TP Optimisation sans contraintes
Méthodes de Newton, quasi-Newton et du gradient

1. Méthodes de Newton.

La méthode de Newton est l'une des plus utilisées pour résoudre les équations non-linéaires du type $f(x) = 0$ ou $\nabla f(x) = 0$.

1.1. Introduction pour la dimension 1.

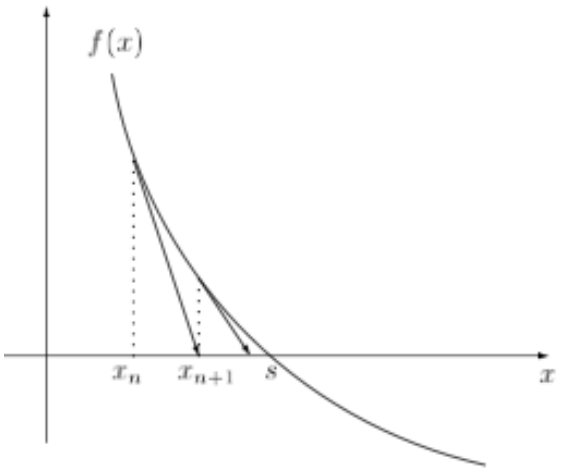

Figure 1: Méthode de Newton

Sur la figure 1 est tracée la courbe d'une fonction f sur un intervalle $[a, b]$ qui s'annule en un point s.
La méthode de Newton consiste à trouver une valeur approchée de cette valeur s. Cette méthode appelée aussi méthode des tangentes est un schéma itératif.
A partir d'un point x^n donné dans l'intervalle $[a, b]$, on construit la tangente à la courbe de f au point x^n. Cette tangente coupe la droite $y = 0$ en un point x^{n+1} qui est plus proche de la valeur s recherchée. Si la distance entre les deux points x^n est x^{n+1} n'est pas suffisamment petite par rapport à une erreur fixée, alors on continue et on construit la tangente à courbe au point x^{n+1} qui coupe la droite $y = 0$ en un point x^{n+2} . Si la distance entre les deux points x^{n+1} est x^{n+2} n'est pas suffisamment petite, on continue, sinon on arrête et x^{n+2} sera une valeur approchée de la valeur s avec une erreur que l'on s'était fixée.
L'équation de la tangente à la courbe de la fonction f au point x^n qui passe par x^{n+1} est :

$$y = f'(x^n)(x - x^n) + f(x^n).$$

La tangente coupe l'axe des abscisses au point x^{n+1}, on obtient :

$$x^{n+1} = x^n - \frac{f(x^n)}{f'(x^n)}$$

Data : f une fonction différentiable, x^0 un point initial et une tolérance ϵ.
Result : x une approximation de la solution
$n = 0$;
Calculer $x^1 = x^0 - \frac{f(x^0)}{f'(x^0)}$;
while $|x^{n+1} - x^n| > \epsilon$ **do**
  $x^{n+1} = x^n - \frac{f(x^n)}{f'(x^n)}$;
  $n = n + 1$;
  Remise à jour du critère d'arrêt;
end
$x = x^n$

Algorithme 1 : Algorithme de Newton pour $f(x) = 0$

1.2. En dimension 2.

Dans cette partie, nous voulons calculer une solution approchée de l'équation $\Phi(x, , y) = 0$ dans $\mathbb{R}^2$ comme par exemple $\Phi(x, y) = \nabla f(x, y)$.

