha} = +\infty$, et on procède comme suit :

- si α_k ne vérifie pas la première condition : on diminue la borne supérieure $\bar{\alpha} = \alpha_k$ et on met à jour le pas : $\alpha_k = \frac{1}{2}(\bar{\alpha} + \underline{\alpha})$
- sinon :
  - si α_k ne vérifie pas la deuxième condition :
    - on augmente la borne inférieure de α : $\underline{\alpha} = alpha_k$
    - on met à jour le pas :
      - $\alpha_k = 2\underline{\alpha}$ si $\bar{\alpha} = +\infty$
      - $\alpha_k = \frac{1}{2}(\bar{\alpha} + \underline{\alpha})$ sinon
  - sinon le pas α_k vérifie les conditions de Wolfe.

1.5.3 Méthode du gradient à pas fixe

Dans cette méthode, la direction de descente d_k est l'opposé du gradient : $d_k = -\nabla f(x_k)$ Cette direction est bien une direction de descente car $\nabla f(x_k).d_k = -\|\nabla f(x_k)\|^2 \leq 0$ Et le pas est fixe, choisi par l'utilisateur au début de l'algorithme.

$$x_{k+1} = x_k - \alpha \nabla f(x_k)$$

En appliquant cette méthode sur la fonction coût du réseau de distribution d'eau, on obtient :

1.5.4 Méthode du gradient à pas variable

La direction de descente d_k est encore une fois l'opposé du gradient : $d_k = -\nabla f(x_k)$ Mais cette fois le pas varie et il est calculé avec la recherche linéaire de Wolfe.

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k)$$

Pour mettre en oeuvre cette méthode, on a utilisé l'algorithme de Flechter-Lemaréchal. On a ensuite implémenté la fonction $Gradient\_V$ qui représente l'algorithme d'optimisation. Cette fonction cherche la valeur optimale de la fonction en testant à chaque itération k si la valeur du gradient est proche de 0. Si ce n'est pas le cas, elle met à jour la valeur de la direction de descente $d_k = -gradient(x_k)$, elle calcule le pas α_k avec la méthode de Wolfe et elle met à jour la valeur du point courant en utilisant la formule $x_{k+1} = x_k + \alpha_k * d_k$.

En appliquant cette méthode sur la fonction coût du réseau de distribution d'eau, on obtient :

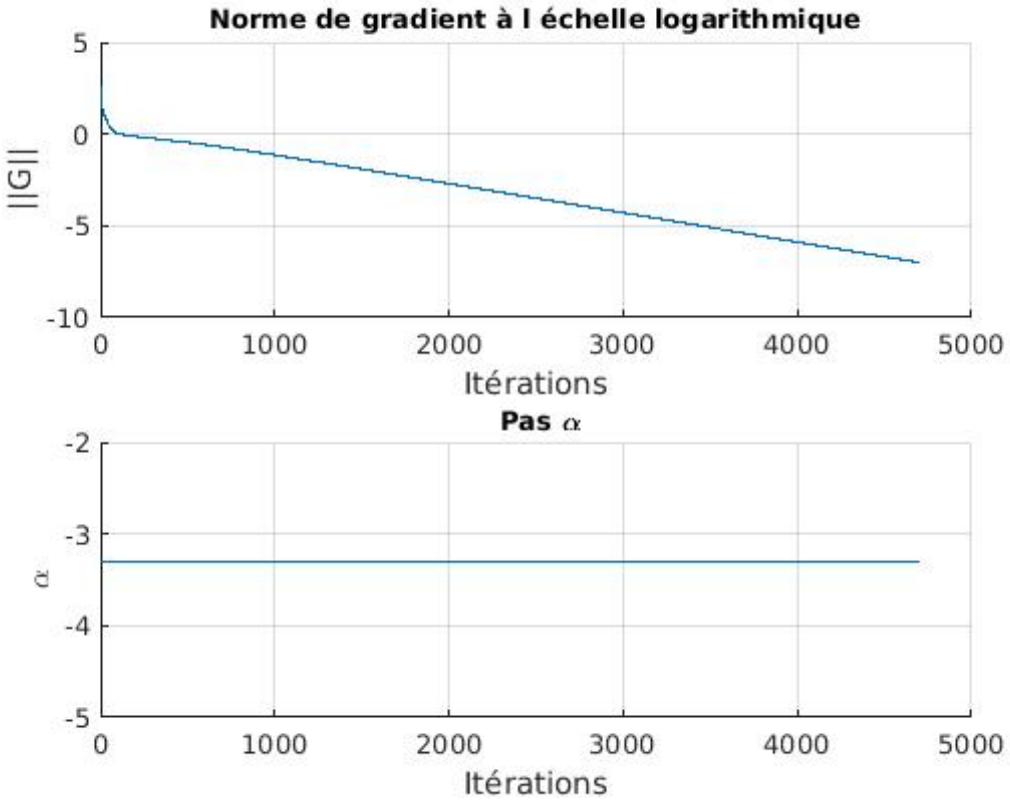

FIGURE 1 – Norme du gradient avec méthode du gradient à pas fixe

1.5.5 Méthode de Polak-Ribière : Méthode du gradient conjugué non linéaire à pas variable calculé par la méthode de Wolfe

