TP2 correction : Méthodes de gradient

1 Ensemble de niveau

Exercice 1. Tracé des lignes de niveau, des gradients.

1.

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

def f(a,b,x,y):
    return .5*(a*x*x+b*y*y)

# definir un maillage carre, les valeurs parcourues par x et y
x=np.linspace(-5,5)
y=np.linspace(-5,5)
X, Y = np.meshgrid(x, y)
# evaluer la fonction pour un choix de a et b
a,b=2,4
Z = f(a,b,X,Y)
# tracer les lignes de niveau
# plusieurs options possibles
#option1
plt.contour(X, Y, Z)
plt.show()
#option2
dessin=plt.contour(X, Y, Z)
plt.clabel(dessin)
plt.show()
#option3
dessin=plt.contour(X, Y, Z, levels=np.linspace(0,20,6))
plt.clabel(dessin, fmt= '%.2f', inline = True)
plt.grid()
plt.axes().set_aspect('equal')
plt.show()
```

2.

$$\nabla J(x,y) = \begin{pmatrix} ax \\ by \end{pmatrix}$$

3.

```
 # Tracer l'ensemble de niveau et le gradient en 1 point
a,b=1,5
Z = f(a,b,X,Y)
x0,y0=3.5,1.5
t=f(a,b,x0,y0)
dessin=plt.contour(X, Y, Z, levels=[t])
plt.clabel(dessin, fmt= '%.3f', inline = True)
plt.quiver(x0,y0,a*x0,b*y0)
plt.grid()
plt.axes().set_aspect('equal')
plt.show()
```

```
# gradient en plusieurs points
X0 = np.arange(-5, 5, 1)
Y0 = np.arange(-5, 5, 1)

for x0 in X0: # on trace les vecteurs en chaque point
    for y0 in Y0:
        plt.quiver(x0,y0,a*x0,b*y0)

dessin=plt.contour(X, Y, Z)# on trace les lignes de niveau
plt.clabel(dessin)

plt.grid()
plt.axes().set_aspect('equal')
plt.show()
```

2 Méthode de gradient à pas fixe

Exercice 2. Gradient à pas fixe.

1.
```
# on trace les lignes de niveau

def F(a,b,x,y):
    return a*x*x+b*y*y

x=np.linspace(-5,5)
y=np.linspace(-2,2)
X, Y = np.meshgrid(x, y)
a,b=1,1
Z = F(a,b,X,Y)
dessin=plt.contour(X, Y, Z, levels=np.linspace(0,20,6))
plt.clabel(dessin, fmt= '%.3f', inline = True)
plt.grid()
plt.axes().set_aspect('equal')
plt.show()
```

2. Pour $u, v \in \mathbb{R}^2$, on a

$$\langle \nabla F(u) - \nabla F(v), u - v \rangle = \langle 2(u - v), u - v \rangle = 2\|u - v\|^2$$

F est donc 2-convexe.

3. On remarque que ∇F est Lipschitz avec une constante égale à 2.

4.
```
from numpy.linalg import norm
def GradF(a, b, v): #calcul de la fonction gradient
    x, y = v[0], v[1]
    F=np.zeros((2,1))
    F[0]=2*a*x
    F[1]=2*b*y
    return F

def gf(GradF, v0, rho, tol, NitMax):
    a,b = 1, 1 #pour le cas x^2 + y^2
    v=v0
    gradfx=GradF(a,b,v)
    it=0
    X = [v0]
    while  norm(gradfx) > tol and it <NitMax: #critere d'arret sur le gradient
        v=v-rho*gradfx
```

```
        X.append(v) # on sauve l'iteration
        gradfx=GradF(a,b,v)
        it+=1
    cvg = norm(gradfx) <= tol #on verifie si on a convergence
    X = np.array(X) # on transforme la liste en array
    return v,X,it,cvg
```

5. D'après le cours, pour que la méthode converge il faut que $0 <$ `rho` $< \frac{2}{M}$ avec M la constante de Lipschitz donc ici $0 <$ `rho` < 1 et le meilleur choix est `rho` $= \frac{\alpha}{M^2} = \frac{1}{2}$.
6. On remarque que le nombre d'itérations pour la tolérance demandée dépend du pas choisi. On obtient un nombre d'itérations minimal avec la valeur optimale du pas `rho` $= \frac{1}{2}$.