Attention, ici Φ est un vecteur de $\mathbb{R}^2$, $\Phi(x, y) = \begin{pmatrix} \Phi_1(x, y) \\ \Phi_2(x, y) \end{pmatrix}$. Pour trouver une approximation du point $X^{n+1} = \begin{pmatrix} x^{n+1} \\ y^{n+1} \end{pmatrix}$ tel que

$$\begin{pmatrix} \Phi_1(x^{n+1}, y^{n+1}) \\ \Phi_2(x^{n+1}, y^{n+1}) \end{pmatrix} \simeq \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

nous allons utiliser la même technique que les cas de la dimension 1.
A partir d'un point $X^n = \begin{pmatrix} x^n \\ y^n \end{pmatrix}$ donné dans le plan $\mathbb{R}^2$, on construit x^{n+1} et y^{n+1} tel que

$$\Phi_1(x^n, y^n) + \frac{\partial \Phi_1}{\partial x}(x^n, y^n)\left(x^{n+1} - x^n\right) + \frac{\partial \Phi_1}{\partial y}(x^n, y^n)\left(y^{n+1} - y^n\right) = 0, \tag{1}$$

$$\Phi_2(x^n, y^n) + \frac{\partial \Phi_2}{\partial x}(x^n, y^n)\left(x^{n+1} - x^n\right) + \frac{\partial \Phi_2}{\partial y}(x^n, y^n)\left(y^{n+1} - y^n\right) = 0. \tag{2}$$

Il est possible d'écrire ces deux dernières équations sous la forme

$$J_\Phi(x^n, y^n).\begin{pmatrix} x^{n+1} - x^n \\ y^{n+1} - y^n \end{pmatrix} + \begin{pmatrix} \Phi_1(x^n, y^n) \\ \Phi_2(x^n, y^n) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

avec J_Φ la matrice Jacobienne de Φ :

$$J_\Phi = \begin{pmatrix} \frac{\partial \Phi_1}{\partial x} & \frac{\partial \Phi_1}{\partial y} \\ \frac{\partial \Phi_2}{\partial x} & \frac{\partial \Phi_2}{\partial y} \end{pmatrix}$$

On construit donc les valeurs x^{n+1}, y^{n+1} en résolvant le système

$$J_\Phi(x^n, y^n).\begin{pmatrix} x^{n+1}-x^n \\ y^{n+1}-y^n \end{pmatrix} = -\begin{pmatrix} \Phi_1(x^n, y^n) \\ \Phi_2(x^n, y^n) \end{pmatrix}$$

donc la solution (quand elle existe!) est

$$\begin{pmatrix} x^{n+1} \\ y^{n+1} \end{pmatrix} = \begin{pmatrix} x^n \\ y^n \end{pmatrix} - J_\Phi^{-1}(x^n, y^n).\begin{pmatrix} \Phi_1(x^n, y^n) \\ \Phi_2(x^n, y^n) \end{pmatrix},$$

avec J_Φ^{-1} l'inverse de la matrice Jacobienne J_Φ. Si la distance entre les deux points X^n est X^{n+1} n'est pas suffisamment petite par rapport à une erreur fixée, alors on continue le processus et on construit le point X_{n+2} de la même manière sinon arrête l'algorithme.

En Matlab la commande $u = A\backslash b$ donne la solution du système $Au = b$

Data : Φ une fonction différentiable, une valeur initiale $X^0 = \begin{pmatrix} x^0 \\ y^0 \end{pmatrix}$ et une tolérance ϵ.
Result : X une approximation de la solution
$n = 0$;
Résoudre le système

$$J_\Phi(x^0, y^0).\begin{pmatrix} x^1-x^0 \\ y^1-y^0 \end{pmatrix} = -\begin{pmatrix} \Phi_1(x^0, y^0) \\ \Phi_2(x^0, y^0) \end{pmatrix}$$

pour trouver le point $X^1 = \begin{pmatrix} x^1 \\ y^1 \end{pmatrix}$.
while $|x^{n+1} - x^n| + |y^{n+1} - y^n| > \epsilon$ **do**
  Résoudre le système $J_\Phi(x^n, y^n).\begin{pmatrix} x^{n+1}-x^n \\ y^{n+1}-y^n \end{pmatrix} = -\begin{pmatrix} \Phi_1(x^n, y^n) \\ \Phi_2(x^n, y^n) \end{pmatrix}$
  pour trouver le point $X^{n+1} = \begin{pmatrix} x^{n+1} \\ y^{n+1} \end{pmatrix}$;
  $n = n + 1$;
  Remise à jour du critère d'arrêt;
end
$X = X^n$

Algorithme 2 : Algorithme de Newton pour $\Phi(x) = 0$ en dimension 2

Nous pouvons maintenant écrire les codes Matlab permettant de mettre en oeuvre la méthode de Newton.

