```python
# algorithmes_de_minimisation.py

import numpy as np


##méthode du nombre d'or
phi=(np.sqrt(5)+1)/2
def minimiser(xmin, xmax, seuil_x, f):
    print(xmin, xmax)
    x2=xmin+(xmax-xmin)/(phi+1)
    if abs(x2-xmin)<seuil_x or abs(x2-xmax)<seuil_x:
        return x2
    else:
        y2=f(x2)
        x3 = x2+(xmax-x2)/(phi+1)
        y3=f(x3)
        if y3>y2: #on réduit l'intervalle
            return  minimiser(xmin, x2, seuil_x, f)
        else:
            return  minimiser(x3, xmax, seuil_x, f)

##algorithme de gradient descent
'''variables contient les N paramètres de f
infinitésimaux contient des variations de chacun des paramètres, suffisantes pour approximer la dérivée
le test de convergence se fait sur la norme du gradient
'''


# approximation de la dérivée partielle par rapport à la variable d'index num_deriv à partir de la tangente
def derive_partielle(f, variables, infinitesimaux, num_deriv):
    X1=variables.copy(); X2=variables.copy()
    X1[num_deriv]-=infinitesimaux[num_deriv]; X2[num_deriv]+=infinitesimaux[num_deriv]
    f1=f(X1); f2=f(X2)
    return (f2-f1)/(2*infinitesimaux[num_deriv])

def gradient(f, variables, infinitesimaux, N):
    grad=[]
    for num_deriv in range(N):
        grad.append(derive_partielle(f, variables, infinitesimaux, num_deriv))
    return grad

def norme(vect):
    somme=0
    for direc in vect:
```

```
044|         somme+=direc**2
045|     return np.sqrt(somme)
046|
047| def diff_vect(U, V):
048|     W=[]
049|     for i in range(len(U)):
050|         W.append(U[i]-V[i])
051|     return W
052|
053| def prod_vect(alpha, U):
054|     W=[]
055|     for u in U:
056|         W.append(alpha*u)
057|     return(W)
058|
059| #dans le changement du pas, si les variables changent peu
sans que pour autant le grad soit petit, c'est que l'une des
variable est limitée par les bornes du modèle
060| def minimum(f, variables0, bornes, infinitesimaux,
pas_cv_grad, pas, dessin):
061|     variables=variables0.copy()
062|     N=len(variables)
063|     iteration=1; grad=gradient(f, variables,
infinitesimaux, N)
064|     if dessin:
065|         X, Y= [], []
066|     while iteration<15 and norme(grad)>pas_cv_grad:
067|         if dessin:
068|             X.append(variables[0])
069|             Y.append(variables[1])
070|         print('itération ', iteration, variables, grad,
pas, '\n')
071|         avancement=prod_vect(pas, grad)
072|         variables2=diff_vect(variables, avancement)
073|         for num_dir in range(N):
074|             borne_dir=bornes[num_dir]
075|             variation_var=0
076|             if variables2[num_dir]<borne_dir[1] and
variables2[num_dir]>borne_dir[0]: #on ne change la valeur du
paramètre que s'il reste dans les limites du modèle
077|                 variation_var+=abs(variables[num_dir]-
variables2[num_dir])
078|                 variables[num_dir]=variables2[num_dir]
079|         #amélioration du taux s'apprentissage:
080|         # if variation_var<pas:
081|         #     pas=pas/10
082|         iteration+=1
083|         grad=gradient(f, variables, infinitesimaux, N)
084|     if dessin:
085|         return X, Y
```

```
086|     else:
087|         return variables
088|
089| ##exemple
090| from mpl_toolkits.mplot3d import Axes3D
091| import matplotlib.pyplot as pl
092|
093| def fonction_test(variables):
094|     x, y = variables
095|     return 10*(np.sin(x)**2 + 0.8*np.sin(y)**2)**2
096|
097| def fction_non_vect(x, y):
098|     return 10*(np.sin(x)**2 + 0.8*np.sin(y)**2)**2
099|
100| def grid():
101|     g=np.vectorize(fction_non_vect)
102|     X=np.arange(-1.5, 0.9, 0.1)
103|     Y=np.arange(-1.5, 0.7, 0.1)
104|     X, Y = np.meshgrid(X, Y)
105|     Z = g(X, Y)
106|     return X, Y, Z
107|
108| def plot_3D():
109|     X, Y, Z = grid()
110|     fig=pl.figure()
111|     ax=Axes3D(fig)
112|     ax.plot_surface(X, Y, Z)
113|     pas = 0.01
114|     X, Y = minimum(fonction_test, [-1, -1.5], [[-10, 10],
[-10, 10]], [0.0001, 0.0001], 0.001, pas, True)
115|     n = len(X)
116|     decal = -1
117|     for k in range(n):
118|         x, y = X[k], Y[k]
119|         z = fction_non_vect(x, y)
120|         ax.quiver(x, y, z+decal, 0, 0, -decal, color =
'black', linewidth=1, arrow_length_ratio=0)
121|         ax.scatter(xs=x, ys=y, zs=z+decal, color='red')
122|     pl.show()
123|
124| def plot_contour():
125|     X, Y, Z = grid()
126|     pl.grid()
127|     pl.contour(X, Y, Z, range(30), colors =['blue']*30,
alpha=0.7)
128|     pas = 0.01
129|     X, Y = minimum(fonction_test, [-1, -1.5], [[-10, 10],
[-10, 10]], [0.0001, 0.0001], 0.001, pas, True)
130|     n = len(X)
131|     for k in range(n):
```

```
132|         x, y = X[k], Y[k]
133|         pl.scatter(x, y, color='red', s=20)
134|         if k<n-1:
135|             pl.annotate('', xy=(x, y), xytext=(X[k+1],
Y[k+1]),  arrowprops=dict(arrowstyle="->", lw=3,
mutation_scale=1))
136|     pl.savefig('contour 2D gradient', dpi=300,
bbox_inches='tight')
137|     pl.show()
138|
139| # minimum(fonction_test, [1, 1], [[-2, 2], [-2, 2]], [0.01,
0.01], 0.001, 0.3, True)
140|
141|
142| ##monte Carlo
143| #n est le nombre de valeur prise par variable d'entrée de f
144| # si f est une fonction de 3 variables, alors il y aura
n^^3 valeurs prises
145| #on représente chaque point testé par une liste (écriture
en base n)
146| def monteCarlo(f, bornes, n):
147|     N = len(bornes)
148|     mini = np.inf; meilleur_var='init'
149|     pas = [(bornes[k][1] - bornes[k][0])/n for k in
range(N)]
150|     pos = [0]*N #point de départ
151|     for k in range(n**N):
152|         if k%200==0:
153|             print(pos)
154|         var = [bornes[j][0]+pos[j]*pas[j] for j in
range(N)]
155|         resultat = f(var)
156|         if resultat<mini:
157|             print('minimum de ', resultat, ' obtenu en ',
var, 'correspondant à', pos)
158|             mini = resultat
159|             meilleur_var = var
160|         dizaine = True
161|         j = 0
162|         while dizaine and j<N:
163|             if pos[j]==n-1:
164|                 pos[j] = 0
165|                 j+=1
166|             else:
167|                 pos[j]+=1
168|                 dizaine = False
169|     return mini, meilleur_var
```