```
# on applique pour differentes valeurs de pas
v0 = np.ones((2,1))
tol = 1e-4
NitMax = 3000

rho1 = 0.8
Sol1, Suite1, it1, cvg1 =  gf(GradF, v0, rho1, tol, NitMax)
print(Sol1)
print(cvg1, it1)

rho2 = 0.1
Sol2, Suite2, it2, cvg2 =  gf(GradF, v0, rho2, tol, NitMax)
print(Sol2)
print(cvg2, it2)

rho3 = 0.5 # valeur optimale
Sol3, Suite3, it3, cvg3 =  gf(GradF, v0, rho3, tol, NitMax)
print(Sol3)
print(cvg3, it3)
```

Exercice 3. Gradient à pas fixe - critère d'arrêt

1.
```
def gf(GradF, v0, rho, tol, NitMax):
    a,b = 1, 100 #pour le cas x^2 + 100 y^2
    v=v0
    gradfx=GradF(a,b,v)
    it=0
    X = [v0]
    while  norm(gradfx) > tol and it <NitMax: #critere d'arret sur le gradient
        v=v-rho*gradfx
        X.append(v) # on sauve l'iteration
        gradfx=GradF(a,b,v)
        it+=1
    cvg = norm(gradfx) <= tol #on verifie si on a convergence
    X = np.array(X) # on transforme la liste en array
    return v,X,it,cvg

v0 = np.ones((2,1))
rho1 = 1.9999/1000.0
tol = 1e-4
NitMax = 3000
Sol, Suite, it, cvg =  gf(GradF, v0, rho, tol, NitMax)
print(Sol)
print(cvg, it)
```

2.
```
plt.figure() #suite des iteres
plt.plot(Suite[:,0], Suite[:,1], 'o', color = 'b')
plt.plot(Sol[0], Sol[1], 'X', color = 'r')
```

3.
```
plt.figure() # F(x_k, y_k)
plt.plot(range(len(Suite)), F(1, 100, Suite[:,0], Suite[:,1]), 'o')
```

4.
```
NormSuite = []
for s in Suite :
    NormSuite.append(norm(s))

plt.figure() #Norme
plt.plot(range(len(Suite)), NormSuite, 'o')
```

5.
```
def gf2(GradF, v0, rho, tol, NitMax):
    a,b = 1, 100
    v=v0
    v_old = 4*v0 # pour avoir une norme initiale > tol
    gradfx=GradF(a,b,v)
    it=0
    X = [v0]
    while  norm(v - v_old) > tol and it <NitMax: #critere d'arret sur la
                                    #difference entre 2 iterations
        v_old = v
        v=v-rho*gradfx
        X.append(v) # on sauve l'iteration
        gradfx=GradF(a,b,v)
        it+=1
    cvg = norm(v-v_old) <= tol #on verifie si on a convergence
    X = np.array(X) # on transforme la liste en array
    return v,X,it,cvg

Sol2, Suite2, it2, cvg2 =  gf2(GradF, v0, rho1, tol, NitMax)
print(Sol2)
print(cvg2, it2)
```

On remarque ici que les deux fonctions ne donnent pas la même solution. Le critère d'arret a donc une importance. En comparant la valeur de F aux points solutions obtenus on observe que la première estimation est meilleure (`F(Sol1) < F(Sol2)`).

6. À faire.

3 Méthode de gradient à pas optimal

Exercice 4. Gradient à pas optimal On s'intéresse au cas $F(x, y) = ax^2 + bx^2$, avec $a, b \in \mathbb{R}$.

1. Si le pas optimal existe il doit minimiser la fonction $g : r \mapsto F(v - ru)$ avec $v, u \in \mathbb{R}^n$ et donc vérifier $g'(r) = 0$, et :
$$g'(r) = \langle \nabla F(v - ru), -u \rangle = 0$$
d'où
$$\langle \nabla F(v_k - r\nabla F(v_k)), \nabla F(v_k) \rangle = 0$$

2. On utilise l'orthogonalité de la question précédente, pour tout $k \in \mathbb{N}$:
$$\rho_k = \frac{a^2x_k^2 + b^2y_k}{2(a^3x_k^2 + b^3y_k^2)}$$

3. Méthode du gradient à pas optimal. Le pas optimal étant calculé explicitement par la formule de la question précédente.