Enfin, vous aurez besoin d'un éditeur de texte. Vous pouvez par exemple utiliser l'éditeur de `matlab` disponible dans File > Open > M-File. Il est également conseillé de créer un répertoire afin d'y enregistrer les divers fichiers que vous serez amené à créer au cours de ce TP (en créer au moins un nouveau pour chaque TP de ce module).

Pour commencer

1. Créer un répertoire TP1
2. Lancer `Matlab`

3. Ouvrir l'éditeur de texte

1.3. Mise en oeuvre en dimension 1

. Nous allons commencer par tester les méthodes sur cas simples. On va considérre le polynôme $f(x) = (x-1)(x^2+1)$ dont les racines sont $x = 1$,$x = i$, $x = -i$,

1. Ecrire une fonction matlab dont le nom est $f.m$ qui prend en argument un point x et qui ressort la valeur $y = f(x)$
2. Ecrire une fonction matlab dont le nom est $df.m$ qui prend en argument un point x et qui ressort la valeur $y = df(x)$ la valeur de la dérivé de la fonction f au point x
3. Ecrire une fonction newton dont les arguments sont f, df, x^0, ε qui renvoie, une solution approcheé de $f(x) = 0$, le nombre d'itérations effectuées et la liste des itérés calculés.
4. Ecrire un script $testnewton1d.m$ qui met en oeuvre la méthodes en dimension 1.
5. Tester la méthode pour différrantes valeurs de x^0 (complexe et réel) et de ε.
6. Conclusion.

1.4. Mise en oeuvre en dimension 2.

1.4.1. Résolution de $f(x) = 0$

Nous allons dans cet exemple simple déterminer les points d'intersection d'un cercle avec une hyperbole. On va rechercher les solutions du système d'équations

$$x^2 + y^2 = 2, \tag{3}$$
$$x^2 - y^2 = 1. \tag{4}$$

Bien sûr il s'agit d'un problème académique, puisqu'on connaît explicitement les solutions :

$$x_\star = \pm\sqrt{\frac{3}{2}} \;\; y_\star = \pm\frac{\sqrt{2}}{2}$$

Le but est d'étudier le comportement de la méthode de Newton pour cet exemple.

1. Ecrire une fonction matlab dont le nom est $f.m$ qui prend en argument un point x et qui ressort la valeur $y = f(x)$.
2. Pour cette exemple, pour quelles valeurs de (x, y) peut on définir la méthode de Newton.
3. Ecrire une fonction matlab dont le nom est $df.m$ qui prend en argument un point x et qui ressort la valeur $y = df(x)$ la valeur de la dérivé de la fonction f au point x

4. Ecrire une fonction newton dont les arguments sont f, df,$x0$,ε qui renvoie, une solution approcheé de $f(x) = 0$, le nombre d'itérations effectuées et la liste des itérés calculés.

5. Ecrire un script *testnewton2d.m* qui met en oeuvre la méthodes en dimension 2.

6. Tester le script. Pour traçer le cercle et l'hyperbole, on utilisera leur représentation sous forme paramétrique.
$$\cos(t)^2 + \sin(t)^2 = 1, \text{ et } \cosh(t)^2 - \sinh(t)^2 = 1.$$

7. Puisque nous connaissons les solutions exactes, calculer et traçer l'erreur effectuer à itération de l'algorithme de Newton à l'échelle logarithmique. Que pensez vous de son évolution? On notant $X^k = (x^k, y^k)$, l'erreur est :
$$\frac{\log |X^{k+1} - X_\star|}{\log |X^{k+1} - X_\star|}.$$

8. Effectuer des simulations pour différrentes valeurs de X^0 de la forme
$$X^0 = (2^{-n}, 1) \; n = 1..10$$
et pour chaque donnée initiale, stocker le nombre d'itérations effectuer dans la méthode de Newton. que pensez vous du résultat? (Voir question 2).

