OPTIMISATION 3

Leçon partie 1

Cette séance s'intéresse à la recherche d'un extremum d'une fonction à l'aide d'une méthode algorithmique efficace : la méthode du gradient. Cette méthode fonctionne avec les fonctions à une ou plusieurs variables.

Pour comprendre la méthode, on peut utiliser une image. Considérons une bille placé en haut d'une colline. Elle descendra la pente en suivant la ligne de plus grande pente et elle s'arrêtera au point bas. C'est exactement ce que fait la descente de gradient : en partant d'un point initial, on cherche la pente la plus grande en calculant le gradient et on descend d'un petit pas, on recommence à partir du nouveau point jusqu'à atteindre un minimum local.

Le gradient est la généralisation de la notion de dérivée : c'est un vecteur qui remplace la notion de dérivée pour les fonctions de plusieurs variables. On sait que la dérivée permet de décider si une fonction est croissante ou décroissante. De même, le vecteur gradient indique la direction dans laquelle la fonction croît ou décroît le plus vite.

Il existe deux nommages pour cette méthode :

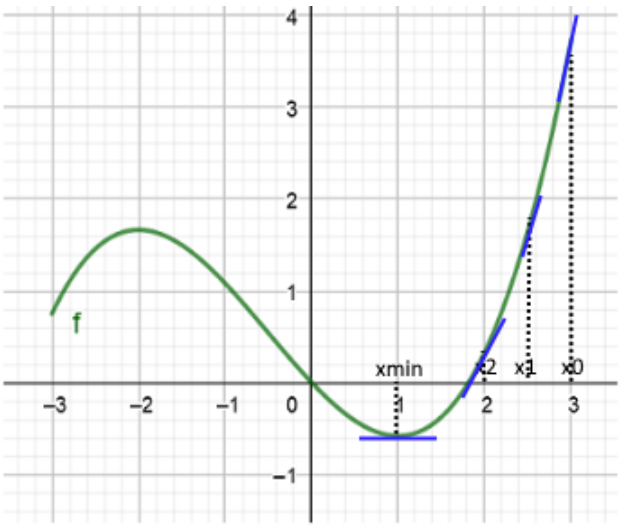
- La montée du gradient qui correspond à la recherche d'un maximum
- La descente du gradient qui correspond à la recherche d'un minimum

Les deux méthodes sont liées. En effet, rechercher un maximum sur la fonction f revient à rechercher un minimum sur la fonction opposée -f. Il est courant d'utiliser la 2eme méthode en particuleir dans la recherche de minima dans le domaine de l'IA.

Fonction à une variable :

Travaillons pour le moment sur une fonction f à une variable f : x → f(x). grad f est tout simplement un vecteur (1,1) dont la valeur est f'(x).

Dans ce cas, la descente du gradient consiste à placer un point à une abscisse initiale x_0 proche d'un minimum atteint à l'abscisse x_{min} puis à faire des petits pas x_1, x_2, x_3 ...en direction de x_{min} .



Un pas est défini par un coefficient α et le nombre dérivée appelée gradient. Une nouvelle abscisse x_{i+1} est calculée par les paramètres α et le nombre dérivé en x_i

$$x_{i+1} = x_i - \alpha \, grad_f(x_i)$$

On arrête les pas vers x_{min} lorsque le gradient est inférieur à une certaine seuil s, par exemple 0,0,1. Cela correspond à un nombre dérivée quasi nul ou une tangente horizontale, c'est à dire un extremum de fonction. On définit souvent un 2eme critère d'arrêt représentant le nombre maximum d'itérations (de pas) entre x_0 et x_{min}

Exercice 1: descente du gradient simple

Dans cet exercice, on définit une fonction gradient_descent qui code l'algorithme en 4 étapes :

- étape 1 : initialisation des valeurs : la valeur initiale X_0 = 2 et le coefficient α sont des paramètres. Les conditions d'arrêt sont définies: gradient plus petit que 0,01 ou un nombre maximum d'itérations égal à 10.
- étape 2 : calcul du gradient en x_i
- étape 3 : calcul de x_{i+1} : $x_{i+1} = x_i - \alpha\, grad_f(x_i)$
- étape 4 : vérification des conditions d'arrêt valeur du gradient par rapport au seuil et le nombre maximum d'itérations. Si les conditions d'arrêt ne sont pas atteintes, l'algorithme boucle à l'étape 2. Dans le cas contraire, l'algorithme retourne x_{i+1} qui représente une valeur approchée de x_{min}