Contrairement aux méthodes précédentes, cette méthode n'utilise pas la formule $d^{(k)} = -\nabla f(x_k)$ pour calculer la direction de descente. Cette direction est alors calculée par cette expression :

$$d^{(k)} = \begin{cases} -g^{(1)} & \text{si } k = 1 \\ -g^{(k)} + \beta^{(k)} d^{(k-1)} & \text{si } k \geq 2 \end{cases}$$

Avec $g^k = \nabla f(x_k)$ le gradient en x_k, et $\beta^{(k)}$ un coefficient donné par la formule :

$$\beta^{(k)} = \frac{(g^{(k)})^T (g^{(k)} - g^{(k-1)}}{||g^{(k-1)}||^2}$$

Cette méthode d'optimisation a été implémentée dans le fichier PR.m. La fonction PR cherche la valeur optimale d'une fonction en testant à chaque itération si la valeur du gradient est proche de 0. Si ce n'est pas le cas, on calcule la valeur de la direction avec les formules de la méthode Polak-Ribière présentées ci-dessus, et le pas α avec la fonction Wolfe qu'on a déjà implémentée. Ainsi, on met à jour la valeur du point courant en utilisant la formule $x^{(k+1)} = x^{(k)} + \alpha^{(k)} * d^{(k)}$.

En appliquant cette méthode à la fonction coût du réseau de distribution d'eau, on obtient :

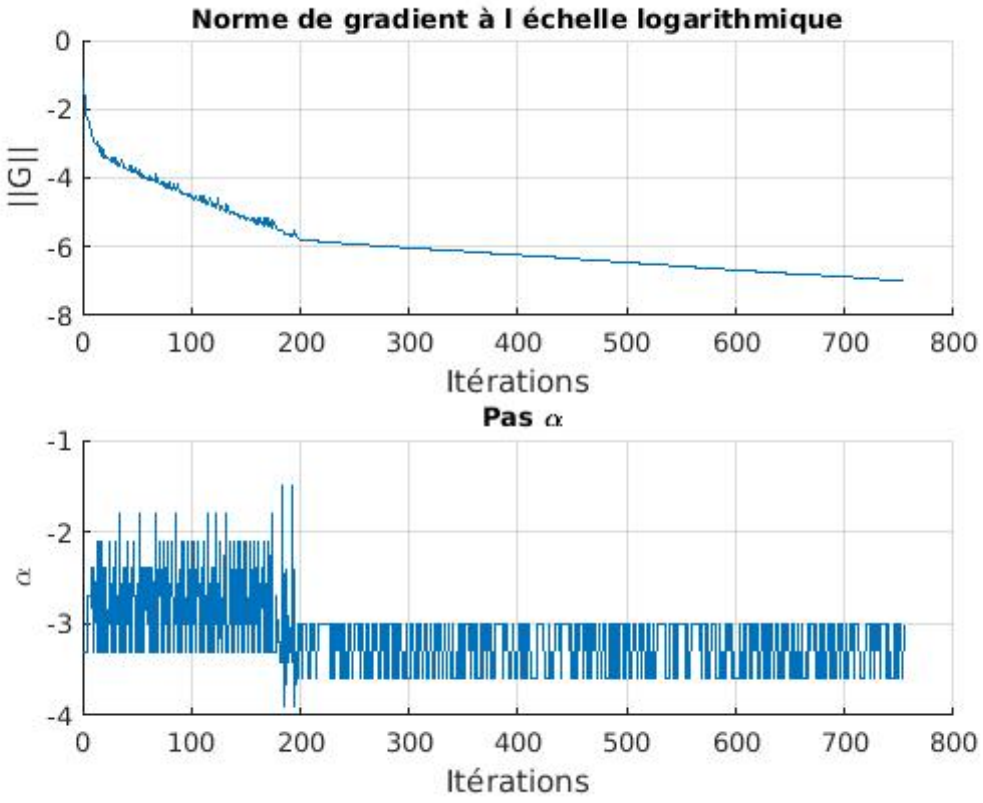

FIGURE 2 – Norme du gradient avec méthode du gradient à pas variable

1.5.6 Méthode de quasi-Newton (algorithme du BFGS)

En pratique, on évite de calculer la hessienne et son inverse, car ça devient coûteux, et la fonction étudiée peut ne pas pas être deux fois différentiable. Dans cette méthode, on cherche à construire des matrices qui approximment l'inverse de la Hessienne ou qui ont des propriétés similaires.

