BE0 -Méthodes d'optimisation sans contrainte

Dans ce bureau d'étude, nous nous intéressons à la résolution de problèmes d'optimisation sans contrainte. L'objectif de ce BE est de réaliser une implémentation de plusieurs algorithmes d'optimisation itératifs , et d'évaluer leurs performances pour la minimisation des fonctions suivantes :

- La fonction Good : $J_1(x,y) = \frac{1}{2}x^2 + \frac{1}{4}x^4 + \frac{1}{2}y^2 + \frac{1}{4}y^4$.
- La fonction Bad : $J_2(x,y) = 100x^4 + 0.01y^4$.
- La fonction Hard : $J_3(x,y) = \sqrt{1+x^2} + \sqrt{1+y^2}$.
- La fonction Rosenbrock : $J_4(x,y) = 100(y-x^2)^2 + (1-x)^2$.

Question 1 *Les fonctions J_1, J_2, J_3 et J_4 admettent-elles des minima sur $\mathbb{R}^2$? Si oui, les déterminer.*

Pour la réalisation de ce BE, on vous fournit les fichiers python suivants :

- le fichier `BE0_algo.py` qui regroupe les définitions des algorithmes à implémenter. Il est à compléter
- le fichier `BE0_noyau.py` qui gère les paramètres des algorithmes d'optimisation ainsi que l'affichage et rassemble la définition des fonctions J_1, J_2, et J_3 avec leur gradient et leur matrice hessienne. Il n'y a rien à modifier dans ce fichier.
- le fichier `BE0_main.py` qui permet de choisir le problème à résoudre et d'appeler l'algorithme à utiliser.

L'idée des méthodes de descente est de remplacer la fonction f par un modèle local plus simple, dont la minimisation nous donnera une direction de descente. Soit $\mathbf{x}_k \in \mathbb{R}^P$ l'itéré courant, on remplace J au voisinage de $\mathbf{x}_k$ par son développement de Taylor au premier ordre :

$$J(\mathbf{x}_k + \mathbf{d}) \approx J(\mathbf{x}_k) + \nabla J(\mathbf{x}_k)^T \mathbf{d}.$$

Une direction de descente possible est la direction de plus profonde descente définie par :

$$\mathbf{d}_k = -\nabla J(\mathbf{x}_k).$$

Le choix de la direction de plus profonde descente définit une famille d'algorithmes appelés algorithmes de descente de gradient, dont un possible schéma est le suivant :

Algorithme de Gradient (avec recherche linéaire)

1: **Initialisation :** $\mathbf{x}_0 \in \mathbb{R}^P$, $\alpha > 0$, $c \in]0,1[$, et $\beta \in]0,1[$.
2: **Étape courante :** $\mathbf{x}_k$ connu
3: Direction de descente : $\mathbf{d}_k = -\nabla J(\mathbf{x}_k)$.
4: Recherche linéaire : choisir un pas $\alpha_k = \alpha\beta^{i_k}$ où i_k est le plus petit entier tel que :

$$J(\mathbf{x}_k + \alpha_k \mathbf{d}_k) \leq J(\mathbf{x}_k) + c\alpha_k \nabla J(\mathbf{x}_k)^T \mathbf{d}_k$$

5: Mise à jour : $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$.
6: Incrémenter l'indice k
7: **Test d'arrêt.**

Dans le cadre de ce BE, on choisit $\alpha = 1$, $\beta = 0.5$ et $c = 0.5$.

Question 2 *Expliciter deux critères d'arrêt convenables (autres que le nombre maximum d'évaluations de la fonction objectif ou d'itérations). Implémenter l'algorithme de gradient avec recherche linéaire.*

Commenter les résultats obtenus.

Pour construire les méthodes de gradient, nous avons remplacé J par son approximation linéaire au voisinage de l'itéré courant. Ces méthodes ne sont pas très performantes, parce qu'elles ne prennent pas en compte la courbure de la fonction (convexité, concavité...) qui est une information au second ordre. Le principe de la méthode de Newton est le suivant : supposons que J est de classe C^2 et remplaçons J au voisinage de l'itéré courant $\mathbf{x}_k$ par son développement de Taylor au second ordre :

$$J(\mathbf{y}) \approx q(\mathbf{y}) = J(\mathbf{x}_k) + \nabla J(\mathbf{x}_k)^T(\mathbf{y} - \mathbf{x}_k) + \frac{1}{2}(\mathbf{y} - \mathbf{x}_k)^T \nabla^2 J(\mathbf{x}_k)(\mathbf{y} - \mathbf{x}_k).$$

On choisit alors comme point $\mathbf{x}_{k+1}$ le minimum de la quadratique q lorsque cela est possible et unique, ce qui n'est le cas que si $\nabla^2 J(\mathbf{x}_k)$ est définie positive. Or le minimum de q est réalisé par $\mathbf{x}_{k+1}$ solution de :

$$\nabla^2 J(\mathbf{x}_k)(\mathbf{x}_{k+1} - \mathbf{x}_k) = -\nabla J(\mathbf{x}_k),$$

soit :

