ECOLE NATIONALE DES PONTS ET CHAUSSÉES

OPTIMISATION ET CONTRÔLE

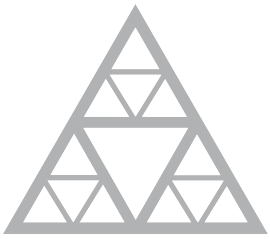
Projet sur les reseaux de distribution d'eau

Author:

Matthieu TOULEMENT

Tong ZHAO

Mai 2017



École des Ponts

ParisTech

1 Présentation du problème

On se concentre sur le problème de la résolution des équations décrivant l'état d'équilibre d'un réseau de distribution d'eau potable au cours de ce projet.

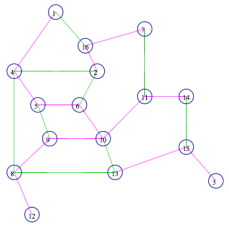


Figure 1: Un réseau d'eau potable

1.1 Modélisation du problème

Le réseau est modélisé par un graphe orienté. L'ensemble des sommets $\mathcal{N}$ est de cardinalit m tandis que l'ensemble des arcs $\mathcal{A}$ de cardinalité n. Parmi les sommets se trouvent des réservoirs $\mathcal{N}_r$ et des puits de demande $\mathcal{N}_d$ de cardinalités respectives m_r et m_d.

On impose la connexité du graphe qui se traduit par $n \geqslant m-1$. On définit la matrice d'incidence A, son rang vaut $m-1$

On définit de plus les entités suivantes :

- f : vecteur des flux aux noeuds
- p : vecteur des pressions aux noeuds
- r : vecteur des résistances des arcs
- q : vecteur des débits dans les arcs

1.2 Equations du modèle

Les équations décrivant les états d'équilibres sont :

- La première loi de Kirchoff : $Aqf - f = 0$
- La seconde loi de Kirchoff et loi d'Ohm non linéaire: $A^T p + r \bullet q \bullet |q| = 0$

1.3 Problème d'optimisation

L'énergie du réseau s'écrit :

$$\overline{J}(q, f_r) = \frac{1}{3} \sum_{\alpha \in \mathcal{A}} r_\alpha q_\alpha^2 |q_\alpha| + \sum_{i \in \mathcal{N}_r} p_i f_i \tag{1}$$

Trouver l'équilibre revient alors à résoudre le problème :

$$\min_{\substack{q \in \mathbb{R}^n \\ Aq - f = 0}} < q, r \bullet q \bullet |q| > + < p, f_r > \tag{2}$$

La partition du graphe en réservoirs et puits de demande induit une partition de la matrice A et des vecteurs p et f. Donc la contrainte est équivalente à :

$$A_r q = f_r \tag{3}$$

et

$$A_d q = f_d \tag{4}$$

Nous pouvons donc réécrire le problème comme :

$$\min_{\substack{q \in \mathbb{R}^n \\ A_d q - f_d = 0}} < q, r \bullet q \bullet |q| > + < p, A_r q > \tag{5}$$

Comme la matrice A_d est de rang plein, on peut en extraire une matrice carré de taille m_d inversible $A_{d,T}$. En posant $A_d = (A_{d,T} A_{d,c})$, $q = (q_T q_c)^T$ la contrainte du problème (5) se réécrit :

$$A_{d,T} q_T + A_{d,c} q_c = f_d \tag{6}$$

puis

$$q_T = A_{d,T}^{-1} (f_d - A_{d,c} q_c) \tag{7}$$

On peut donc réécrire le problème d'optimisation sous la forme :

$$\min_{q_c \in \mathcal{R}^{n-m_d}} \frac{1}{3} < q^{(0)} + B * q_c, r \bullet (q^{(0)} + B * q_c) \bullet |q^{(0)} + B * q_c| > + < p_r, A_r (q^{(0)} + B * q_c) > \tag{8}$$

où: $q^{(0)} = A_{d,T}^{-1} f_d$ et $B = \begin{pmatrix} -A_{d,T}^{-1} A_{d,c} \\ I_{n-m_d} \end{pmatrix}$

2 Première séance de TP

Dans cette partie, on s'intéresse au problème primal d'optimisation sans contrainte.

$$min_{q_C \in \mathbb{R}^{n-m_d}} \frac{1}{3} \left\langle q^{(0)} + B q_c, r \bullet (q^{(0)} + B q_c) \bullet |q^{(0)} + B q_c| \right\rangle + \left\langle p_r, A_r (q^{(0)} + B q_c) \right\rangle \tag{9}$$

2.1 Calcul du gradient

D'abord, on calcule le gradient du premier terme.