On note H_k les matrices qui remplaceront la Hessienne au point x_k de la descente et W_k l'inverse de H_k. On impose à ces matrices d'avoir des propriétés en commun avec la Hessienne :

$$H_k \text{ est symétrique comme la Hessienne}$$

En outre, par Taylor avec reste intégral, on a : :

$$G(x_{k+1}) - G(x_k) := \delta_g^{(k)} = \big(\int_0^1 \nabla^2 f(x_k + ts_k)dt \big) \delta_x^{(k)}$$

avec $\delta_x^{(k)} = x_{k+1} - x_k$. On impose donc

$$\delta_g^{(k)} = H_k \delta_x^{(k)}$$

Par un calcul, on trouve :

$$W_{k+1} = \big(I - \frac{\delta_x^{(k)} \delta_g^{(k)T}}{\delta_g^{(k)T} \delta_x^{(k)}}\big) W_k \big(I - \frac{\delta_g^{(k)} \delta_x^{(k)T}}{\delta_g^{(k)T} \delta_x^{(k)}}\big) + \frac{\delta_x^{(k)} \delta_x^{(k)T}}{\delta_g^{(k)T} \delta_x^{(k)}}$$

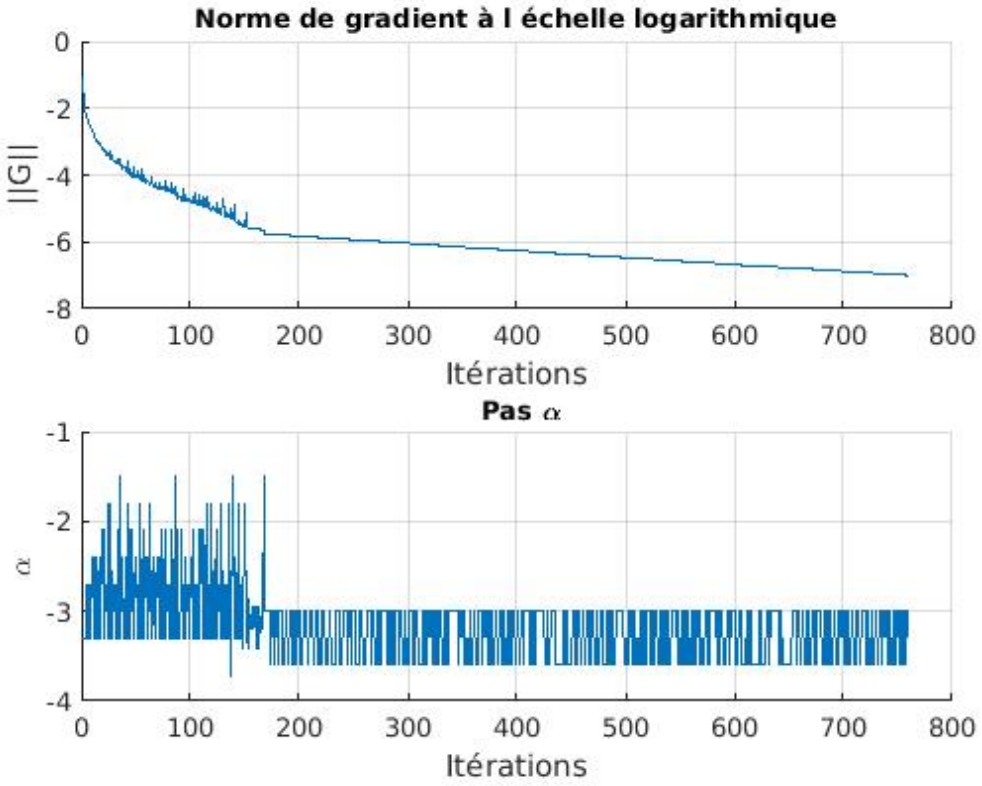

FIGURE 3 – Norme du gradient avec méthode de Poliak-Ribière pour xini=xopt

On crée alors un algorithme avec comme direction de descente d_k qui vérifie

$$d_k = -W_k G(x_k)$$

(donc plus ou moins semblable au $G'(x_k)d_k = -G(x_k)$ de l'algorithme de Newton)

En appliquant cette méthode à la fonction coût du réseau de distribution d'eau, on obtient :

Méthode de Newton :

La direction de descente de cette méthode est calculée avec la formule :

$$d^{(k)} = -\nabla^2 f(x^{(k)})^{-1} g^{(k)}$$

avec :

- f la fonction à minimiser
- $g^{(k)} = \nabla f(x^{(k)}$
- $\nabla^2 f(x^{(k)})$ le hessien au point $x^{(k)}$

On implémente cette méthode dans le fichier Newton.m. Cette fonction cherche la valeur optimale d'une fonction en testant à chaque itération si la valeur du gradient est proche de 0. Si ce n'est pas le cas, on calcule la valeur de la direction avec la formule de Newton présentée ci-dessus, et le pas α avec la fonction Wolfe qu'on a déjà implémentée. Ainsi, on met à jour la valeur du point courant en utilisant la formule $x^{(k+1)} = x^{(k)} + \alpha^{(k)} * d^{(k)}$.

