Nom et Prénom : DARGEL FLORIAN

Table des matières

1 La fonction

1.1 Présentation de la fonction

Soit $f(x_1, x_2) = (x_1 - 1)^2 + p(x_1^2 - x_2)^2$ avec p = 100 et x^0 = (-1,1)
On doit trouver le minimum global de f sur R^2 et calculer $\nabla f(x)$. on a :

$$\frac{\partial f}{\partial x_1}(x_1, x_2) = 2x_1 - 2 + 400x_1^3 - 400x_1x_2$$

$$\frac{\partial f}{\partial x_2}(x_1, x_2) = -200x_1^2 + 200x_2$$

En résolvant le système d'équation :

$$\begin{aligned} 2x_1 - 2 + 400x_1^3 - 400x_1x_2 &= 0 \\ -200x_1^2 + 200x_2 &= 0 \end{aligned}$$

On obtient $x_1 = 1$ et $x_2 = 1$

On a $(x-1)^2 \geq 0$ et $(x_1^2 - x_2)^2 \geq 0$ et $f(1,1) = 0$ donc $f(x_1, x_2) = (x_1 - 1)^2 + p(x_1^2 - x_2)^2 \geq 0 = f(1,1)$
f admet comme minimum global le point (1,1)

On peut tracer la fonction sur python pour voir à quoi elle ressemble.

Entrée [432]:

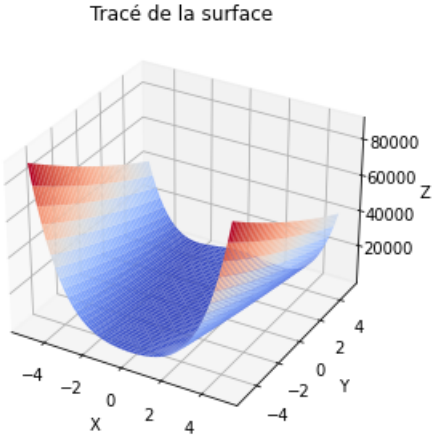
```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d  # Fonction pour la 3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import numpy as np

def fonction1(x1,x2):
    p = 100
    return((x1-1)*(x1-1) + p*(x1**2-x2)**2)

x = np.linspace(-5, 5, 50)
y = np.linspace(-5, 5, 50)
X, Y = np.meshgrid(x, y)
Z=fonction1(X,Y)

fig = plt.figure()
ax = fig.gca(projection='3d')  # Affichage en 3D
ax.plot_surface(X, Y, Z, cmap=cm.coolwarm, linewidth=0)  # Tracé d'une surface
plt.title("Tracé de la surface")
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
plt.tight_layout()
plt.show()
```

executed in 282ms, finished 16:40:09 2021-10-17



Il s'agit d'une fonction non convexe de deux variables. Cette fonction est utilisée comme test pour des problèmes d'optimisation mathématique.

La fonction présente un minimum global à l'intérieur d'une longue vallée étroite de forme parabolique. Si trouver la vallée analytiquement est trivial, on va voir si les algorithmes de recherche du minimum global convergent difficilement.

Le minimum global est obtenu au point (x,y) = (1 ,1) pour lequel la fonction vaut 0.

On fixe sur python les valeurs pour les paramètres et les variables. On choisit comme point de départ $x^0 = [-1, 1]$.

```
Entrée [49]:
p = 100
theta = 0.001
epsilon = 0.001
x1 = -1
x2 = 1
executed in 4ms, finished 12:12:23 2021-10-17
```

On définit une fonction pour le gradient de notre fonction.

```
Entrée [64]:
def grad_f(x1,x2,p):
    return([2*x1 - 2 + 4*p*x1*x1*x1 -4*p*x2*x1,-2*p*x1*x1 + 2*p*x2])
grad_f(x1,x2,p)
executed in 8ms, finished 12:27:19 2021-10-17

Out[64]: [-4, 0]
```

