Valentin De Bortoli debortoli@cmla.ens-cachan.fr
Alain Durmus alain.durmus@cmla.ens-cachan.fr
TP 1 : optimisation numérique Première année Master Hadamard

Méthode de descentes de gradient et algorithmes de Newton

En préambule on suppose que les paquets suivants ont été chargés

```
import numpy as np
import numpy.random as rnd
import matplotlib.pyplot as plt
import scipy
```

Remarque : la plupart du temps on n'implémente pas les méthodes classiques présentées dans ce TP et on utilise des paquets du type `scipy`. Néanmoins, il convient de savoir ce que contienne ces *boîtes noires* et de se forger une intuition sur les différentes méthodes d'optimisation.

1 Méthode de gradient à pas fixe

But : Le but de cet exercice est d'implémenter la descente de gradient à pas fixe et d'identifier ses atouts et limites.

Banane de Rosenbrock Dans notre première exemple, on étudie la fonction $J : \mathbb{R}^2 \to \mathbb{R}$ suivante

$$J(x_1, x_2) = (x_1 - 1)^2 + 100\,(x_1^2 - x_2)^2.$$

J est de classe $\mathcal{C}^1(\mathbb{R}^2, \mathbb{R})$, positive, atteint son unique minimum en $x^* = (1, 1)$ et $J(x^*) = 0$.

1. Tracer les lignes de niveau de J dans le rectangle $(x_1, x_2) \in [-1.5,\ 1.5] \times [-0.5,\ 1.5]$.

2. Implémenter la méthode de gradient à pas fixe

$$\forall k \in \mathbb{N},\ x^{k+1} = x^k - \alpha\,\nabla J(x^k).$$

On pourra considérer $x^0 = (-1, 1)$ et $\alpha = 2 \times 10^{-3}$. Tracer graphiquement les itérées de la méthode de gradient à pas fixe. Afficher le nombre d'itérations nécessaire pour atteindre le critère d'arrêt suivant $J(x^{k+1}) - J(x^*) < 10^{-4}$.

3. Recommencer avec $\alpha = 4.5 \times 10^{-3}$ puis avec $\alpha = 10^{-2}$. Interpréter les résultats.

4. Recommencer avec le gradient normalisé

$$x^{k+1} = x^k - \alpha\,\frac{\nabla J(x^k)}{\|\nabla J(x^k)\|}$$

avec $\alpha = 5 \times 10^{-2}$ et en affichant les 200 premières itérations. Interpréter.

Problème quadratique On introduit la fonction f définie sur $\mathbb{R}^n$ ($n \in \mathbb{N}$ avec $n \geq 2$) par $f(x) = x^T A x$ avec A diagonale telle que $A(1,1) = M$, $A(2,2) = m$ avec $(M, m) \in {\mathbb{R}_+^*}^2$ et tous les autres coefficients diagonaux fixés à 1. f est une fonction strictement convexe coercive qui possède donc un unique minimum en 0.

1. Représenter les lignes de niveaux de la fonction f pour différentes valeurs de M et de m. À votre avis, quand est-ce que le problème d'optimisation est le plus facile à résoudre ?

2. Tester l'algorithme du gradient à pas fixe pour ce problème. Le critère d'arrêt est le même que pour la partie précédente.

3. Vérifier expérimentalement que la vitesse de l'algorithme ne dépend pas de la taille de la matrice A.

2 Choix du pas de descente : règle d'Armijo et règle de Wolfe

But : Pour une direction de descente d^k, i.e $\langle d^k, \nabla J(x^k)\rangle < 0$, la *recherche linéaire exacte* consiste à estimer un pas de descente optimal α^* qui vérifie

$$\alpha* = \text{argmin}_{\alpha\geq 0} J(x^k + \alpha d^k),$$

introduisant donc un nouveau problème d'optimisation qui peut être aussi difficile que le problème de minimisation original. On introduit ici deux règles de *recherche linéaire inexacte*, la règle d'Armijo (1966) et la règle de Wolfe (1969).

Règle d'Armijo Soit $\beta \in]0,1[$, $c \in]0, \frac{1}{2}[$ et $L \in \mathbb{R}_+$ la constante de Lipschitz de J restreinte à K un compact.