```
def gopt(GradF, v0, tol, NitMax):
    v = v0
    a,b = 1, 100
```

```
    gradf = GradF(a, b, v)
    it=0
    X = [v0]
    while  norm(gradf) > tol and it <NitMax: #critere d'arret sur le gradient
        #calcul le pas optimal
        rho= ( a**2 * v[0]*v[0] + b**2 * v[1]*v[1] )/
               ( 2* ( a**3 *v[0]*v[0]+ b**3 *v[1]*v[1]))
        v=v-rho*gradf
        X.append(v) # on sauve l'iteration
        gradf=GradF(a,b,v)#on met a jour le gradient
        it+=1
    cvg = norm(gradf) <= tol #on verifie si on a convergence
    X = np.array(X) # on transforme la liste en array
    return v,X,it,cvg
```

4.

```
#on applique l'algo pour 2 conditions initiales
tol = 1e-4
NitMax = 3000

v01 = np.ones((2,1))
Sol11, Suite11, it11, cvg11 =  gopt(GradF, v01, tol, NitMax)
print(Sol11)
print(cvg11, it11)

v02 = np.array([[0.], [np.sqrt(2)]])
Sol12, Suite12, it12, cvg12 =  gopt(GradF, v02, tol, NitMax)
print(Sol12)
print(cvg12, it12)
```

On remarque que suivant l'initialisation le nombre d'itérations peut changer de manière importante.

5. Dans cette situation le nombre d'itérations est beaucoup réduit avec la méthode de gradient à pas optimal par rapport à la méthode de gradient à pas fixe. De plus, pour la méthode de gradient à pas fixe l'initialisation change aussi le nombre d'itérations.

Exercice 5. Comparaison des méthodes.
On s'intéresse au cas $G(x, y) = x^2 + 2\,y^2$.

1. $r_0 = \frac{1}{3}$, l'hypothèse $r_k = \frac{1}{3}$ implique, $(x_{k+1}, y_{k+1}) = \frac{1}{3}(x_k, -y_k)$ et $r_{k+1} = \frac{(x_{k+1}^2+4y_{k+1}^2)}{(2x_{k+1}^2+16y_{k+1}^2)} = r_k$.
2. Appliquer la méthode du gradient à pas fixe pour $\tau \in \{\frac{1}{3} - 0.01, \frac{1}{3}, \frac{1}{3} + 0.01, 0.9999\}$ avec comme initialisation $v_0 = (20, 10)$

```
# Initialisation
x0= 20*np.ones((2,1))
x0[1]=10

Tol , IterMax =1.e-6 , 100
for rho in [1./3 ,1./3 - 0.01 , 1./3+0.01 ,0.9999]:
   SolG, T, NbrIter, cvg =  gf(GradF, x0, rho, Tol, IterMax)
   print('rho = ',rho, ' on trouve \ n ' , norm ( SolG ))
```

Pour $\rho = 0.9999$, la methode du gradient ne converge pas (pour les valeurs fixées de Tol et IterMax).

3. Méthode du gradient à pas optimal connu = méthode de grdient à pas fixe avec le pas calculé (ici $\frac{1}{3}$)
4. voir exercices précédents

4 Pour aller plus loin

Exercice 6. Cas non quadratique.

1.
```
def J(x,y):
    return ( x +2 * y )**4

