TP n°3

Minimisation d'une fonction de 2 variables, avec et sans contraintes.

Ce TP, à réaliser en binôme, est noté. Travail à rendre 1 semaine après la séance encadrée. L'évaluation de ce travail constituera la note de TP (10% de la note finale de l'UE).

Attention ! Ce TP est trop long pour être fait uniquement pendant la séance de TP3. Il faut commencer à travailler sur les deux premières parties à l'issue du TP2 et sur la troisième partie à l'issue du cours n°3.

Objectifs du TP :

- Consolider les acquis des TP1 et TP2.
- Utiliser les méthodes `root` et `minimize` de la bibliothèque `scipy.optimize`
- Mettre en oeuvre la méthode du gradient à pas fixe et la méthode de Newton
- Appliquer la méthode des multiplicateurs de Lagrange pour déterminer les points critiques d'un problème de minimisation sous contraintes-égalités

Consignes :

- Travail à rendre : le code python d'une part, et la solution rédigée dans un notebook d'autre part.
- Il est conseillé de développer, mettre au point et tester les codes sous spyder ou tout autre environnement de développement, puis d'importer les fonctions développées dans le notebook pour rédiger la solution et présenter les résultats.
- Les dérivées sont calculées analytiquement « à la main » (pas de calcul symbolique, ni de calcul numérique par différences finies). Ces calculs peuvent être rédigés dans le notebook ou sur feuille puis scannés, sous réserve que ce soit facilement lisible.
- Si votre travail comporte une partie sur feuille et une partie sur notebook, veillez à ce que les renvois d'un document à l'autre soient clairs. L'évaluation de votre travail nécessite que celui-ci soit compréhensible.

Enoncé :

Soit la fonction $J(x_1, x_2) = {x_1}^2 + 1{,}5.{x_2}^2 - 3.\sin(2x_1 + x_2) + 5.\sin(x_1 - x_2)$.

1. Première partie :

1.1. Tracer les isovaleurs de la fonction J de façon à mettre en évidence ses points critiques et leur nature. Pour cela, vous devez rechercher un cadrage (domaine de tracé) et un nombre d'isovaleurs adaptés.

1.2. Utiliser la fonction `root` de la bibliothèque `scipy.optimize` afin de déterminer les points critiques de J, puis la fonction `eigvals` de la bibliothèque `numpy.linalg` afin de déterminer leur nature. Comment faire pour trouver tous les points critiques ?

1.3. Utiliser la fonction `minimize` de la bibliothèque `scipy.optimize` afin de déterminer les minima de J. Retrouvez-vous les points de la question précédente ?

2. Deuxième partie :

2.1. Programmer la méthode du gradient à pas fixe pour déterminer le minimum de la fonction J (voir rappels en annexe).

Consignes :

- L'algorithme du gradient est programmé dans une fonction dont les paramètres d'entrée sont le nom de la fonction qui calcule J, le nom de la fonction qui calcule son gradient, le point de départ de l'algorithme, la valeur du pas α, la précision souhaitée ε et le nombre maximal d'itérations autorisées n_{max}. Cette fonction renvoie la suite de points (X_n) ainsi qu'un indicateur de convergence.

- L'affichage des résultats est fait dans le programme principal. Les isovaleurs de J sont tracées, de même que la suite de segments $[X_{n-1}, X_n]$ (voir slide 32 du cours n°2). Le programme indique à l'utilisateur si la recherche de minimum a convergé, le nombre d'itérations réalisées et le dernier point obtenu (solution approchée).

2.2. Dans un premier temps, vérifier que le mécanisme itératif est correctement implanté en exécutant 2 ou 3 itérations et avec une valeur de α petite. Si cela fonctionne et que les résultats sont cohérents, vous pouvez augmenter le nombre d'itérations, la valeur de α et tester la convergence de l'algorithme

2.3. Tester l'algorithme pour différente valeurs de α et $\varepsilon = 10^{-3}$. Mettre en évidence le fait que l'algorithme diverge si α dépasse une certaine valeur.