On remarque que pour $x^0 = [-1, 1]$ notre direction de déplacement se dirige vers la gauche jusqu'au point $x_1 = -4$

2 Méthode de la plus forte pente

2.1 Pas θ fixé

On résout par la méthode de la plus forte pente en partant de $x^0 = [-1, 1]$

Entrée [472]:

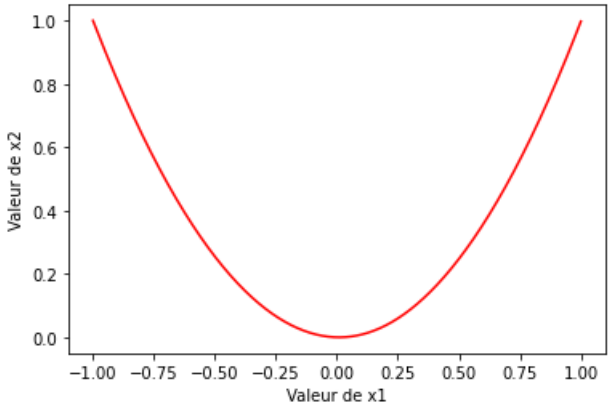
```
def plusfortepente(epsilon,theta,x1,x2,p):
    X = [x1,x2]
    dx1 = grad_f(x1,x2,p)[0]
    dx2 = grad_f(x1,x2,p)[1]
    d = [-(dx1),-(dx2)]
    k = 1
    a = []
    b =[]
    while (np.linalg.norm(d) >= epsilon and k < 100000):
        dx1 = grad_f(x1,x2,p)[0]
        dx2 = grad_f(x1,x2,p)[1]
        d = [-(dx1),-(dx2)]
        x1 = x1 + theta*d[0]
        x2 = x2 + theta*d[1]
        X = [x1,x2]
        a.append(X[0])
        b.append(X[1])
        #print(X0)
        k = k + 1
    plt.plot(a,b,color="red")
    plt.xlabel("Valeur de x1")
    plt.ylabel("Valeur de x2")
    plt.show()
    return(k-1,x1,x2)
```

executed in 7ms, finished 17:19:48 2021-10-17

Entrée [473]:

```
plusfortepente(epsilon,theta,x1,x2,p)
```

executed in 229ms, finished 17:19:48 2021-10-17



Out[473]: (14749, 0.9988835127886635, 0.9977638013863555)

En mettant dans notre algorithme ses conditions, on observe que avec $\epsilon = 0.001$ et $\theta = 0.001$ on se rapproche de la valeur (1,1). Il nous a fallu 14749 itération pour cette méthode.

Sur le graphique on observe que la valeur de X_1 va de -1 à 1. La valeur de X_2 va de 1 à 0 puis de 0 à 1.

2.2 Règle de Polyack : Méthode avec le pas de déplacement θ qui varie

On a un pas θ adaptatif. On va prendre avec un pas $\theta = \frac{\theta_0}{k}$.

Entrée [130]:
```
def methodefortepbis_polyak(epsilon,theta,x1,x2,p):
    X = [x1,x2]
    k = 1
    dx1 = grad_f(x1,x2,p)[0]
    dx2 = grad_f(x1,x2,p)[1]
    d = [-(dx1),-(dx2)]
    thetabis = theta
    while np.linalg.norm(d) > epsilon:
        dx1 = grad_f(x1,x2,p)[0]
        dx2 = grad_f(x1,x2,p)[1]
        d = [-(dx1),-(dx2)]
        theta = thetabis/k
        x1 = x1 + theta*d[0]
        x2 = x2 + theta*d[1]
        X = [x1,x2]
        #print(X)
        k = k + 1
        if (k > 100000):
            break
    return(k-1,x1,x2)
```
executed in 5ms, finished 13:05:25 2021-10-17

Entrée [131]:
```
methodefortepbis_polyak(0.001,0.0170,x1,x2,p)
```
executed in 742ms, finished 13:05:28 2021-10-17

Out[131]: (100000, 0.6325966446069535, 0.39838817390528797)

Pour cette algorithme on augmente la valeur de θ, en prenant $\theta = 0.0170$ car on remarque que cette algorithme converge difficilement vers le minimum global de notre fonction. Elle tend très lentement vers la valeur $(1, 1)$. En prenant les mêmes conditions pour notre algorithme, avec 100 000 itération on a comme valeur environ $(0.63, 0.39)$.

