TME 3 - Descente de gradient

Algorithme de descente de gradient

L'algorithme de descente de gradient permet d'approcher le minimum d'une fonction convexe de manière itérée selon la formule de mise-à-jour de $\mathbf{x}^t$ à l'instant t suivant : $\mathbf{x^{t+1}} = \mathbf{x^t} - \epsilon * \nabla \mathbf{f}(\mathbf{x^t})$.
Coder une fonction `optimize(fonc,dfonc,xinit,eps,max_iter)` qui implémente l'algorithme du gradient avec en paramètre : `fonc` la fonction à optimiser, `dfonc` le gradient de cette fonction, `xinit` le point initial, `eps` le pas de gradient, `max_iter` le nombre d'itérations. Cette fonction doit rendre un triplet `(x_histo,f_histo,grad_histo)`, respectivement la liste des points $\mathbf{x}^t$,$f(\mathbf{x}^t)$ et $\nabla f(\mathbf{x}^t)$. Pour cela vous pouvez utiliser une liste pour stocker au fur et à mesure les points, puis transformer chaque liste en un tableau `np.array`.

Optimisation de fonctions

Testez votre implémentation sur les 3 fonctions suivantes :

- en 1d sur la fonction $f(x) = x\ \cos(x)$
- en 1d sur la fonction $-\log(x) + x^2$
- en 2d sur la fonction Rosenbrock (ou banana), définie par : $f(x_1, x_2) = 100*(x_2-x_1^2)^2+(1-x_1)^2$.

Tracer :

- en fonction du nombre d'itérations, les valeurs de f et du gradient de f
- sur un même graphe avec deux couleurs différentes, la fonction f et la trajectoire de l'optimisation (les valeurs successives de $f(\mathbf{x}^t)$)
- la courbe $(t, \log(||\mathbf{x}^t - \mathbf{x}||))$. Que remarquez vous ?

Vous pouvez utiliser le code suivant pour une visualisation 3d (après avoir fait une visulatisation 2d).

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D

def make_grid(data=None,xmin=-5,xmax=5,ymin=-5,ymax=5,step=20):
    if data is not None:
        xmax, xmin, ymax, ymin = np.max(data[:,0]),np.min(data[:,0]),\
                                  np.max(data[:,1]), np.min(data[:,1])
    x, y =np.meshgrid(np.arange(xmin,xmax,(xmax-xmin)*1./step),\
                      np.arange(ymin,ymax,(ymax-ymin)*1./step))
    grid=np.c_[x.ravel(),y.ravel()]
    return grid, x, y
## Grille de discretisation
grid,xx,yy = make_grid(xmin=-1,xmax=1,ymin=-1,ymax=1)
## Affichage 2D
plt.contourf(xx,yy,mafonction(grid).reshape(xx.shape))
fig = plt.figure()
## construction d'un referentiel 3d
ax = fig.gca(projection='3d')
surf = ax.plot_surface(xx, yy, mafonction(grid).reshape(xx.shape),rstride=1,cstride=1,\
          cmap=cm.gist_rainbow, linewidth=0, antialiased=False)
fig.colorbar(surf)
ax.plot(x_histo[:,0],x_histo[:,1],f_histo.ravel(),color='black')
plt.show()
```

Régression logistique

Implémenter la descente de gradient pour la régression logistique.
Tester vos algorithmes sur les données USPS (chiffres manuscrits - en ne choisissant que deux classes parmi les 10) du TME3 de MAPSI `http://www-connex.lip6.fr/~baskiotisn/ARF16/USPS.zip`. Observer la valeur des poids.
Utilisez le code suivant pour lire le fichier de données :

```
def load_usps(filename):
    with open(filename,"r") as f:
        f.readline()
        data =[ [float(x) for x in l.split()] for l in f if len(l.split())>2]
    tmp = np.array(data)
    return tmp[:,1:],tmp[:,0].astype(int)
datax, datay = load_usps("usps.txt")
```