Université Paris-Est-Créteil

Systèmes Distribués et Technologies de la Data Science

Examen pratique :

Optimisation Stochastique

Préparé par :

M^{me}.Imane KHELOUAT

Année 2021-2022

Optimisation de la fonction Schaffer :

Le but de ce TP est de programmer l'optimisation de la fonction dites de schaffer.

L'optimisation sera à faore sur 1 et plusieurs dimension (1,2 et 10) et en utilisant différents algorithmes (Adam, RMSPROP et SG Momentum)

Travail réalisé

Tout d'abord, On commence par importer les bibliothèques nécessaires.

Ensuite, nous implementons les fonctions objectifs et gradient qui renverant respectivement la valeur de la fonction F(X) en un point w et la valeur de son gradient en ce même point..

In [13]:
```
def f(w):
    return 418.9829*D-w.sin(np.sqrt(w)).sum()
```

In [22]:
```
#Calcul de la dérivée de la fonction f
def grad(w):
    return -sin(np.sqrt(w))-1/2*(np.sqrt(w))*cos(np.sqrt(w))
```

Application des algorithmes :

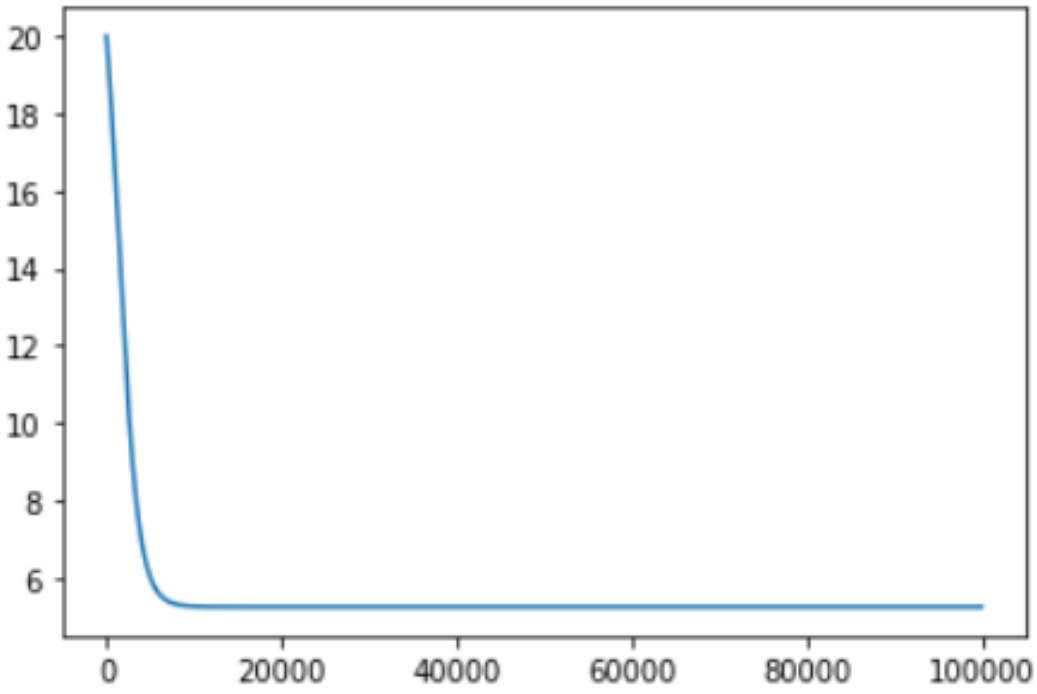
1. Pour commencer, nous utilisons l'algorithme SG Momentum.