2.4. Tester l'algorithme pour différents points de départ. Retrouvez-vous les résultats de la première partie ?

2.5. Proposer et programmer un mécanisme qui ajuste si besoin la valeur de α donnée par l'utilisateur afin d'assurer qu'on a toujours $J(X_{n+1}) < J(X_n)$. On est alors sûr d'assurer la convergence de l'algorithme.

2.6. Programmer et tester la méthode de Newton. Comparer le comportement de la descente de gradient et de la méthode de Newton pour $\varepsilon = 10^{-8}$ pour différents points de départ et en particulier les points de départ suivants : $X_{ini} = [-0{,}5\ ;\ 1{,}5]$ et $X_{ini} = [1{,}0\ ;\ 1{,}5]$.

3. Troisième partie :

On veut maintenant minimiser $J(M)$ sous contrainte que le point M appartient à la droite Δ passant par les points $A(0{,}0)$ et $B(2, -3)$.

3.1. Tracer les isovaleurs de J et superposer la droite Δ. Estimer visuellement la position des extremums de la restriction de J à la droite Δ.

3.2. Appliquer la méthode des multiplicateurs de Lagrange pour résoudre le problème. Le système d'équations obtenu sera résolu numériquement par la fonction `scipy.optimize.root`. Faire une figure représentant le problème, la solution obtenue et les éléments géométriques remarquables.

3.3. Résoudre à nouveau le problème, mais pour la droite Δ' passant par les points $A(0{,}1)$ et $B(2, -2)$.

Rappel 1 : principe de la méthode du gradient

Le problème posé est de minimiser une certaine fonction $J(X)$ définie de $\mathbb{R}^n$ dans $\mathbb{R}$. La variable X est une vecteur : $X = (x_1, \dots, x_n) \in \mathbb{R}^n$. La fonction J est supposée différentiable à l'ordre 1[1], et on suppose que l'on connait l'expression analytique du gradient.

De manière générale, les méthodes de descente sont des méthodes itératives qui génèrent une suite de points X_n tels que : $\forall n \geq 0 : J(X_{n+1}) < J(X_n)$. Au point X_n, la direction dans laquelle la fonction décroît le plus vite est la direction opposée au gradient de J. Une technique simple pour réduire J consiste donc à rechercher un nouveau point en se déplaçant d'une certaine quantité dans cette direction.

Le schéma général de la méthode du gradient est le suivant :

$$X_{n+1} = X_n - \alpha . \nabla J(X_n) \qquad \text{avec} \quad \alpha > 0 \text{ choisi pour avoir : } J(X_{n+1}) < J(X_n)$$

Gradient à pas optimal :

La méthode dite « à pas optimal » consiste à déterminer à chaque itération la valeur de α qui minimise J dans la direction du gradient. Il s'agit donc de minimiser $G(\alpha) = J[X_n - \alpha . \nabla J(X_n)]$ par rapport à la variable α, ce qui ramène à un problème de minimisation à une variable. On peut également formuler ce problème comme un problème d'optimisation sous contrainte égalité : il faut minimiser $J(X)$, sous contrainte $X = X_n - \alpha . \nabla J(X_n)$. Cette méthode est conceptuellement satisfaisante, mais en pratique elle n'est pas vraiment efficace. En effet, les directions de recherche utilisées au cours de l'algorithme à pas optimal sont deux à deux orthogonales. Elles sont donc entièrement déterminées par le point initial et ne sont pas nécessairement adaptées au relief de la fonction dans le voisinage du minimum. Par ailleurs, la minimisation de J dans une direction à chaque itération a un coût de calcul qui ne peut être négligé. Pour cette raison, on préfère d'autres techniques comme le gradient à pas fixe, le gradient à pas adaptatif, ou les gradients conjugués. Dans ce TP, vous programmerez la méthode du gradient à pas fixe, puis vous proposerez une règle empirique pour faire du pas adaptatif.