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [\nabla^2 J(\mathbf{x}_k)]^{-1} \nabla J(\mathbf{x}_k).$$

Algorithme de Newton (basique)

1: **Initialisation :** $\mathbf{x}_0 \in \mathbb{R}^P$, $\alpha > 0$, $c \in]0, 1[$, et $\beta \in]0, 1[$.
2: **Étape courante :** $\mathbf{x}_k$ connu
3: Direction de descente : calculer $\mathbf{d}_k$ solution du système : $\nabla^2 J(\mathbf{x}_k)\mathbf{d}_k = -\nabla J(\mathbf{x}_k)$.
4: Choisir un pas α_k (par exemple, $\alpha_k = 1$).
5: Mise à jour : $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$.
6: Incrémenter l'indice k
7: **Test d'arrêt.**

Question 3 *Implémenter l'algorithme de Newton basique. Commenter les résultats obtenus.*

Malgré sa convergence locale rapide, la méthode de Newton n'est pas globalement convergente (c'est-à-dire que pour certains points de départ, la méthode de Newton risque de diverger). Pour éviter cet inconvénient, on propose dans un premier temps d'ajouter une étape de recherche linéaire.

Algorithme de Newton (avec recherche linéaire)

1: **Initialisation :** $\mathbf{x}_0 \in \mathbb{R}^P$, $\alpha > 0$, $c \in]0, 1[$, et $\beta \in]0, 1[$.
2: **Étape courante :** $\mathbf{x}_k$ connu
3: Direction de descente : calculer $\mathbf{d}_k$ solution du système : $\nabla^2 J(\mathbf{x}_k)\mathbf{d}_k = -\nabla J(\mathbf{x}_k)$.
4: Recherche linéaire : choisir un pas $\alpha_k = \alpha\beta^{i_k}$ où i_k est le plus petit entier tel que :

$$J(\mathbf{x}_k + \alpha_k \mathbf{d}_k) \leq J(\mathbf{x}_k) + c\alpha_k \nabla J(\mathbf{x}_k)^T \mathbf{d}_k$$

5: Mise à jour : $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$.
6: Incrémenter l'indice k
7: **Test d'arrêt.**

Question 4 *Implémenter l'algorithme de Newton avec recherche linéaire. Commenter les résultats obtenus.*

Pour que la méthode de Newton (basique ou avec recherche linéaire) soit bien définie à chaque itération, il faut que la direction $\mathbf{d}_k$ soit une direction de descente pour chaque point $\mathbf{x}_k$ visité par l'algorithme (c'est-à-dire que $\nabla J(\mathbf{x}_k)^T \mathbf{d}_k < 0$). Pour cela, il suffit que la matrice hessienne $\nabla^2 J(\mathbf{x}_k)$ soit définie positive, ce qui n'est pas forcément vrai lorsque $\mathbf{x}_k$ se situe loin de la solution recherchée. Ce problème peut être contourné en se dirigeant dans la direction du gradient lorsque la direction de Newton n'est pas une direction de descente.

Algorithme de Gradient-Newton (avec recherche linéaire)

1: **Initialisation :** $\mathbf{x}_0 \in \mathbb{R}^P$, $\alpha > 0$, $c \in]0,1[$, et $\beta \in]0,1[$.
2: **Étape courante :** $\mathbf{x}_k$ connu
3: Direction de descente : soit $\bar{\mathbf{d}}_k$ solution du système : $\nabla^2 J(\mathbf{x}_k)\bar{\mathbf{d}}_k = -\nabla J(\mathbf{x}_k)$.
Si $\nabla J(\mathbf{x}_k)^T \bar{\mathbf{d}}_k < 0$ (c'est-à-dire que $\bar{\mathbf{d}}_k$ est une direction de descente), alors $\mathbf{d}_k = \bar{\mathbf{d}}_k$.
Sinon, poser $\mathbf{d}_k = -\nabla J(\mathbf{x}_k)$.
4: Recherche linéaire : choisir un pas $\alpha_k = \alpha\beta^{i_k}$ où i_k est le plus petit entier tel que :

$$J(\mathbf{x}_k + \alpha_k \mathbf{d}_k) \leq J(\mathbf{x}_k) + c\alpha_k \nabla J(\mathbf{x}_k)^T \mathbf{d}_k$$

5: Mise à jour : $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$.
6: Incrémenter l'indice k
7: **Test d'arrêt.**

Question 5 *Implémenter l'algorithme de Gradient-Newton avec recherche linéaire. Commenter les résultats obtenus.*

Afin d'éviter le calcul de la matrice hessienne, les méthodes de Quasi-Newton utilisent une approximation de cette matrice comme suit : pour une fonction quadratique, il est aisé de démontrer que

$$\nabla J(\mathbf{x}_{k+1}) - \nabla J(\mathbf{x}_k) = \nabla^2 J(\mathbf{x}_{k+1})(\mathbf{x}_{k+1} - \mathbf{x}_k)$$