$$f_1(q_c) = \left\langle q^{(0)} + Bq_c, r \bullet (q^{(0)} + Bq_c) \bullet |q^{(0)} + Bq_c| \right\rangle \tag{10}$$

On pose $q = q^{(0)} + Bq_c$ et on calcule

$$\frac{\partial f_1}{\partial q} = \frac{\partial \langle q, r \bullet q \bullet |q| \rangle}{\partial q} = \frac{\partial (r \bullet q \bullet q \bullet |q|)}{\partial q} = 3sign(q) \bullet r \bullet q \bullet q = 3r \bullet q \bullet |q| \tag{11}$$

Ensuite, on calcule le gradient du second terme.

$$f_2(q_c) = \left\langle p_r, A_r(q^{(0)} + Bq_c) \right\rangle \tag{12}$$

$$\frac{\partial f_2}{\partial q} = \frac{\partial (p_r \bullet A_r q)}{\partial q} = A_r^T p_r \tag{13}$$

Sachant que $\dfrac{\partial q}{\partial q_c} = B^T$, on calcule le gradient de la fonction F.

$$G(q_c) = \frac{\partial F}{\partial q_c} = \frac{\partial (3f_1 + f_2)}{\partial q} \frac{\partial q}{\partial q_c} = B^T (r \bullet q \bullet |q| + A_r^T p_r) \tag{14}$$

2.2 Calcul du hessien

On veut calculer:

$$H(q_c) = \frac{\partial G}{\partial q_c} = \frac{\partial (B^T (r \bullet q \bullet |q| + A_r^T p_r))}{\partial q_c} \tag{15}$$

On pose $d = r \bullet q \bullet |q|$, alors

$$H(q_c) = \frac{\partial (B^T d)}{\partial d} \frac{\partial d}{\partial q} \frac{\partial q}{\partial q_c} \tag{16}$$

Sachant que:

$$\frac{\partial d}{\partial q} = \begin{bmatrix} \frac{\partial d_1}{\partial q_1} & \bullet & \frac{\partial d_1}{\partial q_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial d_n}{\partial q_1} & \bullet & \frac{\partial d_n}{\partial q_n} \end{bmatrix} \quad \text{d'où} \quad \frac{\partial d_i}{\partial q_j} = \begin{cases} 2r_i|q_i|, & i = j \\ 0, & i \neq j \end{cases}$$

On a:

$$\frac{\partial d}{\partial q} = diag((2r_i|q_i|)_{i=1\cdots n}) \tag{17}$$

On en déduit que:

$$H(q_c) = 2B^T diag((2r_i|q_i|)_{i=1\cdots n})B \tag{18}$$

2.3 Tester l'oracle avec deux méthodes fournies

On teste notre oracle en l'interfaçant avec un algorithme de gradient à pas fixe et on le compare avec la fonction optim de Scilab. Les résultats sont comme suit:

```
 Fin de la methode d'optimisation integree a Scilab

!Temps CPU         :   0.001429   !
!                                 !
!Critere optimal   :   -3.734007  !
!                                 !
!Norme du gradient :   0.0000003  !

 Verification des equations d'equilibre du reseau

!sur les debits    :   2.220D-16  !
!                                 !
!sur les pressions :   0.0000002  !
```

Figure 2: Le calcul avec Optim_scilab

```
 Fin de la methode de gradient a pas fixe

!Iteration         :   4095       !
!                                 !
!Temps CPU         :   0.80611    !
!                                 !
!Critere optimal   :   -3.734007  !
!                                 !
!Norme du gradient :   0.0000010  !

 Verification des equations d'equilibre du reseau

!sur les debits    :   9.714D-17  !
!                                 !
!sur les pressions :   0.0000010  !
```

Figure 3: Le calcul avec Gradient_F

On visualise l'évaluation de la norme du gradient (echelle logarithmique) en fonction d'itération.

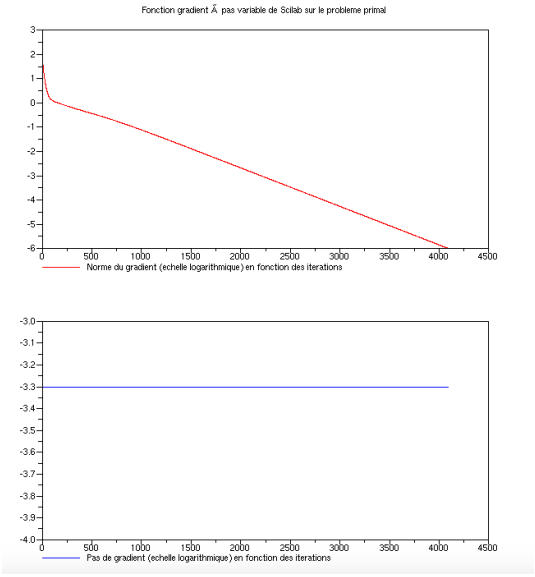


Figure 4: La performance du Gradient_F

A l'observation, on voit que la vitesse de convergence est assez lent donc il prend plus de temps ($\times$ 800) que la fonction optim de Scilab. De plus, la perfoemance de convergence est pire que celle de la fonction optim de Scilab.