1.4.2. Résolution de $\nabla f(x) = 0$ avec Newton.

Dans cette partie, nous allons considérer le problème de minimisation :
$$\min_{x \in \mathbb{R}^n} (f(x))$$
avec f une fonction suffisamment régulière.
Nous savons que la solution du problème (quand elle existe!) vérifie $\nabla f(x) = 0$. Nous allons appliquer l'algorithme proposer dans la première partie. Cependant, cela n'assure pas que la solution numérique approche bien la solution recherchée $\overline{x}$ puisqu'un maximum de la fonction f a également un gradient nul.
La suite $(X)_k$ des itérées de l'algorithme de Newton vérifie :
$$X^{n+1} = X^n - [H_f(X^n)]^{-1} \nabla f(X^n)$$
où $H_f(X^n)$ est la matrice Hessienne de f évaluée au point X^n, $H_f(X^n) = \nabla^2 f(X^n)$. La méthode n'est donc pas définie si la matrice hessienne n'est pas inversible.
Rappel : Calculer $[H_f(X^n)]^{-1} \nabla f(X^n)$ revient à calculer d solution de $H_f(X)d = -\nabla f(X)$.

Data : f une fonction différentiable et X^0 un point initial
Result : X une approximation de la solution
$X = X^0$;
while *critère d'arrêt non satisfait* **do**
  Calculer d solution de $H_f(X)d = -\nabla f(X)$;
  X=X+d;
  Remise à jour de l'erreur pour le critère d'arrêt;
end

Algorithme 3 : Newton pour résoudre $\nabla f(x) = 0$

Nous allons nous intéresser au problème de minimisation de la fonction de Rosenbrock qui est fonction non-convexe :

$$f(x,y) = (1-x)^2 + 100(y-x^2)^2.$$

La principale difficulté de cette fonction est de capturer son minimum global au point $(1,1)$.

1. Créer un fichier rosen.m pour définir la fonction de Rosenbrock (argument d'entrée x et y 2 vecteurs et en sortie $z = f(x,y)$).

2. Nous allons commencer par tracer la fonction de Rosenbrock. Ecrire ce qui suit dans main2.m

   ```
   % courbes de niveau
   hold on;
   [xi, yi] = meshgrid(-2.2:0.1:2.2, -0.5:0.1:3);
   contour(xi,yi,rosen(xi,yi),30);
   plot3(x(1),x(2),rosen(x(1),x(2)),'b.');
   ```

   et taper main2 dans la console matlab. Regarder l'allure de la fonction autour du point $(1,1)$. Que voyez-vous?

3. Créer un fichier diffrosen.m et définir le Jacobien de la fonction de Rosenbrock (argument d'entrée x et y 2 vecteurs et en sortie $z = df(x,y)$ une matrice)

4. Créer un fichier hessrosen.m et définir la Hessienne de la fonction de Rosenbrock (argument d'entrée x et y 2 vecteurs et en sortie $z = d2f(x,y)$ une matrice)

5. Tester la méthode de Newton sur la fonction de Rosenbrock en l'appelant dans le main2.m avec comme point de départ le point $(-1.9, 2)$ et une erreur égale à 0.0000001. Fixer un nombre d'itération maximum. A l'aide des commandes tic et toc (help tic sur matlab) calculer le temps de calcul de l'approximation.