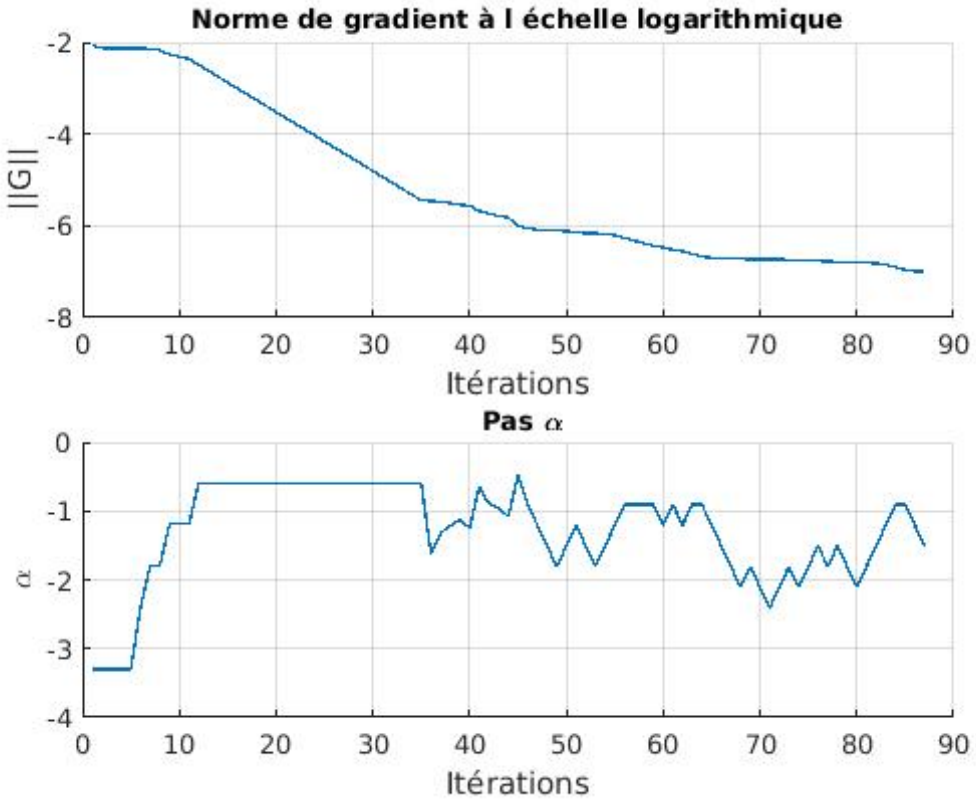

FIGURE 4 – Norme du gradient avec méthode de quasi-newton BFGS pour xini=xopt

En appliquant cette méthode à la fonction coût du réseau de distribution d'eau, on obtient :