Gradient à pas fixe :

La méthode dite « à pas fixe » consiste à prendre une valeur de α constante au cours des itérations. Le pas de déplacement étant proportionnel au module du gradient, il diminue automatiquement au voisinage du minimum et on peut espérer converger vers la solution, à condition de choisir une valeur de α correcte. En effet, si α est « trop grand », on peut osciller autour du minimum sans jamais l'atteindre. A l'inverse, si α est « trop petit », on est sûr de converger vers le minimum, mais avec un nombre d'itérations inutilement grand. En pratique, à moins que l'on dispose d'informations sur la dérivée seconde de la fonction, la détermination de α est empirique (essais-erreurs). Pour un problème donné, on teste différentes valeurs de α jusqu'à trouver quelque chose de satisfaisant.

Schéma général de l'algorithme du gradient :

- Notations :
  - X_n et X_{n+1} : point initial et point final de l'itération n.
  - dX : pas de déplacement pour une itération
  - n et n_{max} : compteur d'itérations et nombre maximal d'itérations autorisé
  - X_0 : point de départ de l'algorithme
  - α : pas de recherche
  - ε : critère de précision
  - $converge$: indicateur booléen de convergence
- Algorithme :
  - Choix des paramètres de l'algorithme : $X_0, \alpha, \varepsilon, n_{max}$

[1] C'est-à-dire que le vecteur gradient est défini en tous points

- Initialisation : $X_n \leftarrow X_0$, $dX \leftarrow 1, n \leftarrow 0$
- Tant que $dX > \varepsilon$ et $n < n_{max}$:
  - $X_{n+1} \leftarrow X_n - \alpha . \nabla J(X_n)$
  - $dX \leftarrow \|X_{n+1} - X_n\|$
  - $X_n \leftarrow X_{n+1}$
  - $n \leftarrow n + 1$
- $converge \leftarrow (dX \leq \varepsilon)$

Rappel 2 : principe de la méthode de Newton

On souhaite trouver le minimum de la fonction J, supposée convexe sur $\mathbb{R}$. On suppose que l'on dispose des expressions analytiques du gradient ∇J et de la matrice hessienne HJ.

Soit X_n, une solution approchée. La fonction est approximée par $\tilde{J}_n$ son développement limité d'ordre 2 en X_n :

$$\widetilde{J_n}(X) = J(X_n) + \nabla J(X_n)^T . (X - X_n) + \frac{1}{2}(X - X_n)^T . HJ(X_n) . (X - X_n)$$

La nouvelle solution approchée X_{n+1} est le minimum de $\widetilde{J_n}$, obtenu en annulant le gradient de $\widetilde{J_n}$.

$$\nabla\widetilde{J_n}(X) = \nabla J(X_n) + HJ(X_n) . (X - X_n)$$

$$\nabla\widetilde{J_n}(X_{n+1}) = 0 \iff \nabla J(X_n) + HJ(X_n) . (X_{n+1} - X_n) = 0$$

d'où : $X_{n+1} = X_n + \Delta X$, où ΔX est solution du système linéaire : $HJ(X_n) . \Delta X = -\nabla J(X_n)$

Algorithme :

- Choix des paramètres de l'algorithme : $X_0, \varepsilon, n_{max}$
- Initialisation : $X_n \leftarrow X_0$, $dX \leftarrow 1, n \leftarrow 0$
- Tant que $dX > \varepsilon$ et $n < n_{max}$:
  - Calculer $\nabla J(X_n)$
  - Calculer $HJ(X_n)$
  - $\Delta X \leftarrow$ solution du système matriciel $HJ(X_n) . \Delta X = -\nabla J(X_n)$
  - $X_n \leftarrow X_n + \Delta X$
  - $dX \leftarrow \|\Delta X\|$
  - $n \leftarrow n + 1$
- $converge \leftarrow (dX \leq \varepsilon)$