1. *Proposition* $\alpha_t = -\frac{\langle \nabla J(x^k), d^k\rangle}{L\|d^k\|^2}$,
2. *Test d'acceptation* Si $J(x^k + \alpha_t d^k) \leq J(x^k) + c\,\alpha_t \langle \nabla J(x^k), d^k\rangle$, alors $\alpha^k = \alpha_t$ sinon on reprend le test avec $\beta\alpha_t$.

La règle d'Armijo converge en un nombre fini d'étapes.

Règle de Wolfe Soit $c_1 \in]0, \frac{1}{2}[$, $c_2 \in]0, \frac{1}{2}[$, et $L \in \mathbb{R}_+$ la constante de Lipschitz de J restreinte à K un compact. Soit $(m_0, M_0) \in \mathbb{R}_+^2$. On fixe également un nombre maximal d'itérations dans la règle de Wolfe (10^3 dans notre cas).

1. *Proposition* $\alpha_t = -\frac{\langle \nabla J(x^k), d^k\rangle}{L\|d^k\|^2}$,
2. *Test d'acceptation (1)* Si $J(x^k + \alpha_t d^k) \leq J(x^k) + m\,\alpha_t \langle \nabla J(x^k), d^k\rangle$, **et** également ${d^k}^T \nabla J(x^k + \alpha_t d^k) \leq -c_2 {d^k}^T \nabla J(x^k)$ alors $\alpha^k = \alpha_t$.
3. *Test d'acceptation (2)* Si $J(x^k + \alpha_t d^k) > J(x^k) + m\,\alpha_t \langle \nabla J(x^k), d^k\rangle$, alors on pose $\alpha_t = \frac{m_0+M_0}{2}$ ainsi que $M_0 = \alpha_t$ et on reprend le premier test d'acceptation.
4. *Test d'acceptation (3)* Si aucune de ces conditions n'est remplie alors on peut poser $\alpha_t = \frac{m_0+M_0}{2}$ et $m_0 = \alpha_t$ et on reprend le premier test d'acceptation.

Banane de Rosenbrock et problème quadratique On considère les deux problèmes d'optimisation introduits dans l'exercice précédent.

1. Implémenter la méthode de gradient avec la règle d'Armijo et règle de Wolfe.

2. Tester l'algorithme pour la fonction de Rosenbrock avec les valeurs suivantes $\beta = \frac{1}{2}$, $m = 4 \times 10^{-1}$ et $L = 10^2$ et $x^0 = (-1, 1)$. Comme pour le premier exercice, afficher le

nombre d'itérations nécessaire pour atteindre le critère d'arrêt suivant $J(x^{k+1}) - J(x^*) < 10^{-4}$. Conclure.

3. Tester l'algorithme sur le problème quadratique. Afficher le nombre d'itérations nécessaires en fonction du quotient $\frac{M}{m}$. Comparer avec le gradient à pas fixe.

Remarque : il existe encore d'autres règles de *recherche linéaire inexacte* pour le choix d'un pas de descente comme la règle de Goldstein. Ces règles permettent d'éviter le problème de *recherche linéaire exacte* tout en accélérant les méthodes de gradient.

3 Gradient conjugué

But : Le gradient conjugué est une méthode de type quasi-Newton qui a la propriété de converger en temps fini pour des fonctionnelles quadratiques. On présente ici une extension de cet algorithme.

Gradient conjugué On itère de la manière suivante :

1. *Direction de descente :* $\beta^k = \frac{\langle \nabla J(x^k) - \nabla J(x^{k-1}), \nabla J(x^k) \rangle}{\|\nabla J(x^{k-1})\|^2}$,
2. *Pas de descente :* on applique une règle de *recherche linéaire inexacte* (ici la règle d'Armijo) pour trouver le meilleur pas de descente pour la direction de descente $d^k = -\nabla J(x^k) + \beta^k d^{k-1}$,
3. *Itération :* $x^{k+1} = x^k + \alpha^k d^k$.

Remarque : on a ici présenté l'algorithme de *Polak-Ribière* mais il existe d'autres variantes comme *Fletcher-Reeves*.

Banane de Rosenbrock et problème quadratique On considère de nouveau les fonctions du premier exercice.