2.3 Règle de SHOR : Méthode avec le pas de déplacement θ qui varie

Pour la régle de SHOR on a un pas θ adaptatif, on va prendre $\theta = \theta_0 - \alpha^k$.

```
Entrée [486]:
def methodefortepbis_SHOR(epsilon,theta,x1,x2,p):
    X = [x1,x2]
    k = 1
    dx1 = grad_f(x1,x2,p)[0]
    dx2 = grad_f(x1,x2,p)[1]
    d = [-(dx1),-(dx2)]
    alpha = 0.95
    thetabis = theta
    while np.linalg.norm(d) >= epsilon:
        dx1 = grad_f(x1,x2,p)[0]
        dx2 = grad_f(x1,x2,p)[1]
        d = [-(dx1),-(dx2)]
        theta = thetabis - alpha**k
        x1 = x1 + theta*d[0]
        x2 = x2 + theta*d[1]
        X = [x1,x2]
        print(X)
        k = k + 1
        if (k > 10000):
            break
    return(k-1,x1,x2)
executed in 14ms, finished 17:46:21 2021-10-17

Entrée [487]:
methodefortepbis_SHOR(epsilon,0.001,x1,x2,p)
executed in 20ms, finished 17:46:22 2021-10-17

[-4.795999999999999, 1.0]
[-38065.668159161585, -3965.891364799999]
[-1.889407053669442e+16, -248177317675.32855]
[-2.194812789831464e+51, -5.808205239757334e+34]
[-3.2682016042409638e+156, -7.445285582858124e+104]
[-inf, -inf]
```

Pour la régle de SHOR on a des résultats différents. Cela doit être le fait que la recherche du minimum global de l'algorithme converge très rapidement. On doit trouver les bonnes valeurs pour les paramètres θ et ϵ. On doit prendre comme valeur un pas de déplacement qui est assez bas.

3 Minimisation unidirectionnelle

3.1 Méthode de Newton

On a une autre méthode pour trouver le point minimum global de notre fonction. Il s'agit de la méthode de Newton.

Méthode de newton $\theta_{k+1} \leftarrow \theta_k - \frac{g'(\theta)}{g''\theta}$

On a :

$$\begin{aligned} g(\theta) &= f(x_1 + \theta d_1, x_2 + \theta d_2) \\ &= (x_1 + \theta d_1 - 1)^2 + p([x_1 + \theta d_1]^2 - (x_2 + \theta d_2))^2 \end{aligned}$$

$$g'(\theta) = 2d_1[x_1 + \theta d_1 - 1] + 2p[2d_1x_1 + 2d_1^2\theta - d_2][x_1^2 + \theta d_1 x_1 + \theta^2 d_1^2 - x_2 - \theta d_2]$$

$$g''(\theta) = 2d_1^2 + 4pd_1^2[x_1^2 + \theta d_1 x_1 + \theta^2 d_1^2 - x_2 - \theta d_2] + [4pd_1x_1 + 4pd_1^2\theta - 2pd_2][dx_1 + 2\theta d_1^2 - d_2]$$

On définit donc en python une fonction pour $g'(\theta)$ et une autre fonction pour $g''(\theta)$ car on en aura besoin pour notre algorithme de newton.