2. Méthodes de descentes

Soit $f : \mathbb{R}^n \to \mathbb{R}$ une fonction différentiable. Soient X,d dans $\mathbb{R}^n$. la direction d est une direction de descente en X si

$$d^T \nabla f(X) < 0.$$

La direction du gradient est celle dans laquelle la fonction a la plus forte pente. La direction opposée au gradient est donc celle dans laquelle la fonction a la plus forte descente.
L'algorithme de descente consiste simplement à suivre une direction de descente de façon itérative jusqu'à l'obtention d'un bon minimiseur. On utilise souvent plus d'un critère d'arrêt :

- un critère sur le déplacement, si $|X^{n+1} - X^n|$ est très petit, c'est qu'on ne progresse plus beaucoup.
- un critère sur la progression de l'objectif, si $|f(X^{n+1}) - f(X^n)|$ est très petit, on peut être presque arrivé à un minimum. On peut aussi mesurer la norme du gradient.
- un critère sur le temps de calcul ou le nombre d'itérations.

Data : f une fonction différentiable et X^0 un point initial
Result : X une approximation de la solution
$k = 0$;
while *critère d'arrêt non satisfait* **do**
  Trouver une direction de descente d_k telle que $d_k^T \nabla f(X^k) < 0$;
  Trouver un pas α_k telle que $f(X^k + \alpha_k d_k) < f(X^k)$;
  $X^{k+1} = X^k + \alpha_k d_k$;
  k=k+1;
  Remise à jour du critère d'arrêt;
end

Algorithme 4 : Algorithme de descente

2.1. Algorithme du gradient à pas fixe.

Soit $f : \mathbb{R}^n -> \mathbb{R} \in \mathcal{C}^1(\mathbb{R}^n, \mathbb{R})$, l'algorithme du gradient à pas fixe s'écrit : pour $\alpha > 0$ et $\varepsilon > 0$ une tolérence fixée

Data : f une fonction différentiable, x^0 un point initial et une tolérance ϵ.
Result : x une approximation de la solution
$n = 0$;
$\epsilon^0 = 2\varepsilon$;
while $\epsilon^n \geq \varepsilon$ **do**
  Calculer $w^n = -\nabla f(x^n)$;
  Calculer $x^{n+1} = x^n + \alpha w^n$ et calculer le résidu : $\epsilon^{n+1} = ||x^{n+1} - x^n||$;
  n=n+1;
end
$x = x^n$

Algorithme 5 : Algorithme du gradient à pas fixe

Nous verrons par la suite que l'algorithme ne converge pas toujours (voir cours résultat de convergence).

2.2. Algorithme du gradient à pas optimal.

L'idée de l'algorithme du gradient à pas optimal est d'essayer de calculer à chaque itération le paramètre α_k qui minimise la fonction f dans la direction de descente donnée par le gradient, c'est à dire que $x^{k+1} = x^k + \alpha_k w^k$, où w^k est une direction de descente en x^k et $\alpha_k > 0$.

Data : f une fonction différentiable, x^0 un point initial, $\alpha_0 > 0$ et une tolérance ϵ.
Result : x une approximation de la solution
$n = 0$;
$\epsilon^0 = 2\varepsilon$;
while $\epsilon^n \geq \varepsilon$ **do**
  Calculer $w^n = -\nabla f(x^n)$;
  Calculer $\alpha_n \geq 0$ tel que $f(x^n + \alpha_n w^n) \leq f(x^n + \alpha w^n), \ \forall \alpha \geq 0$;
  Calculer $x^{n+1} = x^n + \alpha_n w^n$;
  Calculer le résidu : $\epsilon^{n+1} = ||x^{n+1} - x^n||$;
  n=n+1;
end
$x = x^n$

Algorithme 6 : Algorithme du gradient à pas optimal

Pour cet algorithme, il n'est pas toujours évident de calculer le pas optimal α_n qui est solution

du problème de minimisation en dimension un. Pour cela nous introduisons, deux algorithmes de recherche linéaires