L'efficacité de l'algorithme dépend fortement du choix de x_0 et de α.
Une valeur trop faible du coefficient α induit une convergence lente de x_i vers x_{min} .
Une valeur trop grande de α empêche la convergence de x_i vers x_{min} .
Une valeur x_0 trop éloignée de x_{min} induit souvent une convergence incomplète car le critère d'arrêt du nombre maximum d'itérations est atteint.

a. Créez un fichier **gradient.py** puis coder la fonction **gradient_descent_simple** avec le code :

```
def gradient_descent_simple(X0, alpha):
    # initalisation
    X = X0
    gradX = grad_h(X0)
    nb_tour = 0
    # boucle des itérations
    while(abs(gradX)>1e-2 and nb_tour < 10):
        gradX = grad_h(X)
        X -= gradX * alpha * (-1)
        nb_tour += 1
    return X
```

```
def gradient_descent_cours(X0, alpha):
    # initalisation
    X = X0
    gradX = grad_h(X0)
    nb_tour = 0
    # boucle des itérations
    while(abs(gradX)>1e-2 and nb_tour < 10):
        gradX = grad_h(X)
        X -= gradX * alpha * (-1)
        nb_tour += 1
    return X
```

Dans le même fichier, coder une fonction **exercice1** avec le code suivant :

```
def exercice1():
    print("** EXERCICE 1 **")
    print("Extremum : fmax=51.11 en x=3.35")
    print("Alpha = 0.01")
    gradient_descent_cours(2, 0.01)
    print("Alpha = 0.1")
    gradient_descent_cours(2, 0.1)
    print("Alpha = 0.5")
    gradient_descent_cours(2, 0.5)
    print("Alpha = 1")
    gradient_descent_cours(2, 1)
```

```
def exercice1():
    print("** EXERCICE 1 **")
    print("Extremum : fmax=51.11 en x=3.35")
    print("Alpha = 0.01")
    gradient_descent_cours(2, 0.01)
    print("Alpha = 0.1")
    gradient_descent_cours(2, 0.1)
    print("Alpha = 0.5")
    gradient_descent_cours(2, 0.5)
    print("Alpha = 1")
    gradient_descent_cours(2, 1)
```

b. Sur une feuille, calculer la dérivée de la fonction h définie par x → $-x^2+\frac{20}{3}x+40$ à la main

Coder la fonction **grad_h** avec l'expression de h' que vous venez de calculer.

c. Exécuter le programme. Observer l'influence des différents paramètres lors des 4 appels à la fonction. Indiquer vos remarques dans un commentaire ajouté à la fonction exercice1.

Exercice 2: descente du gradient complète

a. Codez la fonction **gradient_descent** qui offre la même fonctionnalité avec une liste plus importante de paramètres :

- grad : expression du gradient
- X0 : valeurs initiales
- alpha : taux d'apprentissage
- stop_condition : condition d'arrêt sur la valeur absolue du gradient. La valeur par défaut est 1e-2
- max_iter : condition d'arrêt sur le nombre maximum d'itérations. La valeur par défaut est 100
- sens : sens du calcul du nouveau paramètre. La valeur par défaut est +1. L'autre valeur peut être -1.

Dans la fonction, vous ajusterez le code pour supporter les différents paramètres listés. Vous calculerez également le temps d'exécution et vous afficherez avant le return les informations : le dernier x_{i+1} , le nombre d'itérations, le temps de traitement en ms.

b. Codez une fonction **exercice2**. Celle-ci :

- appelle la fonction gradient_descent pour calculer le minimum en passant l'expression de la fonction dérivée de x → (x-1)(x-3)(x-5)+15, appelée grad dans les paramètres, ainsi que les valeurs : X_0 = 3,3, alpha = 0,1 , stop_condition = 0,001. Vous pouvez calculer grad à la main puis coder cette fonction en Python.
- affiche l'abscisse obtenue et son image
- appelle la fonction gradient_descent pour calculer le maximum en passant l'expression de la fonction dérivée de x → (x-1)(x-3)(x-5)+15 appelée grad dans les paramètres, ainsi que les valeurs : X_0 = 2,9, alpha = 0,1 , stop_condition = 0,001, sens = -1.
- affiche l'abscisse obtenue et son image

c. Vérifiez la cohérence des résultats obtenus avec ceux de l'exercice 2 du TD2.

d. Est-il possible d'obtenir le résultat attendu en mois d'itérations pour la même condition d'arrêt (stop_condition) ? Ajoutez votre réponse en commentaire de la fonction exercice2.