Entrée [277]:
```
d1= 1
d2 = 4
def g_theta_prime(theta,x1,x2,d1,d2,p):
    uprime = 2*d1 *(x1+theta*d1-1)
    vprime = 2*p*(2*d1*x1+2*d1*d1*theta-d2)*(x1*x1+theta*d1*x1+theta*theta*d1*d1-x2-thet
    return(uprime+vprime)

def g_theta_seconde(theta,x1,x2,d1,d2,p):
    useconde = 2*d1*d1
    vseconde = 4*p*d2*d2*(x1*x1+theta*d1*x1+theta*theta*d1*d1-x2-theta*d2)+(2*p*d1*x1+4*
    return(useconde+vseconde)
```
executed in 7ms, finished 15:22:47 2021-10-17

On définit donc l'algorithme de Newton qui nous retourne la valeur de θ

Entrée [362]:
```
def methode_newton(g_theta_prime,g_theta_seconde,theta,x1,x2,d1,d2,epsilon,p):
    condition = 10 * epsilon
    while condition > epsilon:
        thetabis=theta-g_theta_prime(theta,x1,x2,d1,d2,p)/g_theta_seconde(theta,x1,x2,d1,
        condition=np.linalg.norm(thetabis-theta)
        theta=thetabis
    return(theta)
```
executed in 19ms, finished 15:39:01 2021-10-17

Entrée [363]:
```
methode_newton(g_theta_prime,g_theta_seconde,theta,x1,x2,d1,d2,epsilon,p)
```
executed in 6ms, finished 15:39:01 2021-10-17

Out[363]: 0.0005975035235551074

On définit ensuite l'algorithme suivant :

Entrée [506]:
```
def Newton(epsilon,theta,x1,x2,p):
    X0 = [x1,x2]
    dx1 = grad_f(x1,x2,p)[0]
    dx2 = grad_f(x1,x2,p)[1]
    d = [-(dx1),-(dx2)]
    k = 1
    while np.linalg.norm(d) >= epsilon:
        dx1 = grad_f(x1,x2,p)[0]
        dx2 = grad_f(x1,x2,p)[1]
        d = [-(dx1),-(dx2)]
        thetabis = methode_newton(g_theta_prime,g_theta_seconde,theta,x1,x2,d1,d2,epsilon,p)
        x1 = x1 + thetabis*d[0]
        x2 = x2 + thetabis*d[1]
        X0 = [x1,x2]
        print(X0)
        k = k + 1
        if (k > 10000):
            break
    return(k-1,x1,x2)
```
executed in 9ms, finished 18:22:46 2021-10-17

Entrée [507]:
```
Newton(0.001,0.001,x1,x2,p)
```
executed in 1.65s, finished 18:22:48 2021-10-17

```
[-0.9976099859057795, 1.0]
[-0.9980960888900102, 1.0002220808654254]
[-0.9983099763389839, 1.0002941831278482]
[-0.9983648117664321, 1.0003100943888847]
```

Pour l'algorithme de Newton on choisit comme paramètre $\theta = 0.001$, on observe qu'on se rapproche de la valeur $(-1, 1)$ et non $(1, 1)$. On doit modifier le pas de déplacement c'est à dire le paramètre θ et trouver exactement une bonne valeur pour qu'on se rapproche de la valeur du minimum global de notre fonction.

Dans ce cas, par exemple pour un $\theta \geq 1$ on n'est pas proche de la valeur $(-1, 1)$, on a des résultats différents.

```
Entrée [508]: Newton(0.001,1,x1,x2,p)
              executed in 21ms, finished 18:26:24 2021-10-17

[10.945673725764955, 1.0]
[4645839.662205194, -212213.03011123292]
[1.8634454181909837e+29, -2.005499063334528e+22]
[4.8231047822570774e+119, -1.2941363173758275e+90]
[nan, nan]
[nan, nan]

Out[508]: (6, nan, nan)
```

Dans ce cas là à $\theta = 1$, on a choisit un pas de déplacement qui est trop grand.