Cela indique que la connaissance de deux vecteurs distincts $\mathbf{x}_{k+1}$ et $\mathbf{x}_k$ et de la différence de gradient associée permet d'obtenir, au voisinage de la solution, une approximation de $\nabla^2 J(\mathbf{x}_{k+1})$. Plus généralement, on suppose connus $\mathbf{z}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$ et $\mathbf{y}_k = \nabla J(\mathbf{x}_{k+1}) - \nabla J(\mathbf{x}_k)$, ainsi qu'une approximation courante $\mathbf{H}_k$ de la Hessienne. On cherche une nouvelle approximation $\hat{\mathbf{H}}$, telle que $\hat{\mathbf{H}}$ soit symétrique et que $\hat{\mathbf{H}}\mathbf{z}_k = \mathbf{y}_k$. Cela ne suffit pas pour définir de manière unique $\hat{\mathbf{H}}$, et on peut rechercher un $\hat{\mathbf{H}}$ de norme minimale (pour certaines normes) pour forcer l'unicité.

Algorithme de Quasi-Newton dit "de BFGS"(avec recherche linéaire)

1: **Initialisation :** $\mathbf{x}_0 \in \mathbb{R}^P$, $\alpha > 0$, $c \in]0,1[$, et $\beta \in]0,1[$, et une approximation initiale de l'inverse de la matrice hessienne B_0 en $\mathbf{x}_0$.
2: **Étape courante :** $\mathbf{x}_k$ et $\mathbf{B}_k$ connus
3: Calculer $\mathbf{d}_k$: $\mathbf{d}_k = -\mathbf{B}_k \nabla J(\mathbf{x}_k)$.
4: Recherche linéaire : choisir un pas $\alpha_k = \alpha\beta^{i_k}$ où i_k est le plus petit entier tel que :

$$J(\mathbf{x}_k + \alpha_k \mathbf{d}_k) \leq J(\mathbf{x}_k) + c\alpha_k \nabla J(\mathbf{x}_k)^T \mathbf{d}_k$$

5: Mise à jour : $\mathbf{x}_{k+1} = \mathbf{x}_k + s_k \mathbf{d}_k$.
6: Mise à jour de $\mathbf{y}_k$ et $\mathbf{z}_k$: $\mathbf{y}_k = \nabla J(\mathbf{x}_{k+1}) - \nabla J(\mathbf{x}_k)$, $\mathbf{z}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$ et $\rho_k = 1/(\mathbf{y}_k^T \mathbf{z}_k)$.
7: Mise à jour de $\mathbf{B}_k$: $\mathbf{B}_{k+1} = (\mathbf{I}_P - \rho_k \mathbf{z}_k \mathbf{y}_k^T)\mathbf{B}_k(\mathbf{I}_P - \rho_k \mathbf{y}_k \mathbf{z}_k^T) + \rho_k \mathbf{z}_k \mathbf{z}_k^T$.
8: Incrémenter l'indice k.
9: **Test d'arrêt.**

Un exemple d'algorithme de Quasi-Newton est l'Algorithme de BFGS (Broyden, Fletcher, Goldfarb, Shanno) qui approxime directement l'inverse de la matrice hessienne afin d'éviter la résolution d'un sytème linéaire en plus du calcul des dérivées secondes. En effet, nous avons vu que dans la méthode de Newton, il s'agit de résoudre des systèmes linéaires de la forme $\nabla^2 J(\mathbf{x}_k)\mathbf{d}_k = -\nabla J(\mathbf{x}_k)$. À chaque itération k de l'algorithme de BFGS, on recherche une nouvelle matrice $\mathbf{B}_{k+1}$, symétrique, supposée mieux approcher que $\mathbf{B}_k$ l'inverse de la Hessienne en $\mathbf{x}_{k+1}$ (pour une certaine norme)

$$\mathbf{B}_{k+1} = \underset{\substack{\mathbf{B}=\mathbf{B}^T \\ \mathbf{B}\mathbf{y}_k = \mathbf{z}_k}}{\operatorname{argmin}} \|\mathbf{B} - \mathbf{B}_k\|^2$$

La solution de ce problème est donnée par la formule :

$$\mathbf{B}_{k+1} = (\mathbf{I}_P - \rho_k \mathbf{z}_k \mathbf{y}_k^T)\mathbf{B}_k(\mathbf{I}_P - \rho_k \mathbf{y}_k \mathbf{z}_k^T) + \rho_k \mathbf{z}_k \mathbf{z}_k^T,$$

avec $\rho_k = 1/(\mathbf{y}_k^T \mathbf{z}_k)$ et $\mathbf{I}_P$ la matrice identité d'ordre P.

Question 6 *Implémenter l'algorithme de Quasi-Newton avec recherche linéaire. Commenter les résultats obtenus.*