Exemple d'affichage partiel :

```
** EXERCICE 2 **
Descente gradient, 5 itérations, calcul en 0.008 ms
Minimum 11.920798564326567 en 4.1547016872523175
```

Exercice 3: plusieurs extrema locaux

Dans cet exercice, on travaille sur une fonction présentant plusieurs minima. On s'intéresse à la difficulté de choisir des valeurs initiales permettant d'avoir tous les minima possibles. On propose une solution finale reposant sur un choix aléatoire.

On considère la fonction $r : x \rightarrow x^4 - 5x^2 + x + 10$
La liste initiale des valeurs de x_0 est : -2 , -0.5 , 0 , 0.11 , 0.5 , 2
alpha = 0.02

a. Coder les fonctions **r** et **gradr** (gradient de r).

b. Coder la fonction **exercice3** qui :

- a pour paramètres une liste de valeurs de x_0
- appelle gradient_descent pour chaque valeur de x_0, sauvegarde et affiche le retour de gradient_descent dans une liste appelée candidats
- affiche la courbe de r sur l'intervalle [-3 , 3] ainsi que les différents points de coordonnées (x_0 , r(x_0)) sur le même graphique

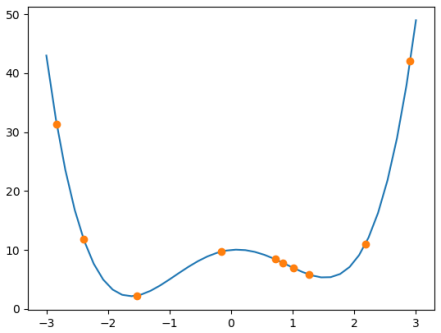
- élimine les résultats similaires de la liste candidats. Deux résultats sont considérés comme similaires lorsque la différence de leurs ordonnées est < 0.1
- affiche le minimum global et les minima locaux s'ils existent

c. Appeler `exercice3([-2,-0.5,0,0.11,0.5,2])` et vérifier son bon fonctionnement. L'affichage doit conduire à un minimum global et un minimum local.
d. Piocher aléatoirement 10 valeurs réelles sur l'intervalle [-3 , 3] suivant une loi uniforme et les placer dans une liste. Appeler exercice3 en passant votre liste de valeurs aléatoires en arguments. Vérifier que le résultat obtenu est cohérent avec la question c.

Le choix aléatoire de valeurs initiales permet de couvrir tout l'intervalle d'études afin de lister tous les extrema : global et locaux.

Exemple d'affichage partiel :

```
** EXERCICE 3 **
Descente gradient, 17 itérations, calcul en 0.011 ms
 Minimum 2.144605814532638 en -1.6287864976991937
Descente gradient, 31 itérations, calcul en 0.01 ms
 Minimum 5.305294545394004 en 1.528230681996116
Descente gradient, 38 itérations, calcul en 0.011 ms
 Minimum 2.144606156174371 en -1.628708639840655
Descente gradient, 27 itérations, calcul en 0.009 ms
 Minimum 2.144605877558684 en -1.6287695682298753
Descente gradient, 12 itérations, calcul en 0.004 ms
 Minimum 2.144606144809811 en -1.629186159935242
```



Leçon partie 2 : fonctions de plusieurs variables

Dans les pages précédentes, le gradient était confondu avec la fonction dérivée pour une fonction de $\mathbb{R} \rightarrow \mathbb{R}$. Dans un cas plus général, le gradient de f noté ∇f est un vecteur de dérivées partielles pour une fonction f de $\mathbb{R}^p \rightarrow \mathbb{R}$ avec p entier naturel >0.

Étudions les fonctions de plusieurs variables, leurs dérivées partielles et les propriétés liées aux extrema. Les notions de lignes de niveau, de continuité, de dérivabilité, de dérivées directionnelles ne sont pas abordées dans ce cours.