3.2 Section doré

On utilise cette fois la méthode de la section dorée qui est une méthode utilisée pour localiser un minimum local d'une fonction unidimensionnelle réelle. Cette méthode se base sur un algorithme appelé méthode de dichotomie.

On crée une fonction sur python pour la fonction g en rappelant que $g(\theta) = f(x_1 + \theta d_1, x_2 + \theta d_2) = (x_1 + \theta d_1 - 1)^2 + p([x_1 + \theta d_1]^2 - (x_2 + \theta d_2))^2$

```
Entrée [206]: def g_theta(theta,x1,x2,dx1,dx2,p):
                  return((x1 + theta*dx1-1)**2+p*((x1+theta*dx1)**2-(x2+theta*dx2))**2)
              executed in 10ms, finished 13:50:10 2021-10-17
```

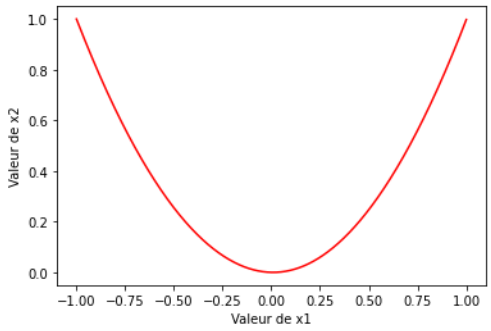
On crée notamment une fonction sur python pour l'algorithme de recherche de dichotomie.

```
Entrée [207]: def section_doree(g_theta,a0,b0,x1,x2,d1,d2,epsilon,p):
                  alpha=(np.sqrt(5)-1)/2
                  a,b=a0,b0
                  c=alpha*a+(1-alpha)*b
                  d=a+b-c
                  while b-a>2*epsilon:
                      if g_theta(c,x1,x2,d1,d2,p)<g_theta(d,x1,x2,d1,d2,p):
                          b,d=d,c
                          c=a+b-d
                      else:
                          a,c=c,d
                          d=a+b-c
                  return((a+b)/2)
              executed in 17ms, finished 13:50:10 2021-10-17
```

On applique ensuite l'algorithme suivant

Entrée [488]:
```
def doree(g_theta,x1,x2,a0,b0,epsi,p):
    k=1
    a = []
    b =[]
    while np.linalg.norm(grad_f(x1,x2,p))>=epsi:
        d= grad_f(x1,x2,p)
        theta=section_doree( g_theta,a0,b0,x1,x2,d1,d2,epsilon,p)
        x1=x1-theta*d[0]
        x2=x2-theta*d[1]
        a.append(x1)
        b.append(x2)
        k=k+1
        if (k > 100000):
            break
    plt.plot(a,b,color="red")
    plt.xlabel("Valeur de x1")
    plt.ylabel("Valeur de x2")
    plt.show()
    return(k-1,x1,x2)
```
executed in 9ms, finished 18:10:52 2021-10-17

Entrée [489]:
```
doree(g_theta,x1,x2,0,1,0.001,p)
```
executed in 1.72s, finished 18:10:55 2021-10-17



Out[489]: (15368, 0.9988832249126427, 0.9977632251237118)

En gardant la même valeur d'epsilon, on observe qu'il faut 15368 itérations pour que l'algorithme recherche le minimum global de notre fonction. On a comme valeur environ (0.998,0.997) ce sont des valeurs qui sont très proches de (1,1). En obsevant le comportement des points x1 et x2, on observe que x_1 va de -1 à 1 et x_2 de 1 à 0 puis de 0 à 1

4 conclusion

Pour l'étude de notre fonction, après avoir appliqué différentes méthodes pour trouver le minimum global, on a des méthodes qui sont plus performantes que d'autres. On a ceux qui convergent assez rapidement et ceux qui convergent très lentement vers le minimum global de la fonction. Le choix du pas de déplacement θ et le point initial x^0 sont très importants pour atteindre le minimum global de notre fonction.