3 Seconde et troisième séance de TP

Lors de cette séance nous souhaitons étudier le problème (1) à l'aide de 4 algorithmes d'optimisations différents.

3.1 Fletcher - Lemaréchal

Dans cette partie, on va programmer une recherche linéaire vérifiant les conditions de Wolfe et coder l'algorithme de gradient à pas variable utilisant cette recherche linéaire.

En prenant les notations dans l'énoncé de TP, les conditions de Wolfe sont comme suit:

- La fonction f doit décroître de manière significative

$$f(x^{(k)} + \alpha^{(k)} d^{(k)}) \leq f(x^{(k)}) + \omega_1 \alpha^{(k)} {g^{(k)}}^T d^{(k)}$$

- Le pas $\alpha^{(k)}$ doit être suffisamment grand

$$(\nabla f(x^{(k)} + \alpha^{(k)} d^{(k)}))^T d^{(k)} \geq \alpha_2 {g^{(k)}}^T d^{(k)}$$

avec $0 < \omega_1 < \omega_2 < 1$.

On utilise *Gradient_W.sci* pour tester le Wolfe et on obtient le résultat suivant.

```
 Fin de la methode de gradient a pas variable

!Iteration          :   296         !
!                                   !
!Temps CPU          :   0.355025    !
!                                   !
!Critere optimal    :   -3.734007   !
!                                   !
!Norme du gradient  :   0.0000010   !

 Verification des equations d'equilibre du reseau

!sur les debits     :   6.939D-17   !
!                                   !
!sur les pressions  :   0.0000010   !
```

Figure 5: La performance du Grandient_W

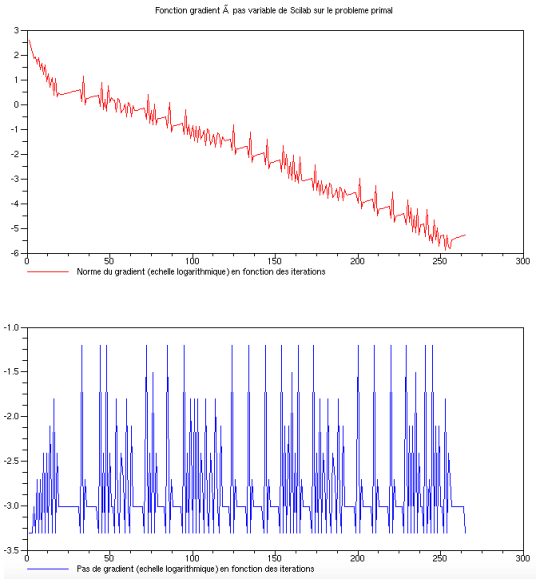


Figure 6: La performance du Grandient_W

A l'observation, on voit que le gradient augmente de temps en temps, surtout quadn le pas change.

3.2 Polak - Ribière

L'algorithme de Polak-Ribière est comme suit:

Algorithm 1 Polak-Pibère

Initialiser x_0

$f_0 = f(x_0), \;\; g_0 = f'(x_0), \;\; d_0 = -g_0$

$x_1 = x_0 + d_0$

$n = 1$

while True **do**

  if $\|g(x_n)\| \leq \epsilon$ **then** return

  $\beta_n = \dfrac{g_n^T(g_n - g_{n-1})}{\|g_{n-1}\|_2^2}$

  $d_n = -g_n + \beta_n d_{n-1}$

  α_n calculé par Wolfe

  $x_{n+1} = x_n + \alpha_n * d_n$

  $n = n + 1$

Le résultat est présenté suivant:

```
 Fin de la methode de gradient conjugue non lineaire

!Iteration          :   177         !
!                                   !
!Temps CPU          :   0.248885    !
!                                   !
!Critere optimal    :   -3.734007   !
!                                   !
!Norme du gradient  :   0.0000007   !

 Verification des equations d'equilibre du reseau

!sur les debits     :   1.249D-16   !
!                                   !
!sur les pressions  :   0.0000005   !
```