Définitions : fonction et représentation graphique

On appelle fonction numérique à n variables toute fonction f définie sur un sous ensemble D de $\mathbb{R}^n$ à valeur dans $\mathbb{R}$: f : $(x_1, x_2, \dots, x_n) \rightarrow f(x_1, x_2, \dots, x_n)$

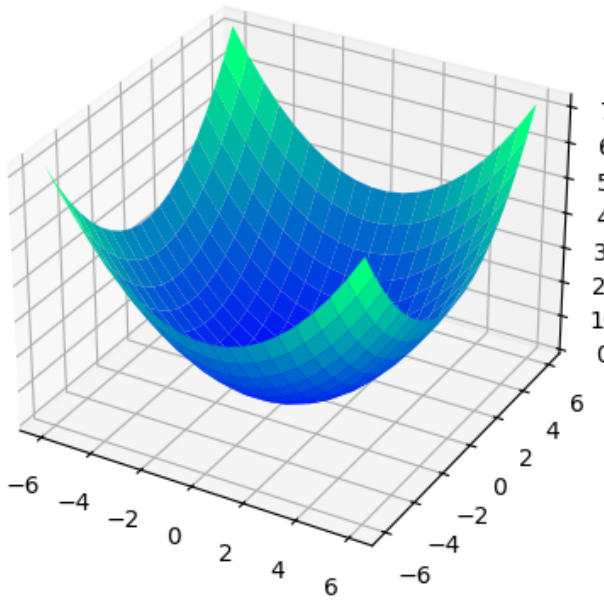
On appelle graphe de f le sous-ensemble de $\mathbb{R}^{n+1}$ défini par :

$C_f = \{ (x_1, x_2, \dots, x_{n+1}) \in \mathbb{R}^{n+1}, x_{n+1} = f(x_1, x_2, \dots, x_n) \}$

Exemple :

f : $\mathbb{R}^2 \rightarrow \mathbb{R}$ définie par $f(x,y)=x^2+y^2$ est une fonction de plusieurs variables.

Sa représentation graphique sur [-6,6]x[-6,6] est l'ensemble des points (x,y,z) $\in \mathbb{R}^3$ tels que $z=x^2+y^2$ avec x ∈ [-6,6] et y ∈ [-6,6]. On peut représenter la fonction à l'aide d'un programme Python utilisant la fonction meshgrid du module numpy et le paramètre 3d de axes dans le module matplotlib.pyplot.




```
import matplotlib.pyplot as plt
import numpy as np

a,b = -6,6
c,d = -6,6
n = 20

x=np.linspace(a,b,n)
y=np.linspace(c,d,n)

X,Y = np.meshgrid(x,y)
Z = f(X,Y)

ax = plt.axes(projection = '3d')
ax.plot_surface(X,Y,Z)
plt.show()
```

Définition : dérivées partielles

Soit i indice entier pris dans l'intervalle [1,n] et a = $(a_1, \dots a_n) \in \mathbb{R}^n$. On dit que f admet une dérivée partielle d'ordre 1 par rapport à la ième variable en a lorsque :

$$\lim_{h \rightarrow 0} \frac{f(a_1,\dots,a_i+h,\dots,a_n)-f(a_1,\dots,a_i,\dots,a_n)}{h} \text{ existe}$$

On appelle alors dérivée partielle d'ordre 1 de f par rapport à la ième variable en a et on note $\partial x_i(f)(a)$ le réel : $\partial i(f)(a) = f'_{a,i}(a_i)$.

On peut noter aussi : $\frac{\partial f}{\partial x_i}(a)$ à la place de $\partial x_i(f)(a)$

Méthode de calcul

Une dérivée partielle $\frac{\partial f}{\partial x_i}$ est une dérivée de f par rapport à la variable x_i . Pour la calculer, on considère que les autres variables que x_i sont constantes et on dérive par rapport à x_i .

Exemple : On considère f : $\mathbb{R}^2 \rightarrow \mathbb{R}$ définie par $\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow 2x + xy$

Pour dériver l'expression $2x + xy$ par rapport à x, on considère y comme un nombre fixé puis on dérive par rapport à x ainsi $\frac{\partial f}{\partial x}(x,y) = 2 + y$. De même, pour dériver l'expression $2x + xy$ par rapport à y , on considère x comme un nombre fixé puis on dérive par rapport à y ainsi $\frac{\partial f}{\partial y}(x,y) = 0 + x$

Définition : gradient

Soit a $\in \mathbb{R}^n$ et f : $\mathbb{R}^n \rightarrow \mathbb{R}$ une fonction de plusieurs variables. On suppose que f admet une dérivée partielle d'ordre 1 en a par rapport à chaque variable.