x = np.linspace (-1 ,1)
y = np.linspace (-1 ,1)
X, Y = np.meshgrid (x , y)
Z = J(X,Y)
dessin=plt.contour(X, Y, Z)
plt.clabel(dessin)
plt.show()
```

2. En partant de la relation d'orthogonalité, pour tout $k \in \mathbb{N}$:

$$\langle \nabla J(v_k - \rho_k \nabla J(v_k)), \nabla J(v_k) \rangle = 0$$

on trouve le pas optimal, pour tout $k \in \mathbb{N}$ tel que $x_k + 2y_k \neq 0$

$$\rho_k = \frac{1}{20\,(x_k + 2\,y_k)^2}.$$

3. De même, on trouve un pas optimal approché, pour tout $k \in \mathbb{N}$ tel que $x_k + 2y_k \neq 0$

$$\tilde{\rho}_k = \frac{1}{60\,(x_k + 2\,y_k)^2}.$$

4.
```
def GradJ(p): #calcul du gradient
    x , y = p[0],p[1]
    F=np.ones((2 ,1))
    F[1]=2
    return 4*( x +2*y )**3 * F

def pasExa(p) : #calcul du pas exact
    x, y= p[0], p[1]
    #Attention : x +2 y <> 0
    if(x + 2*y==0) :
        print('error x +2 y ==0')
        return(np.nan)

    return 1./(20.*( x +2* y )**2)

def pasApp(p) :#calcul du pas approche
    x, y= p[0], p[1]
    #Attention : x +2 y <> 0
    if(x + 2*y==0) :
        print('error x +2 y ==0')
        return(np.nan)

    return 1./(120.*( x +2* y )**2)

# redefinir une fonction gradient optimal pour ce cas particulier
def gopt_gene(GradJ, v0, tol, NitMax, typePas):#ajout d'un booleen pour
                                    #choisir le pas,
                                    # True : pas exact, False : pas approche
    v = v0
    grad = GradJ(v)
    it=0
    X = [v0]
    while  norm(grad) > tol and it <NitMax:
        #calcule le pas
        if(typePas): #choix du pas
```

```
            rho= pasExa(v)
        else :
            rho = pasApp(v)

        v=v-rho*grad
        X.append(v)
        grad=GradJ(v)
        it+=1

    cvg = norm(grad) <= tol #on verifie si on a convergence
    X = np.array(X) # on transforme la liste en array
    return v,X,it,cvg
tol = 1e-4
NitMax = 3000

v0 = np.ones((2, 1))

print('Gradient optimal : pas optimal exact')
SolGO, TGO, NbrIter0, cvg0 = gopt_gene(GradJ, v0, tol, NitMax, True)
print('v = ', SolGO )
print('J(v) = ' , J(SolGO[0], SolGO[1]))
print('iteration',  NbrIter0)
print('Gradient optimal : pas optimal approche')
SolG1, TG1, NbrIter1, cvg1 = gopt_gene(GradJ, v0, tol, NitMax, False)
print('v = ', SolG1 )
print('J(v) = ' , J(SolG1[0], SolG1[1]))
print('iteration',  NbrIter1)
```

On remarque que le nombre d'itérations est petit pour le pas optimal, et que le pas approché permet une bonne estimation également en peu d'itérations.

5.
```
#On redefinit une fonction gradient a pas fixe pour ce cas particulier
def gf_gene(GradF, v0, rho, tol, NitMax):
    v=v0
    gradfx=GradF(v)
    it=0
    X = [v0]
    while  norm(gradfx) > tol and it <NitMax:
        v=v-rho*gradfx
        X.append(v) # on sauve l'iteration
        gradfx=GradF(v)
        it+=1
    cvg = norm(gradfx) <= tol #on verifie si on a convergence
    X = np.array(X) # on transforme la liste en array
    return v,X,it,cvg


print('Gradient pas fixe')
rho = 0.01
NitMax = 6000
SolG2, TG2, NbrIter2, cvg2 = gf_gene(GradJ, v0, rho, tol, NitMax)
print('v = ', SolG2 )
print('J(v) = ' , J(SolG2[0], SolG2[1]))
print('iteration',  NbrIter2)
```

6. À faire.