1. Implémenter la méthode du gradient conjugué version *Polak-Ribière*.

2. Tester l'algorithme avec pour initialisation $x^0 = (-1, 1)$. Comme pour le premier exercice, afficher le nombre d'itérations nécessaire pour atteindre le critère d'arrêt suivant $J(x^{k+1}) - J(x^*) < 10^{-4}$. Conclure.

3. Pour un problème quadratique, le gradient conjugué converge en au plus n itérations où n est la taille du problème. Vérifier cette affirmation pour différentes tailles de matrices dans le cadre du problème quadratique introduit précédemment.

4 Le problème de Lennard-Jones

But : Les problèmes de conformation atomique consistent à trouver les coordonnées spatiales de N noyaux atomiques formant une molécule d'énergie minimale. Dans le cas du problème de Lennard-Jones de taille N, la force d'interaction entre deux atomes identiques distants d'une longueur r est supposée issue d'un potentiel radial de van der Walls, s'écrivant sous forme adimensionnée

$$V(r) = \frac{1}{r^{12}} - \frac{2}{r^6}.$$

Soit une molécule u composée de N atomes $u := (X_i)_{i\in[\![1,N]\!]} = ((x_i, y_i, z_i))_{i\in[\![1,N]\!]} \in (\mathbb{R}^3)^N$, l'énergie potentielle totale du système est donnée par

$$J(u) = \sum_{(i,j)\in[\![1,N]\!], i<j} V\left(\|X_i - X_j\|\right).$$

La configuration la plus stable de la molécule $u^\star$ correspond à un minimum global d'énergie potentielle. Lorsque $N > 4$, le problème de la recherche du minimum global de J devient complexe. Il est d'ailleurs conjecturé que le nombre de minimums locaux de J croît exponentiellement avec N.

Recherche du minimum global on va tenter d'appliquer les méthodes précédemment introduites pour l'étude de ce problème physique.

1. Tracer la fonction V sur $]0, 10[$.

2. Appliquer la méthode du gradient conjugué avec règle d'Armijo pour ce problème. Tester l'algorithme pour $N = 4$. on initialisera l'algorithme de la manière suivante : $X_1 = 0$ et (X_1, X_2, X_3) choisis aléatoirement de manière indépendante dans $B(0, \sqrt[3]{N})$.

3. Pour $N = 4$ le minimum de la fonction J est atteint pour tout **tétraèdre** (X_1, X_2, X_3, X_4) et vaut -6, afficher (X_1, X_2, X_3, X_4). Interpréter.

4. Reprendre la question précédente pour $N = 13$. Dans ce cas le minimum de la fonction J est atteint pour tout **icosaèdre régulier** $(X_1, \dots, X_{13})$. Interpréter.

5 Méthode de Levenberg-Marquardt, Gauss-Newton et application à la régression non linéaire

But : Étant donné $(x_i, y_i)_{i\in[\![1,N]\!]} \in (\mathbb{R}^2)^N$ on introduit le problème statistique classique de la régression non linéaire, i.e on cherche à minimiser la fonction suivante

$$F(\theta) = \frac{1}{2}\sum_{i=1}^{N}(f(x_i, \theta) - y_i)^2,$$

avec $f \in \mathcal{C}^1(\mathbb{R}\times\mathbb{R}^k, \mathbb{R})$. Cela correspond à rechercher l'*estimateur du maximum a posteriori* dans le cadre où l'erreur sur les données est gaussienne. Dans le cadre où certaines données sont aberrantes (*outliers* en anglais) l'utilisation de la norme $\ell^2(\mathbb{R}^N)$ est peu judicieuse et on la remplace par la norme $\ell^1(\mathbb{R}^N)$ malheureusement plus dure à minimiser. On introduit donc

$$G(\theta) = \frac{1}{2}\sum_{i=1}^{N}\eta\left(f(x_i, \theta) - y_i\right),$$

avec η une fonction *de coût* sous-quadratique définie sur $\mathbb{R}$. On note :

- $\eta_1 : x \mapsto x^2$,
- $\eta_2 : x \mapsto x^2$, si $x \in [0, 1]$ et $2x - 1$ sinon (*fonction de perte de Huber*),
- $\eta_3 : x \mapsto 2(\sqrt{1 + x^2} - 1)$,
- $\eta_4 : x \mapsto \ln(1 + x^2)$,
- $\eta_5 : x \mapsto \arctan(x^2)$.