On appelle gradient de f en a, le vecteur de $\mathbb{R}^n$ défini par : $(\frac{\partial f}{\partial x_1}(a), \frac{\partial f}{\partial x_2}(a), \ldots, \frac{\partial f}{\partial x_n}(a))$

Le gradient est noté ∇(f)(a), se lit «nabla de f en a».

Exemple : On considère f : $\mathbb{R}^2 \rightarrow \mathbb{R}$ alors ∇(f) : $\mathbb{R}^2 \rightarrow \mathbb{R}^2$

définie par $\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow 2x + xy$ $\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{pmatrix} \frac{\partial f}{\partial x}(x,y) \\ \frac{\partial f}{\partial y}(x,y) \end{pmatrix} = \begin{pmatrix} 2+y \\ x \end{pmatrix}$

Définitions : extremum

Soit a $\in \mathbb{R}^n$ et f : $\mathbb{R}^n \rightarrow \mathbb{R}$ une fonction de plusieurs variables.

On dit que f admet un minimum global (ou un maximum global) lorsque : $\forall$ x $\in \mathbb{R}^n$, f(x) ≥ f(a) (ou f(x) ≤ f(a)). On parle plus généralement d'extremum pour un minimum ou un maximum.

Condition nécessaire

Soit f une fonction de plusieurs variables ayant des dérivées partielles.

Si f admet un extremum global en c alors ∇(f)(c) = (0, … ,0).

Attention : la réciproque est fausse !

Exemple : Soit f : $\mathbb{R}^2 \rightarrow \mathbb{R}$ définie par $f(x,y) = x^2 + y^2$

$\forall$ (x,y) $\in \mathbb{R}^2$, $x^2 + y^2 \geq 0$, ce qui s'écrit aussi $f(x,y) \geq 0$ donc f admet un minimum global en (0,0). Or ∇(f)(x,y) = (2x , 2y) donc ∇(f)(0,0)=0 s'annule bien en (0,0)

Exercice 4 : calculs et représentation graphique

On considère la fonction f : $\mathbb{R}^2$ → $\mathbb{R}$ définie par $f(x,y)=x^2+2x+3y^2-6y+1$

a. Coder les fonctions Python :
- **f** qui retourne l'image f(x,y) de (x,y) paramètre de f
- **exercice4** qui affiche la représentation graphique de f sur [-6,6]x[-6,6]

b. Calculer son gradient à la main

c. Un autre programme informatique indique que f admet un minimum en $\left(-1,\frac{1}{2}\right)$. Est-ce vrai ?

En quel point le gradient de f s'annule-t-il ? Répondre aux 2 questions dans un commentaire

d. Il est difficile de vérifier graphiquement que le point trouvé en d. en un minimum. Pour faciliter cette vérification visuelle, on peut représenter le projeté de la représentation graphique dans un plan et indiquer par la couleur des surfaces la valeur du gradient, ce qui représente la pente de la surface. En Python, vous utiliserez la méthode pcolormesh de l'objet pyplot.axes. Par exemple : ax.pcolormesh(X, Y, Z) pour représenter cette projection.
Compléter la fonction exercice4 pour afficher le projeté en plus de la représentation graphique.

Toutefois, la recherche visuelle n'est pas pratique. Il serait intéressant d'avoir un outil plus performant. Nous allons généraliser la descente du gradient aux fonctions de plusieurs variables. Celle-ci nous fournira une méthode algorithmique efficace pour déterminer un minimum.

Exercice 5 : descente du gradient vectoriel

Dans cet exercice, nous manipulons des vecteurs colonnes ou lignes de dimensions 2. Ils sont équivalents à des matrices de dimensions 2,1. La bibliothèque numpy.array permet de manipuler des matrices et d'effectuer des opérations : * est la multiplication par un scalaire, add ou subtract sont les opérations d'addition ou de soustraction de matrices de même dimensions… En savoir plus

a. Calculer à la main le gradient de la fonction f de l'exercice 4 puis coder en Python la fonction **gradf** qui retourne le vecteur ∇(f)(x,y), de type array, à partir de X paramètre de type array.