Figure 7: La performance du Grandient_Conj

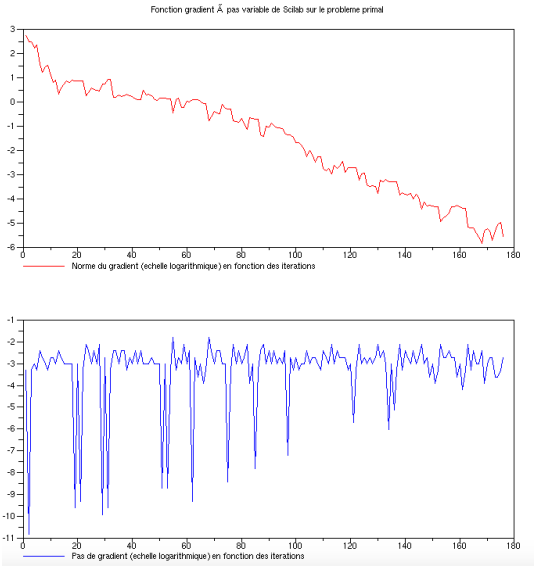


Figure 8: La performance du Grandient_Conj

A l'aide de algorithme de Polak - Ribière, la vitesse augmente et le changement de pas est plus doux.

3.3 BFGS

L'idée principale de cette méthode est d'approximer la matrice hessienne pour éviter le calcul coûteux.

On pose $\delta_x^{(k)} = x^{(k+1)} - x^{(k)}$ et $\delta_G^{(k)} = \nabla J(x^{(k+1)}) - \nabla J(x^{(k)}$, et chaque itération on utilise la formule suivante pour mettre à jour l'approximation de la matrice de Hessienne:

$$W^{(k+1)} = (I - \frac{\delta_x^{(k)} {\delta_G^{(k)}}^T}{{\delta_G^{(k)}}^T}\delta_x^{(k)})W^{(k)}(I - \frac{\delta_G^{(k)} {\delta_x^{(k)}}^T}{{\delta_G^{(k)}}^T}\delta_x^{(k)}) + \frac{\delta_x^{(k)} {\delta_x^{(k)}}^T}{{\delta_G^{(k)}}^T}\delta_x^{(k)}$$

En prenant $d_k = -W^{(k)}G^{(k)}$, on calcule $x^{(k+1)} = x^{(k)} + \alpha_k d_k$ où α_k est optimisé par l'algorithme de Wolfe.

```
 Fin de la methode de gradient quasi-Newton

!Iteration          :   22          !
!                                   !
!Temps CPU          :   0.019164    !
!                                   !
!Critere optimal    :   -3.734007   !
!                                   !
!Norme du gradient  :   0.0000005   !

 Verification des equations d'equilibre du reseau

!sur les debits     :   1.665D-16   !
!                                   !
!sur les pressions  :   0.0000003   !
```

Figure 9: La performance de Quasi_Newton

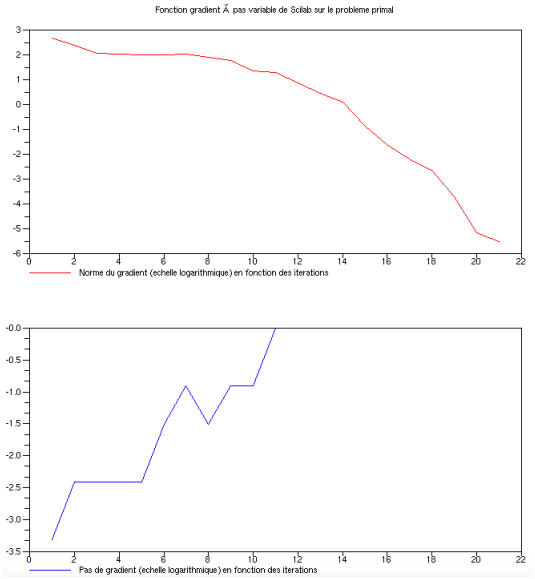


Figure 10: La performance de Quasi_Newton

A l'observation, on voit que la vitesse de convergence est assez rapide.

3.4 Méthode de Newton

La méthode de Netwon utilise la matrice de Hessienne pour calculer la direction de descente. On pose $d_k = -H_k^{-1}G$ et on calcule $x_{k+1} = x_k + \alpha_k d_k$.

```
 Fin de la methode de gradient (Newton)

!Iteration          :   6           !
!                                   !
!Temps CPU          :   0.004687    !
!                                   !
!Critere optimal    :   -3.734007   !
!                                   !
!Norme du gradient  :   3.927D-09   !
```

```
 Verification des equations d'equilibre du reseau

!sur les debits    :   2.220D-16  !
!                                 !
!sur les pressions :   2.016D-13  !
```