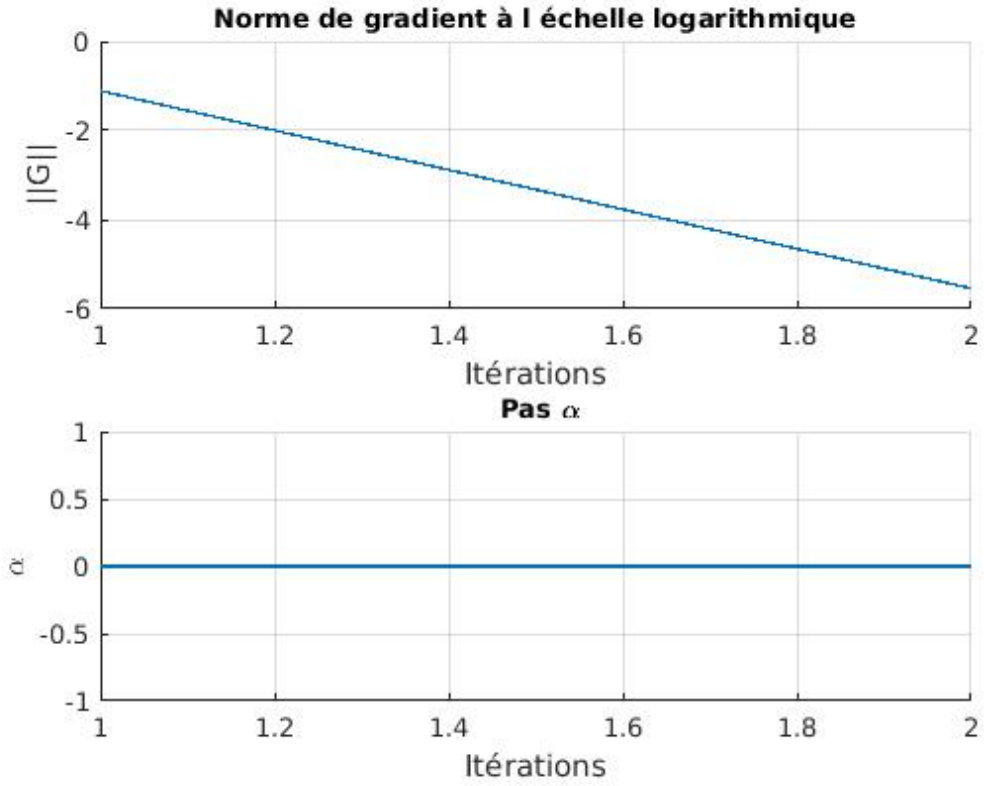

FIGURE 5 – Norme du gradient avec méthode de Newton pour xini=xopt

1.6 Comparaison des méthodes :

pas initial de $5 * 10^{-4}$	nombre d'itérations	temps CPU
gradient à pas fixe avec xini random	4705	0,51s
gradient à pas fixe avec xini amélioré	2099	0,33 s
gradient à pas variable avec xini amélioré	756	0,56 s
Polak-Ribière avec xini=0.1+0.1*ones	954	$2.27 * 10^{-1}$ s
Polak-Ribière avec xini random	935	$2.2 * 10^{-1}$ s
Polak-Ribière avec xini amélioré	761	$1.6 * 10^{-1}$ s
BFGS avec xini=0.1+ 0.1*ones	153	$6.4 * 10^{-2}$ s
BFGS avec xini random	84	$2,5 * 10^{-2}$ s
BFGS avec xini amélioré	88	$2.2 * 10^{-2}$ s
Newton avec pas initial=1 et xini random	8	$4.4 * 10^{-3}$ s
Newton avec pas initial=1 et xini=0.1+ 0.1*ones	7	$2.8 * 10^{-3}$ s
Newton avec pas initial=1 et xini amélioré	3	$1,02 * 10^{-3}$ s

Le xini amélioré est le point obtenu lors de la dernière itération du programme de la première ligne. Le fait qu'il soit proche de la solution diminue significativement le nombre d'itérations.

1.7 Analyse des résultats

1.7.1 Convergence

D'après l'analyse qu'on a faite sur le nombre d'itérations et le temps CPU, on remarque que les facteurs qui entrent en jeu sont d'une part le point de départ xini et le type d'algorithme de l'autre.

D'abord, on remarque que le fait de choisir un xini proche de la solution réduit considérablement le nombre d'itérations (on passe de 4705 à 2099 pour la méthode de gradient à pas fixe).

Ensuite, le type d'algorithme agit aussi sur les résultats à savoir le nombre d'itérations et le temps CPU. Par exemple, la méthode du gradient à pas variable converge en moins d'itérations que celle à pas fixe, mais en plus de temps. Les autres algorithmes (PR,BFGS et Newton) qui utilise un calcul plus élaboré de la direction de descente et du pas, donnent de meilleurs résultats : ils convergent en moins d'itérations et plus rapidement.

1.7.2 Comportement du pas α

Les algorithmes à pas fixe : pour la méthode de gradient à pas fixe et Newton, α reste constant tout au long de l'exécution.
On remarque que l'algorithme de Newton converge avec un pas privilégié de 1. Les autres algorithmes ne donnent pas de bons résultats avec un pas aussi grand.

Les algorithmes à pas variable : Pour la méthode de gradient à pas variable et Polak-Ribière, α varie de la même manière en oscillant autour de 10^{-3}. Pour BFGS, c'est différent. La variation du pas α paraît beaucoup plus aléatoire. Cela s'explique par le calcul de la direction de descente, qui se base sur gradient pour les deux premiers et sur une approximation de la hessienne pour BFGS.

1.7.3 Conclusion

Nous avons tout d'abord traduit notre problème d'optimisation du réseau d'eau en un problème de minimisation de l'énergie de réseau. Cela nous a poussé à étudier différentes méthodes algorithmiques pour résoudre ce problème. D'après notre étude de convergence, l'algorithme de Newton se démarque par sa rapidité d'exécution en un nombre très réduit d'itérations. Il est donc à privilégier dans les cas où les conditions de son application sont réunies. En effet, la méthode de Newton se base sur un calcul de la hessienne, ce qui présuppose que la fonction coût est 2 fois différentiable. Si ce n'est pas le cas, alors on pourra utiliser la méthode de quasi-Newton BFGS qui elle aussi converge en un nombre raisonnable d'itérations (88 pour xini amélioré et 84 pour xini random).

2 Problème Dual

Dans cette deuxième partie, on s'intéresse à la résolution de ce problème par **dualité**.

2.1 Ecriture du problème dual

Nous avons montré dans les TPs précédents que le problème primal pouvait se réécrire sous la forme :

$$min_{q\in\mathbb{R}^n} \frac{1}{3} < q, r \bullet q \bullet |q| > + < p_r, A_r q >$$

$$\text{Sous la contrainte } A_d q - f_d = 0$$

On va alors dualiser le problème précédent avec sa contrainte. Le lagrangien de ce problème s'écrit :

$$l(q,\lambda) = \frac{1}{3}\langle q, r \bullet q \bullet |q|\rangle + \langle p_r, A_r q\rangle + \langle \lambda, A_d q - f_d\rangle$$

Le problème primal correspond alors a :

$$\min_{q\in\mathbb{R}^n} \quad \max_{\lambda\in\mathbb{R}^{m_d}} \quad l(q,\lambda)$$