Dans notre cas on pose $f : \mathbb{R} \times \mathbb{R}^k \mapsto A\exp(-\sigma x)\sin(\omega x)$. On va maintenant introduire la méthode de Newton et la méthode de Levenberg-Marquardt qui est une méthode de quasi-Newton.

Méthode de Newton Supposons que $f \in \mathcal{C}^2(\mathbb{R}\times\mathbb{R}^k, \mathbb{R})$ on note $H(x)$ sa hessienne. On introduit la suite $\theta^k := (A^k, \sigma^k, \omega^k)$ (avec initialisation aléatoire). Les itérations sont données par

$$\theta^{k+1} = \theta^k - \alpha M_N(\theta^k)^{-1} \sum_{i=1}^{N} \nabla_\theta f(x_i, \theta^k)\eta'(f(x_i, \theta^k) - y_i),$$

où on a

$$\nabla_\theta f(x, \theta) = \begin{pmatrix} \exp(-\sigma x)\sin(\omega x) \\ -Ax\exp(-\sigma x)\sin(\omega x) \\ Ax\exp(-\sigma x)\cos(\omega x) \end{pmatrix},$$

$$H_\theta(x, \theta) = \exp(-\sigma x) \begin{pmatrix} 0 & -x\sin(\omega x) & x\cos(\omega x) \\ -x\sin(\omega x) & Ax^2\sin(\omega x) & -Ax^2\cos(\omega x) \\ x\cos(\omega x) & -Ax^2\cos(\omega x) & Ax^2\sin(\omega x) \end{pmatrix},$$

$$M_N(\theta) = \sum_{i=1}^{N} \nabla_\theta f(x_i, \theta)\nabla_\theta f(x_i, \theta)^T \eta''(f(x_i, \theta) - y_i) + H_\theta(x_i, \theta)\eta'(f(x_i, \theta) - y_i).$$

On rappelle que la méthode de Newton est une méthode d'ordre deux.

Méthode de Levenberg-Marquardt Dans certains cas, par exemple lorsque la hessienne est mal conditionnée, lorsqu'elle ne peut pas être stockée en mémoire ou lorsqu'elle n'est tout simplement pas calculable, on peut utiliser la méthode de Levenberg-Marquardt dont les itérées sont données par

$$\theta^{k+1} = \theta^k - \alpha M_{LM}(\theta^k)^{-1} \sum_{i=1}^{N} \nabla_\theta f(x_i, \theta^k)\eta'(f(x_i, \theta^k) - y_i),$$

avec $M_{LM} = \mu I_3 + \sum_{i=1}^{N} \nabla_\theta f(x_i, \theta)\nabla_\theta f(x_i, \theta)^T \eta''(f(x_i, \theta) - y_i)$ où μ est un paramètre. La méthode de Levenberg-Marquardt est une méthode de quasi-Newton. Elle requiert par contre l'introduction d'un paramètre de régularisation μ.

1. Visualiser sur $[0, 10]$ les différentes fonctions *de coût* introduites dans l'énoncé.

2. Générer des données à l'aide de la fonction suivante.

```
def generate_data(t, A, sigma, omega, noise=0, n_outliers=0):
    y = A * np.exp(-sigma * t) * np.sin(omega * t)
    error = noise * rnd.randn(t.size)
    outliers = rnd.randint(0, t.size, n_outliers)
    error[outliers] *= 35
    return y + error

A = 2
sigma = 0.1
```

```
omega = 0.1 * 2 * np.pi
x_true = np.array([A, sigma, omega])

noise = 0.1
n_outliers = 4

t_min = 0
t_max = 30

t_train = np.linspace(t_min, t_max, 30)
y_train = generate_data(t_train, A, sigma, omega, noise, n_outliers)
```

Visualiser les données générées.

3. Implémenter la méthode de Newton et la méthode de Levenberg-Marquardt pour la fonction *de coût* quadratique.

4. Tester les deux méthodes sur les données générées. On initialisera les paramètres à $\theta^0 = (1, 1, 1)$. Interpréter. Faire varier le nombre de données aberrantes et conclure sur la nécessité d'utiliser d'autres fonctions *de coût*.