Figure 11: La performance de Newton

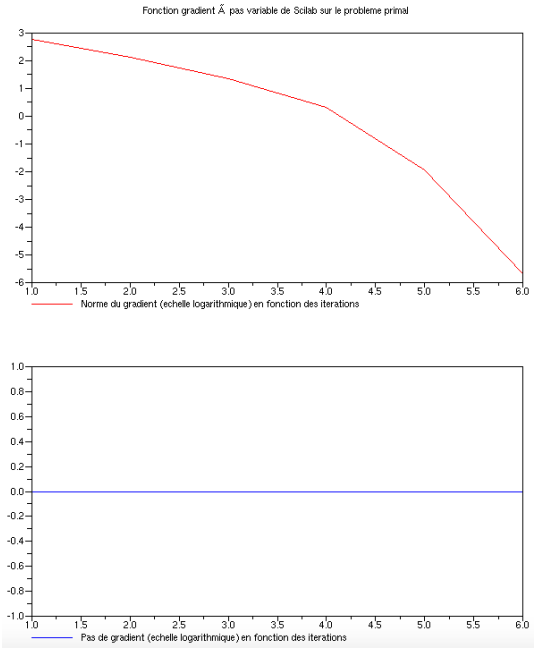


Figure 12: La performance de Newton

3.5 Tableau - Comparatif des résultats

Dans cette partie, on compare tous les algorithmes qu'on utilise et on analyse les différences entre eux.

Algorithme	Itération	Temps CPU	Critère optimal	Norme de gradient
La fonction optim de Scilab	-	0.001429	-3.734007	3×10^{-7}
Le gradient à pas fixe	4095	0.80611	-3.734007	10^{-6}
Le gradient à pas variable	296	0.355025	-3.734007	10^{-6}
Le gradient conjugué	177	0.248885	-3.734007	7×10^{-7}
L'algorithme de quasi-Newton	22	0.019164	-3.734007	5×10^{-7}
L'algorithme de Newton	6	0.004687	-3.734007	3.927×10^{-9}

Au travers du tableau, on voit que l'algorithme de Newton nous donne le meilleur résultat. De plus il est le plus rapide parmi tous les algorithmes qu'on implémente par soi même. Cependant il faudrait noter que l'inverse de la matrice de Hessienne est quelque fois difficile à calculer. Dans ce cas, cet algorithme ne marche plus. On a le classement suivant:

Newton > Quasi-Newton(BFGS) > Polak-Ribière > Le gradient à pas variable > Le gradient à pas fixe

4 Quatrième séance de TP

Dans ce TP nous avons étudié la résolution de ce problème par la dualité. Pour primal nous considérons le problème :

$$\begin{aligned} &\min_{q\in\mathbb{R}^n} \frac{1}{3} < q, r \bullet q \bullet |q| > + < p_r, A_r q > \\ &s.c. A_d q - f_d = 0 \end{aligned} \tag{19}$$

Lagrangien Le Lagrangien du problème s'écrit :

$$\mathcal{L}(q,\lambda) = \frac{1}{3} < q, r \bullet q \bullet |q| > + < p_r, A_r q > + \lambda^T (A_d q - f_d) \tag{20}$$

Pour trouver les valeurs extrêmes de $\mathcal{L}$, on calcule le gradient:

$$\frac{\partial \mathcal{L}(q,\lambda)}{\partial q} = r \bullet q \bullet |q| + A_r^T p_r + A_d^T \lambda \tag{21}$$

Soit $\hat{q}$ la valeur extrême du Lagrangien, on a:

$$r \bullet \hat{q} \bullet |\hat{q}| = -(A_r^T p_r + A_d^T \lambda) \tag{22}$$

$$C_i = \hat{q}_i |\hat{q}_i| = -\frac{1}{r_i} (\sum_{j=1}^{m_r} A_r(j,i) p_r(j) + \sum_{j=1}^{m_d} A_d(j,i) \lambda(j)) \qquad \forall i \in [1,n] \tag{23}$$

$$\hat{q}_i = -signe(C_i)\sqrt{|C_i|} \qquad \forall i \in [1,n] \tag{24}$$

$$\tag{25}$$

Fonction duale Le problème dual est:

$$\max_{\lambda\in\mathbb{R}^{m_d}} \min_{q\in\mathbb{R}^n} \mathcal{L}(q,\lambda) \tag{26}$$

La fonction duale est donc de la forme :

$$\forall \lambda \in \mathbb{R}^d : H(\lambda) = \min_{q\in\mathbb{R}^n} \mathcal{L}(q,\lambda) = -\frac{1}{3} \hat{q}(\lambda)^T (A_d^T \lambda + A_r^T p_r) + p_r^T A_r \hat{q}(\lambda) \tag{27}$$

Gradient de la fonction duale Ce gradient est simple à exprimer car $\hat{q}(\lambda)$ vérifie $\frac{\partial \mathcal{L}}{\partial q}(\hat{q}(\lambda), \lambda) = 0$

Il vient :

$$\nabla H(\lambda) = \frac{\partial \mathcal{L}}{\partial \lambda}(\hat{q}(\lambda), \lambda) \tag{28}$$

Qui devient :

$$\nabla H(\lambda) = A_d \hat{q}(\lambda) - f_d \tag{29}$$

Hessien de la fonction duale Le hessien de la fonction duale est de la forme :

$$\nabla^2 H(\lambda) = A_d \nabla \hat{q}(\lambda) \tag{30}$$

Pour déterminer le gradient de l'argmin : $\hat{q}(\lambda)$ procédons coordonnées par coordonnées.