Et le problème dual est :

$$\max_{\lambda\in\mathbb{R}^{m_d}} \quad \min_{q\in\mathbb{R}^n} \quad l(q,\lambda)$$

A λ fixé, $l(q,\lambda)$ est continue et tend vers $+\infty$ quand |q| tend vers $+\infty$. Donc on peut définir :

$$\Phi(\lambda) = \min_{q\in\mathbb{R}^n} \quad l(q,\lambda)$$

Comme l est différentiable selon q, on sait que son minimum sera atteint en un point d'annulation de la différentielle par rapport à q.

$$\nabla_q l = r \bullet q \bullet |q| + A_r^T p_r + A_d^T \lambda$$

Il y a un seul point où cette fonction s'annule : quand $r \bullet q \bullet |q| = -A_r^T p_r - A_d^T \lambda$ soit quand :

$$\forall i \in [1,n], \quad q_i = signe\Big(-\frac{(A_r^T p_r + A_d^T\lambda)_i}{r_i}\Big)\sqrt{\Big|\frac{(A_r^T p_r + A_d^T\lambda)_i}{r_i}\Big|}$$

On note q^* ce point. On voit qu'il dépend de λ. On a alors $\Phi(\lambda) = l(q^*(\lambda), \lambda)$. Cette expression se simplifie d'ailleurs quand on injecte $r \bullet q^* \bullet |q^*| = -A_r^T p_r + A_d^T \lambda$:

$$\Phi(\lambda) = -\frac{2}{3}\langle q^*, r \bullet q^* \bullet |q^*| \rangle - \langle \lambda, f_d \rangle$$

On souhaite alors trouver le gradient et la Hessienne de Φ afin de pouvoir appliquer nos algorithmes dessus.

$$\nabla\Phi(\lambda) = -2\nabla_\lambda q^* * r \bullet q \bullet |q^*| - f_d$$

Or $r \bullet q^* \bullet |q^*| = -A_r^T p_r + A_d^T \lambda$ donc en différenciant on voit que $2\nabla_\lambda q^* * r \bullet |q^*| = -A_d$

$$\nabla\Phi(\lambda) = A_d q^* - f_d$$

Calcul de la hessienne :
Soient $h \in \mathbb{R}^{m_d}$ et $i \in 1, ..., m_d$ On a :

$$\nabla\Phi(\lambda + h)_i = \Sigma_{k=1}^{m_d}(A_d)_{ik} signe(-\frac{(A_r^T p_r + A_d^T(\lambda + h))_k}{r_k})\sqrt{\left|\frac{(A_r^T p_r + A_d^T(\lambda + h))_k}{r_k}\right|} - f_{di}$$

Or, pour h assez petit, on a :

$$signe(-\tfrac{(A_r^T p_r + A_d^T(\lambda+h))_k}{r_k}) = signe(-\tfrac{(A_r^T p_r + A_d^T \lambda)_k}{r_k}) = signe(q_k^*)$$

Et :

$$\sqrt{\left|\frac{(A_r^T p_r + A_d^T(\lambda + h))_k}{r_k}\right|} = \sqrt{signe(\frac{(A_r^T p_r + A_d^T(\lambda + h))_k}{r_k})\frac{(A_r^T p_r + A_d^T(\lambda + h))_k}{r_k}}$$

$$= \sqrt{signe(\frac{(A_r^T p_r + A_d^T \lambda)_k}{r_k})\frac{(A_r^T p_r + A_d^T \lambda)_k}{r_k} + signe(\frac{(A_r^T p_r + A_d^T \lambda)_k}{r_k})\frac{A_d^T h_k}{r_k}}$$

$$= \sqrt{signe(-q_k^*)\frac{(A_r^T p_r + A_d^T \lambda)_k}{r_k} + signe(-q_k^*)\frac{(A_d^T h)_k}{r_k}}$$

$$= \sqrt{signe(-q_k^*)\frac{(A_r^T p_r + A_d^T \lambda)_k}{r_k}} + \frac{signe(-q_k^*)\frac{(A_d^T h)_k}{r_k}}{2\sqrt{signe(-q_k^*)\frac{(A_r^T p_r + A_d^T \lambda)_k}{r_k}}} + o(signe(-q_k^*)\frac{(A_d^T h)_k}{r_k})$$

$$= |q_k^*| + \frac{signe(-q_k^*)\frac{(A_d^T h)_k}{r_k}}{2|q_k^*|} + o(h)$$

donc :

$$\nabla\Phi(\lambda + h)_i = \Sigma_{k=1}^{m_d}(A_d)_{ik} signe(q_k^*)[|q_k^*| + \frac{signe(-q_k^*)\frac{(A_d^T h)_k}{r_k}}{2|q_k^*|} + o(h)] - f_{di}$$

$$= \nabla\Phi(\lambda)_i - \Sigma_{k=1}^{m_d}(A_d)_{ik}\frac{1}{2r_k|q_k^*|}(A_d^T h)_k + o(h)$$

$$= \nabla\Phi(\lambda)_i - \frac{1}{2}(A_d * diag(\frac{1}{r \bullet |q|}) * A_d^T * h)_i + o(h)$$

Ainsi :

$$\nabla^2\Phi(\lambda) = -\frac{1}{2}A_d * diag(\frac{1}{r \bullet |q|}) * A_d^T$$

2.2 Implémentation dans Matlab

Après avoir fait les calculs, on implémente alors des fonctions OracleDG et OracleDH qui renvoient les trois différentielles de $-\Phi$ (en effet, maximiser Φ est équivalent à minimiser $-\Phi$) et on peut alors appliquer nos algorithmes de minimisation précédemment développés à ces fonctions.