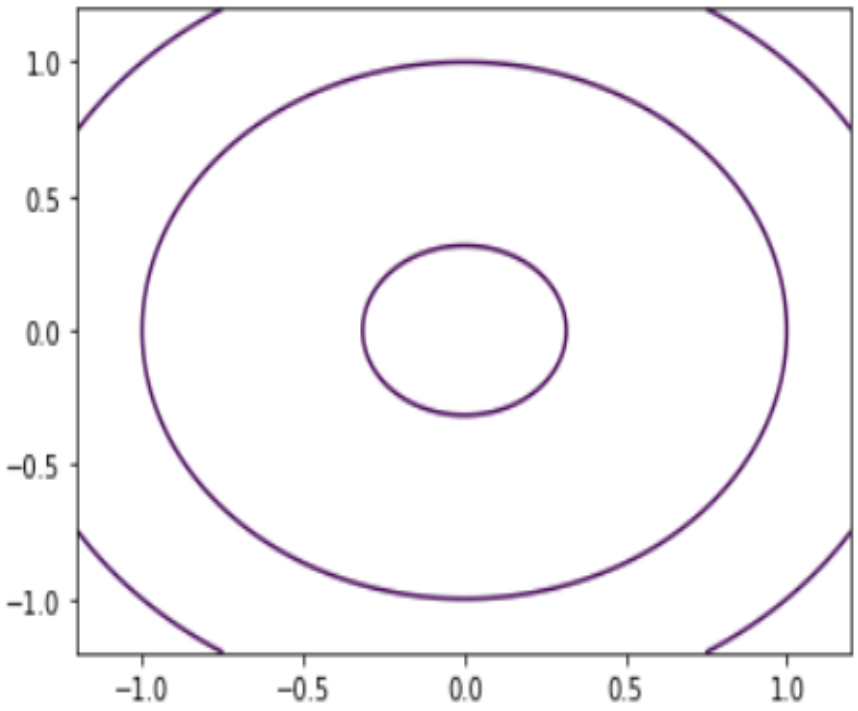
Cet algorithme est une version amélioré de la descente de gradient. En effet, il apporte une nouveauté en améliorant la convergence et en minimisant les probabilités de tomber sur un minimum local. Pour se faire, à chaque étape de calcul de la descente, au lieu de prendre en compte que la dérivé au point w, nous allons prendre en compte aussi les dérivées calculées précedement en faisant des moyennes pondérées de ces valeurs.

```python
#Application de la méthode des
def gd_momentum(w, grad, alpha, beta=0.9, max_iter=10):
    ws = np.zeros(1 + max_iter)
    ws[0] = w
    v = 0
    for i in range(max_iter):
        v = beta*v + (1-beta) * grad(w)
        vc = v/(1+beta**(i+1))
        w = w - alpha * vc
        ws[i+1] = w
    return ws
```

```python
#Appel a la fonction qui calcul la la desecnte de gradient avec momentum
w = 20
gradient = grad(w)
gradient_momentum = gd_momentum(w, grad, alpha= 0.02, beta=0.9, max_iter= 2000)
```

Dans les deux figure qui suit, je vous présente les résultat obtenu lors de l'exécution de cet algorithme :


20
18
16
14
12
10
8
6
0
20000
40000
60000
80000
100000


1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0

On peut donc voir l'évolution de la courbe qui montre l'avancée de l'algorithme momentum jusqu'au minimum local qu'il a trouvé en retournant le point correspondant.

sur deux dimensions

```
def f(w):
    return 418.9829*D-w.sin(np.sqrt(w)).sum()
#Calcul de la dérivée de la fonction f
def grad(w):
    return -sin(np.sqrt(w))-1/2*(np.sqrt(w))*cos(np.sqrt(w))



#Application de la méthode des
def gd_momentum(w, grad, alpha, beta=0.9, max_iter=10):
    ws = np.zeros((1 + max_iter, w.shape[0]))
    ws[0, :] = w
    v = 0
    for i in range(max_iter):
        v = beta*v + (1-beta) * grad(w)
        vc = v/(1+beta**(i+1))
        w = w - alpha * vc
        ws[i+1, :] = w
    return ws
```

Dans la prochaine étape nous exécutons l'algorithme Adam sur la fonction F(x), et ceci en implementant le code suivant :

adam

```
def gd2_adam(w, grad, alpha, beta1=0.9, beta2=0.999, eps=1e-8, max_iter=10):
    w = np.zeros((1 + max_iter))
    w [0] = w
    m = 0
    v = 0
    for i in range(max_iter):
        m = beta1*m + (1-beta1)*grad(x)
        v = beta2*v + (1-beta2)*grad(x)**2
        mc = m/(1+beta1**(i+1))
        vc = v/(1+beta2**(i+1))
        x = x - alpha * m / (eps + np.sqrt(vc))
        xs[i+1] = x
    return xs
```

```
w = np.array([20])
xgrad = gd2_adam(w [0], grad, alpha= 0.005, beta1=0.9 , beta2=0.999, eps=1e-8, max_iter= 10)
print(xgrad)
plt.plot(xgrad)
```

Malheureusement, par contrainte de temps, nous n'avons pas pu corriger à temps des erreurs et donc finir l'exécution et commenter les résultats.