$\forall i \in [1, n], \forall j \in [1, d]$

$$\frac{\partial \hat{q}_i}{\partial \lambda_j} = -signe(C_i) \frac{\frac{\partial |C_i|}{\partial \lambda_j}}{2\sqrt{|C_i|}} \tag{31}$$

Comme $\frac{\partial |C_i|}{\partial \lambda_j} = \frac{1}{r_i} A_d[j, i]$ il vient :

$$\frac{\partial \hat{q}_i}{\partial \lambda_j} = -signe(C_i) \frac{A_d(j, i)}{2 r_i \sqrt{|C_i|}} \tag{32}$$

Ainsi en posant $\delta_i = -\frac{1}{2 r_i \sqrt{|C_i|}}$ nous avons la forme simple suivante :

$$\nabla \hat{q}(\lambda) = diag(\delta_1, .., \delta_n) A_d^T \tag{33}$$

Puis, finalement nous pouvons écrire le hessien de la fonction duale :

$$\nabla^2 H(\lambda) = A_d diag(\delta_1, .., \delta_n) A_d^T \tag{34}$$

5 Algorithmes d'optimisation pour le problème dual

On résoudre le problème dual à l'aide des algorithmes développés lors des séances dernières. En pratique, on s'intéresse à la minimisation de l'opposé de fonction objective $\mathcal{L}$.

D'abord on utilise la fonction optim fourni par Scilab. On affiche le résultat pour faire la comparaison avec nos algorithmes.

```
 Fin de la methode d'optimisation integree a Scilab

!Temps CPU         :   0.007465   !
!                                 !
!Critere optimal   :   3.734007   !
!                                 !
!Norme du gradient :   5.757D-09  !
```

```
 Verification des equations d'equilibre du reseau

!sur les debits    :   4.191D-09  !
!                                 !
!sur les pressions :   0          !
```

Figure 13: La performance de la fonction Optim Scilab

5.1 Gradient à pas fixe

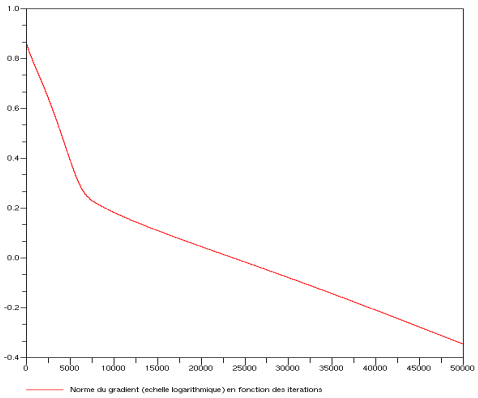
```
 Fin de la methode de gradient a pas fixe

!Iteration          :   50000       !
!                                   !
!Temps CPU          :   36.236974   !
!                                   !
!Critere optimal    :   31.031862   !
!                                   !
!Norme du gradient  :   0.4505867   !
```

```
 Verification des equations d'equilibre du reseau

!sur les debits     :   0.1732179   !
!                                   !
!sur les pressions  :   0           !
```

Figure 14: La performance du Gradient à pas fixe



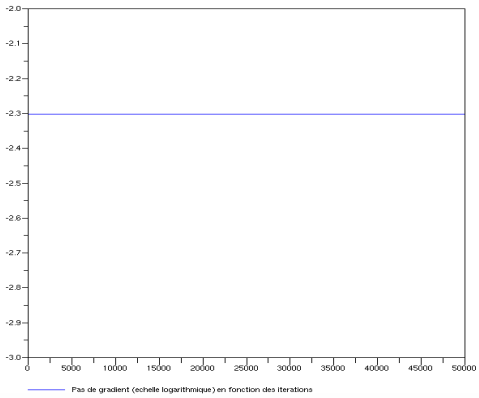


Figure 15: La performance du Gradient à pas fixe

A l'observation, on voit que le grqdient à pas fixe ne suffit pas à resoudre le problème dual. Bien qu'on a multiplié le nombre d'itération par 10 et le pas par 10, le résultat ne converge pas.