Nous avons également redimensionné nos variables pour qu'elles soient de la même taille que λ et appliqué la méthode du gradient à pas fixe et cela fonctionne.

2.3 Résultats des méthodes sur le problème dual

2.3.1 Résultats numériques

pas initial de $5 * 10^{-4}$	nombre d'itérations	temps CPU
gradient à pas fixe	ne converge pas	
gradient à pas variable	5000	3.73 s
gradient à pas variable xini optimal	1440	3.73 s
Polak-Ribière avec pas de la méthode de Wolfe = 1 et xini rand	2361	8,71 10^{-1} s
Polak-Ribière avec pas de la méthode de Wolfe = 1 et xini optimal	761	2,54 10^{-1} s
BFGS	83	0,023 s
méthode de Newton simple (pas initial de 1)	20	4,59 10^{-3}
méthode de Newton pas initial optimal	3	0.09 s

Le xini optimal est le point obtenu lors de la dernière itération du programme de la deuxième ligne. Le fait qu'il soit proche de la solution diminue significativement le nombre d'itérations.

2.3.2 Commentaires :

On remarque qu'avec le problème dual, le gradient à pas fixe ne fonctionne pas. Le pas choisi dépend en effet de α, qui est adapté à la fonction primale et pas à la fonction duale.

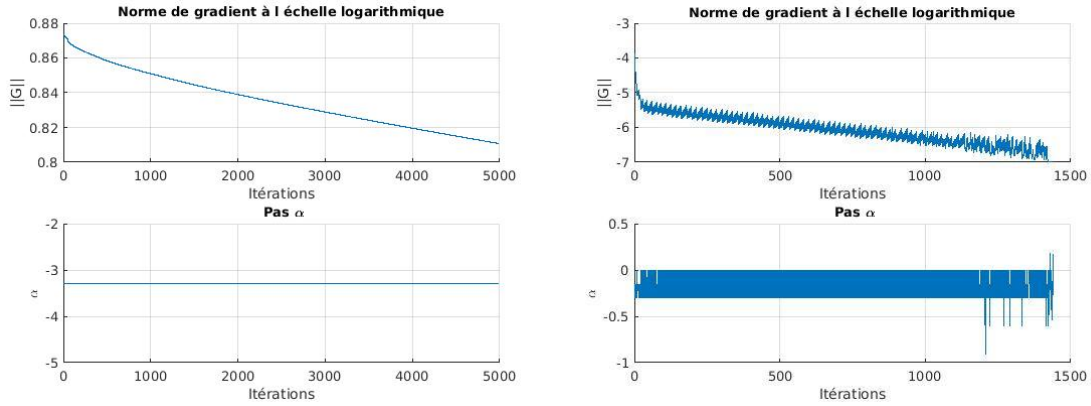

FIGURE 6 – Méthode du Gradient fixe et variable

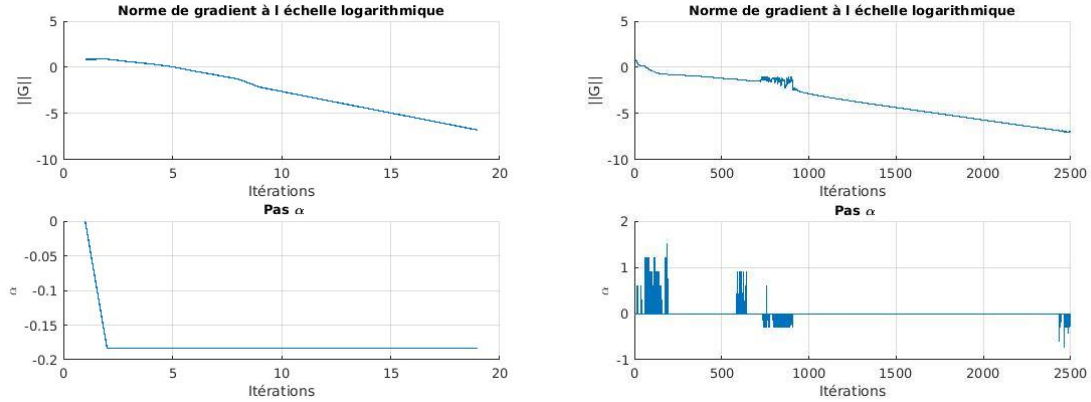

FIGURE 7 – Méthode de Newton et Polak-Ribière

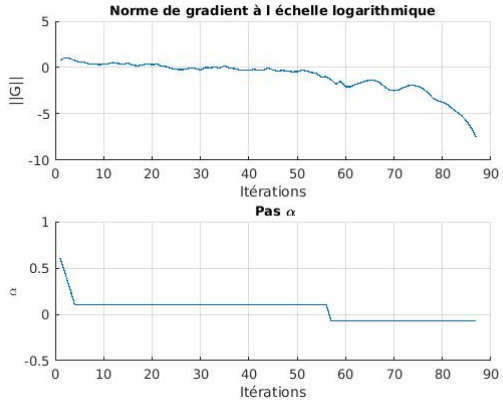

FIGURE 8 – Méthode BFGS

3 Gradient Projeté

3.1 Calcul du projeteur sur l'ensemble des contraintes

Soit y un élément de l'ensemble des contraintes : $A_d y - f_d = 0$. Et soit $x \in \mathbb{R}^{m_d}$. On a :

$$\begin{aligned} C &=< y - (x - A'_d * inv(A_d A'_d) * (A_d x - f_d)), x - (x - A'_d * inv(A_d A'_d) * (A_d x - f_d)) > \\ &=< A_d y - A_d x + A_d x - f_d, inv(A_d A'_d) * (A_d x - f_d) > \geq 0 \end{aligned}$$

Et ceci pour tout y dans l'ensemble des contraintes. Ainsi :

$$P_X(x) = x - A'_d * inv(A_d A'_d) * (A_d x - f_d)$$

3.2 Implémentation en Matlab