2.3. Recherche du pas optimal

2.3.1. Méthode de section dorée (Voir Cours poly de Y. privat et X.Antoine page 48)

Soit $q(\alpha)$ la fonction unidimensionnelle à minimiser comme par exemple

$$q(\alpha) = f(x^k + \alpha w^k).$$

Supposons que l'on connaisse un intervalle $[a, b]$ contenant le minimum α^* de q et tel que q soit décroissance sur $[a, \alpha^*]$ et croissante sur $]\alpha^*, b]$. On construit alors une suite décroissance d'intervalles $[a_k, b_k]$ qui contiennent tous le minimum α^*.
Il y a différentes possibilités pour le choix des points intérieurs. La méthode la plus utilisée est la méthode de la section d'orée. Pour éviter d'évaluer la fonction en deux nouveaux points à chaque itération, l'astuce est de s'arranger pour en réutiliser un d'une itération à l'autre.
Soit $[a_k, b_k]$ l'intervalle de recherche à l'instant k, on construit deux points intérieurs à $[a_k, b_k]$ de la manière suivante :

$$\begin{aligned} x_k^- &= \rho a_k + (1-\rho) b_k, \\ x_k^+ &= (1-\rho) a_k + \rho b_k \end{aligned}$$

avec ρ une constante positive tel que $0 < \rho < 1$. A l'itération suivante, si le minimum est à droite de x_k^-, on a $a_{k+1} = x_k^-$ et $b_{k+1} = b_k$. On souhaite réutiliser x_k^+, c'est à dire que l'on veut $x_{k+1}^- = x_k^+$. Cette dernière égalité s'écrit :

$$\rho a_{k+1} + (1-\rho) b_{k+1} = (1-\rho) a_k + \rho b_k.$$

En substituant $a_{k+1} = x_k^-$ et $b_{k+1} = b_k$, on obtient l'équation suivante sur ρ :

$$\rho^2 + \rho - 1 = 0$$

qui a pour solution positive $\rho = \frac{\sqrt{5}-1}{2}$.
L'algorithme de la section d'orée est

Data : q une fonction sur $[a, b]$, une tolérance ϵ et $\rho = \frac{\sqrt{5}-1}{2}$
Result : x une approximation de la solution
$x^- = \rho a + (1-\rho)b$;
$x^+ = a + b - x^-$;
$v^- = f(x^-)$;
$v^+ = f(x^+)$;
while $b - a \geq \epsilon$ **do**
  if $v^- \leq v^+$ **then**
    $b = x^+$;
    $x^+ = x^-$;
    $x^- = a + b - x^+$;
    $v^+ = v^-$;
    $v^- = f(x^-)$;
  else
    $a = x^-$;
    $x^- = x^+$;
    $x^+ = a + b - x^-$;
    $v^- = v^+$;
    $v^+ = f(x^+)$;
  end
end
$x = \frac{a+b}{2}$

Algorithme 7 : Algorithme de la section d'orée

2.3.2. Méthode de Wolfe (Voir Cours poly de Y. privat et X.Antoine page 52)

2.4. Méthode de quasi-Newton DFP et BFGS

Pour des problèmes de grandes dimensions, le calcul du hessien et de son inverse est trop coutêux dans le cas ou la fonction f n'est pas analytique. Le gradient quant à lui est toujours plus ou moins accessible (méthodes inverses). On souhaite donc ne pas calculer exactement l'inverse du hessien, mais simplement en évaluer une approximation. On peut alors utiliser des algorithmes, dits de quasi-Newton, qui calculent donc une approximation B_k de $\left[H_f(X^k)\right]^{-1}$ l'inverse de la matrice hessienne de f évaluée au point X^k.
Dans ce qui suit on note, $s_k = X_{k+1} - X_k$ et $y_k = \nabla f(X^{k+1}) - \nabla f(X^k)$ et on choisit une matrice B_0 définie positive (pour des raisons de convexité) : la matrice identité est habituellement choisie. On trouve principalement deux méthodes de calcul pour B_k:

- la formule de Davidon-Fletcher-Powell (DFP)

$$B_{k+1} = B_k + \frac{s_k s_k^T}{s_k^T y_k} - \frac{B_k y_k y_k^T B_k}{y_k^T B_k y_k}.$$

- la formule de Broyden-Fletcher-Goldfarb-Shanno (BFGS)

$$B_{k+1} = B_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k}.$$

3. Tavail à effectuer

1. Créer un fichier descente1.m et écrire l'algorithme de la descente à pas fixe et tester cette fonction sur la fonction de Rosenbrock en l'appelant dans le main.m avec comme point de départ le point $(-1.9, 2)$ et une erreur égale à 0.0000001. Fixer un nombre d'itération maximum. A l'aide des commandes tic et toc (help tic sur matlab) calculer le temps de calcul de l'approximation.

2. Ecrire une fonction gs.m (gold section) qui effectue la recherche linéaire avec la méthode de la section d'orée : Tester ce code sur la fonction $f(x) = exp(x) - 2x$.

```
function y = fun(x)
%
y = exp(x) - 2*x;
```

3. Ecrire un fichier wolfe.m qui effectue la recherche linéraire par la méthode de Wolfe et tester wolfe.m sur la fonction $f(x) = exp(x) - 2x$ et comparer le temps de calcul entre la section d'orée et Wolfe.

4. Créer un fichier descente2.m et écrire l'algorithme de la descente à pas optimal et tester cette fonction sur la fonction de Rosenbrock en l'appelant dans le main.m avec comme point de départ le point $(-1.9, 2)$ et une erreur égale à 0.0000001. Fixer un nombre d'itération maximum. A l'aide des commandes tic et toc (help tic sur matlab) calculer le temps de calcul de l'approximation. **Pour la recherche linéraire, on comparera les résultats de convergence avec Wolfe et la section d'orée.**
   Visualiser les déplacements des itérés sur une carte de la fonction (en utilisant la fonction contour2d)
   Estimer les vitesses de convergence asymptotique en traçant par exemple la suite des valeurs $\log(f(X^k))$

5. Ecrire un fichier dfp.m qui effectue l'algorithme de Davidson-Fletcher-Powell et qui utilise la méthode de Wolfe pour la recherche linèraire (page 62 poly Y. Privat) et tester cette fonction sur la fonction de Rosenbrock en l'appelant dans le main.m avec comme point de départ le point $(-1.9, 2)$ et une erreur égale à 0.0000001. On prendra la matrice identité (commande eye en matlab) pour la première matrice et un pas initial égal à 1 en entrée de la recherche linéeaire de Wolfe.

   Fixer un nombre d'itération maximum. Calculer le temps de calcul de l'approximation.
   Visualiser les déplacements des itérés sur une carte de la fonction (en utilisant la fonction contour2d)
   Estimer les vitesses de convergence asymptotique en traçant par exemple la suite des valeurs $\log(f(X^k))$

6. Ecrire un fichier bfgs.m qui effectue l'algorithme de BFGS et qui utilise la méthode de Wolfe pour la recherche linèraire (page 62 poly Y. Privat) et tester cette fonction sur la fonction de Rosenbrock en l'appelant dans le main.m avec comme point de départ le point $(-1.9, 2)$ et une erreur égale à 0.0000001. On prendra la matrice identité (commande eye en matlab) pour la première matrice et un pas initial égal à 1 en entrée de la recherche lin?eaire de Wolfe.

   Fixer un nombre d'itération maximum. A l'aide des commandes tic et toc (help tic sur matlab) calculer le temps de calcul de l'approximation.
   Visualiser les déplacements des itérés sur une carte de la fonction (en utilisant la fonction

contour2d)
Estimer les vitesses de convergence asymptotique en traçant par exemple la suite des valeurs $\log(f(X^k))$

7. Comparer toutes les méthodes et conclure.