5.2 Fletcher - Lemaréchal

```
 Fin de la methode de gradient a pas variable

!Iteration          :   6031        !
!                                   !
!Temps CPU          :   2.619408    !
!                                   !
!Critere optimal    :   3.734007    !
!                                   !
!Norme du gradient  :   0.0000010   !
```

```
 Verification des equations d'equilibre du reseau

!sur les debits     :   0.0000008   !
!                                   !
!sur les pressions  :   0           !
```

Figure 16: La performance du Gradient à pas variable

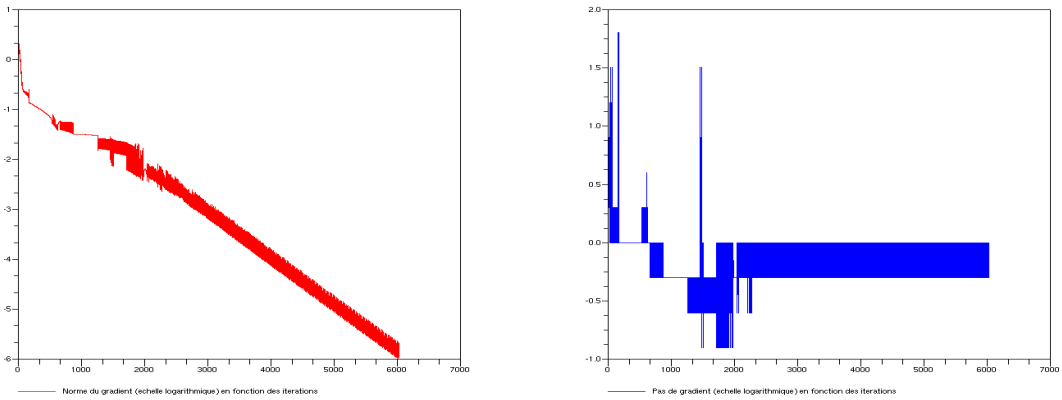


Figure 17: La performance du Gradient à pas variable

Le gradient à pas variable résout le problème dual mais la vitesse est lente.

5.3 Polak - Ribière

```
 Fin de la methode de gradient conjugue non lineaire

!Iteration          :   1752        !
!                                   !
!Temps CPU          :   0.727864    !
!                                   !
!Critere optimal    :   3.734007    !
!                                   !
!Norme du gradient  :   0.0000010   !
```

```
 Verification des equations d'equilibre du reseau

!sur les debits     :   0.0000008   !
!                                   !
!sur les pressions  :   0           !
```

Figure 18: La performance du Gradient conjugé

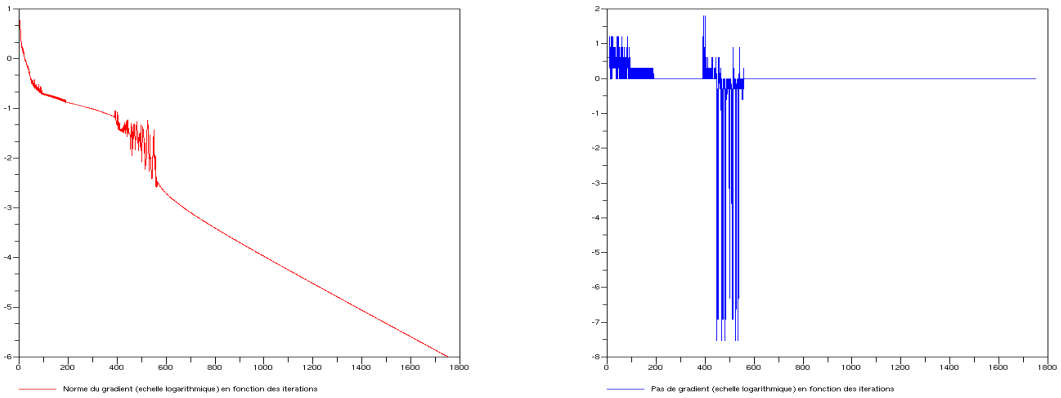


Figure 19: La performance du Gradient conjugé

5.4 BFGS

```
 Fin de la methode de gradient quasi-Newton

!Iteration          :   83          !
!                                   !
!Temps CPU          :   0.035059    !
!                                   !
!Critere optimal    :   3.734007    !
!                                   !
!Norme du gradient  :   8.550D-08   !
```

```
 Verification des equations d'equilibre du reseau

!sur les debits     :   5.005D-08   !
!                                   !
!sur les pressions  :   0           !
```