Une fois le projecteur sur l'ensemble des contraintes calculé, on souhaite maintenant implémenter l'algorithme du gradient projeté sur le problème primal, avec un pas fixe et la recherche linéaire de Wolfe.

Tout d'abord, nous avons créé une fonction **Projection.m** qui calcule la projection sur l'ensemble des contraintes d'un élément q rentré en paramètre.

Ensuite, sur le modèle des méthodes à gradient fixe et variable, nous avons implémenté l'oracle du gradient projeté qui prend en paramètre q et calcule la fonction coût associée et son gradient par rapport à q (on calculait le gradient par rapport à q_c précédemment).

Finalement, nous avons implémenté la routine **GradientP.m** qui prend en paramètre l'OracleGP et qui à chaque itération vérifié un critère d'arrêt qui a changé ici, et qui maintenant calcule la norme du projeté du gradient au lieu du gradient lui-même.

Résultats

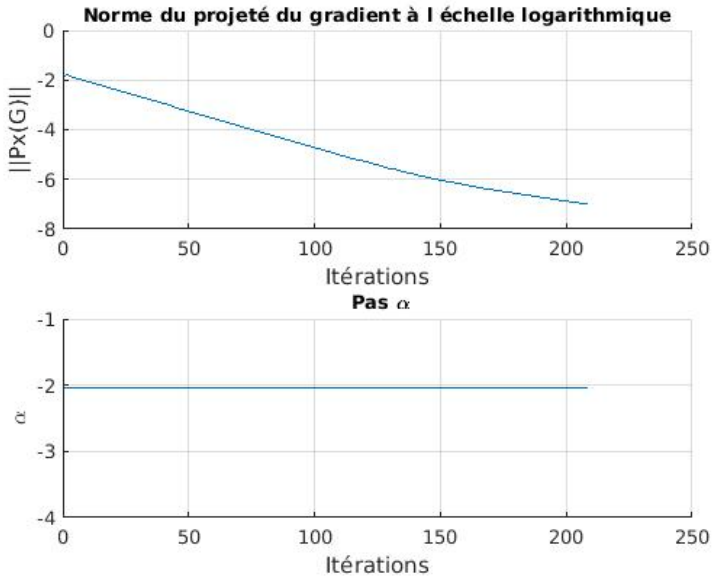

FIGURE 9 – Méthode Gradient Projeté

Le pas initial utilisé est égal à 0.0089. Ce pas est optimal car la méthode de Wolfe l'a gardé. Ainsi, on remarque dans la figure que le pas est resté constant.

Le xini utilisé est optimal. Il s'agit du point obtenu en lançant le programme avec un xini random.

3.3 Comparaison avec les autres algorithmes

Algorithme	nombre d'itérations	temps CPU
gradient à pas fixe	2099	0,33 s
gradient à pas variable	756	0,56 s
Polak-Ribière	761	$1.6 * 10^{-1}$ s
BFGS	88	$2.2 * 10^{-2}$ s
Newton avec pas initial=1	3	$1,02 * 10^{-3}$ s
Gradient Projeté	210	$2,82 * 10^{-1}$ s

Dans la résolution de ce problème, les méthodes des gradients à pas fixe et à pas variable sont les plus lentes, et sont également celles qui convergent en le plus d'itérations. Les méthodes BFGS et gradient projeté donne de bons résultats avec 88 et 210 itérations respectivement. Mais la méthode de Newton reste la meilleure pour résoudre ce problème. C'est la plus rapide, et converge en seulement 3 itérations.