b. Coder une fonction **exercice5** qui calcule puis affiche ∇(f)(-3,2)

c. Dans cette question, coder une fonction **gradient_descent_vect** avec le contenu de votre fonction gradient_descent de l'exercice 2.
Vous apporterez quelques modifications pour supporter les vecteurs :
- les paramètres grad et X0 sont des matrices de type numpy.array
- Les paramètres stop_condition et max_iter sont toujours présents avec la même signification. Leurs valeurs par défaut sont 10^{-3} , 50000
- Le paramètre sens est remplacé par un paramètre type_calcul avec une valeur par défaut 'norm'
- Les calculs dans la boucle sont tous matriciels sur des vecteurs de type numpy.array
- La condition d'arrêt vérifie que la norme du gradient est inférieure au seuil de valeur stop_condition. Cette condition est équivalente à la condition nécessaire du cours : en un extremum, le gradient tend vers le vecteur nul. Dans votre programme La norme d'un vecteur se calcule avec numpy.linalg.norm

La logique de la fonction et les informations affichées sont identiques avec celles de la fonction gradient_descent de l'exercice 2.

d. Compléter la fonction exercice5 pour appeler et afficher le résultat de la fonction gradient_descent_vect avec les paramètres gradf, X0 = (0,0), alpha = 0.1 pour 50 itérations au maximum.
Il reste à vérifier qu'un extremum est bien atteint au point trouvé dans la question c. Dans cet exercice, cette vérification est évidente en s'appuyant sur la représentation graphique de la fonction f obtenue à l'exercice précédent mais ce n'est pas toujours le cas !

Exercice 6 : application

Dans cet exercice, on utilise l'algorithme de descente du gradient pour résoudre un système linéaire qui peut s'écrire sous la forme A X = b avec A et b matrices de dimensions 3,3 et 3,1.

Pour cela, on introduit une fonction d'erreur J : $\mathbb{R}^3 \rightarrow \mathbb{R}$ définie par :

$$X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \rightarrow \|AX-b\|^2$$

La notation $\|AX-b\|^2$ est la norme de AX-b, au carré. Elle représente ici la distance entre les vecteurs AX et b. Résoudre le système revient à trouver la valeur du vecteur X pour lequel AX = b c'est à dire J(X) tend vers son minimum global 0.

Le gradient de J s'écrit : ∇J : $\mathbb{R}^3 \rightarrow \mathbb{R}^3$

$$X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \rightarrow \begin{pmatrix} \frac{\partial J}{\partial x_1}(x_1,x_2,x_3) \\ \frac{\partial J}{\partial x_2}(x_1,x_2,x_3) \\ \frac{\partial J}{\partial x_3}(x_1,x_2,x_3) \end{pmatrix} = 2A^T(AX-b)$$

A^T est la transposée de la matrice A.

Trouver le minimum global de J revient à trouver X qui tend vers la solution du système et ∇J qui tend vers 0.

Pour la suite, on veut résoudre le système :

$$\begin{pmatrix} 1 & -2 & 2 \\ 3 & -5 & 9 \\ -2 & 3 & -6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \\ 2 \end{pmatrix}$$ qui peut s'écrire A X = b

a. Coder les fonctions **J** et **dJ** (gradient de J) en utilisant des calculs matriciels et les matrices suivantes :

```
A=np.array([[1,-2,2],[3,-5,9],[-2,3,-6]])
b=np.array([1,-1,2])
```

Ces fonctions prennent un vecteur X en entrée et retourne un nombre réel pour J et une matrice de dimensions 3,1 pour dJ.

b. Coder une fonction **exercice6** qui appelle la descente du gradient vectoriel avec les paramètres : dJ, une valeur initiale (0,0,0), 10 itérations au maximum pour différentes valeurs de alpha prise dans la liste : Alpha=[1,0.5,0.1,0.01,0.001,0.005] dans une première boucle. L'objectif est de déterminer les deux meilleures valeurs de alpha, c'est à dire celles montrant le résultat J convergeant le plus rapidement vers 0.

c. Compléter la fonction exercice6 pour qu'elle appelle la descente du gradient vectoriel avec les paramètres : dJ, une valeur initiale (0,0,0), avec la limite par défaut du nombre d'itérations (50000).

Pour finir, la fonction exercice6 affiche les solutions du système linéaire par une autre méthode. Vous pouvez utiliser un solver d'équations, par exemple numpy.linalg.solve afin de trouver la solution du système.
Ajoutez un commentaire à la fin d'exercice6 sur la comparaison des résultats trouvés par la descente du gradient vectoriel et le solveur d'équations.