Figure 20: La performance du Gradient de Quasi-Newton

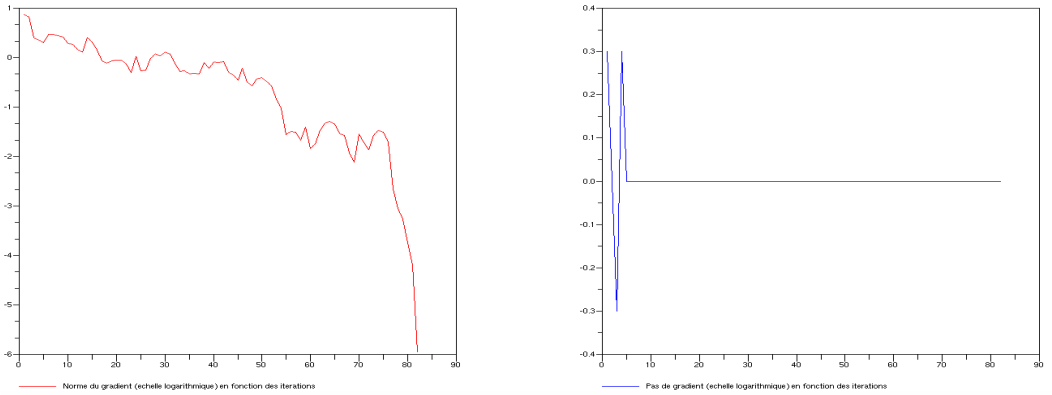


Figure 21: La performance du Gradient de Quasi-Newton

5.5 Méthode de Newton

```
 Fin de la methode de gradient (Newton)

!Iteration          :   23          !
!                                   !
!Temps CPU          :   0.014822    !
!                                   !
!Critere optimal    :   3.734007    !
!                                   !
!Norme du gradient  :   0.0000010   !
```

```
 Verification des equations d'equilibre du reseau

!sur les debits     :   0.0000008   !
!                                   !
!sur les pressions  :   0           !
```

Figure 22: La performance du Gradient de Newton

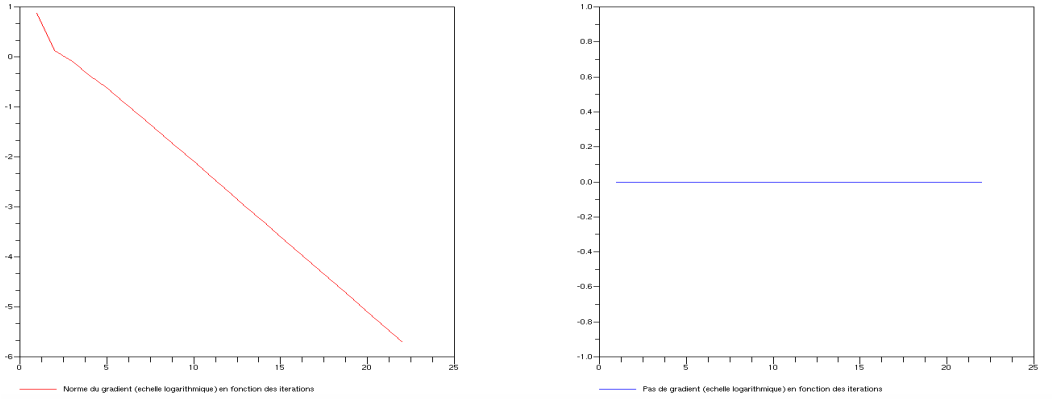


Figure 23: La performance du Gradient de Newton

5.6 Tableau - Comparatif des résultats

Algorithme	Problème	Itération	Temps CPU	Critère optimal	Norme de gradient
La fonction optim de Scilab	Primal	-	0.001429	-3.734007	3×10^{-7}
	Dual	-	0.007465	3.734007	5.757×10^{-9}
Le gradient à pas fixe	Primal	4095	0.80611	-3.734007	10^{-6}
	Dual	50000	36.236974	31.031862	0.4505867
Flechter-Lemaréchal	Primal	296	0.355025	-3.734007	10^{-6}
	Dual	6031	2.619408	3.734007	10^{-6}
Polak-Ribière	Primal	177	0.248885	-3.734007	7×10^{-7}
	Dual	1752	0.727864	3.734007	10^{-6}
BFGS	Primal	22	0.019164	-3.734007	5×10^{-7}
	Dual	83	0.035059	3.734007	8.55×10^{-8}
L'algorithme de Newton	Primal	6	0.004687	-3.734007	3.927×10^{-9}
	Dual	23	0.014822	3.734007	10^{-6}

Au travers le tableau, on voit bien que le problème dual est plus difficile à optimiser que le problème primal.

On en déduit le classement d'algorithmes:

Newton > BFGS > Polak-Pibère > Flechter-Lemaréchal > Gradient à pas fixe