Lab 04 The steepest-descent method

Moahmed Stifi

2IA

article
amsmath

Introduction

The objective function is defined as:

$$f(x, y) = (1 - x)^2 + 100 \cdot (y - x^2)^2 \tag{1}$$

The minimum of f is claimed to occur at the point $(1, 1)$.

Proof

To prove that the minimum of f is attained at $(1, 1)$, we need to show that $\nabla f(1, 1) = \mathbf{0}$ and verify the second-order conditions.
The gradient of f is given by:

$$\begin{align}
\nabla f(x, y) &= \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right) \tag{2}\\
&= (-2(1 - x) - 400x(y - x^2), 200(y - x^2)) \tag{3}
\end{align}$$

Now, let's show that $\nabla f(x, y) = \mathbf{0} \Rightarrow (x, y) = (1, 1)$

$$\begin{align}
\nabla f(x, y) = 0 &\Rightarrow (-2(1 - x) - 400x(y - x^2), 200(y - x^2)) = (0, 0) \tag{4}\\
&\Rightarrow (y - x^2) = 0, -2(1 - x) - 400x(y - x^2) = 0 \tag{5}\\
&\Rightarrow (y - x^2) = 0, (1 - x) = 0 \tag{6}\\
&\Rightarrow x = y = 1 \tag{7}
\end{align}$$

The second-order conditions are satisfied if the Hessian matrix of f is positive definite at the critical point $(1, 1)$. The Hessian matrix is given by:

$$Hf(1, 1) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} \tag{8}$$

Evaluate the second-order partial derivatives at $(1, 1)$:

$$\begin{align}
\frac{\partial^2 f}{\partial x^2}(1, 1) &= 802 \tag{9}\\
\frac{\partial^2 f}{\partial x \partial y}(1, 1) &= -400 \tag{10}\\
\frac{\partial^2 f}{\partial y \partial x}(1, 1) &= -400 \tag{11}\\
\frac{\partial^2 f}{\partial y^2}(1, 1) &= 200 \tag{12}
\end{align}$$

Check if the Hessian matrix is positive definite.

$$tr(Hf(1,1)) = \frac{\partial^2 f}{\partial x^2}(1,1) + \frac{\partial^2 f}{\partial y^2}(1,1) = 802 + 200 = 602 > 0 \tag{13}$$

$$det(Hf(1,1)) = 802 \times 200 - (-400) \times (-400) = 400 > 0 \tag{14}$$

Then all eigenvalues are positive, then the Hessian matrix is positive definite.

$\nabla f(1,1) = \mathbf{0}$ and the second-order conditions are satisfied, then $(1,1)$ is the minimum of f.

1 Code

1.1 Python implementation of the steepest-descent method

Given an objective function $f(x,y)$, the Steepest Descent algorithm iteratively updates the current guess $\mathbf{x}_k = (x_k, y_k)$ using the steepest descent direction, which is the negative gradient of the objective function.

Algorithm 1 Steepest Descent Algorithm

Initialize $\mathbf{x}_0$ ▷ Initial guess
while not converged **do**
  Compute the gradient $\nabla f(\mathbf{x}_k)$
  Choose a step size α_k
  Update the guess: $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)$
end while

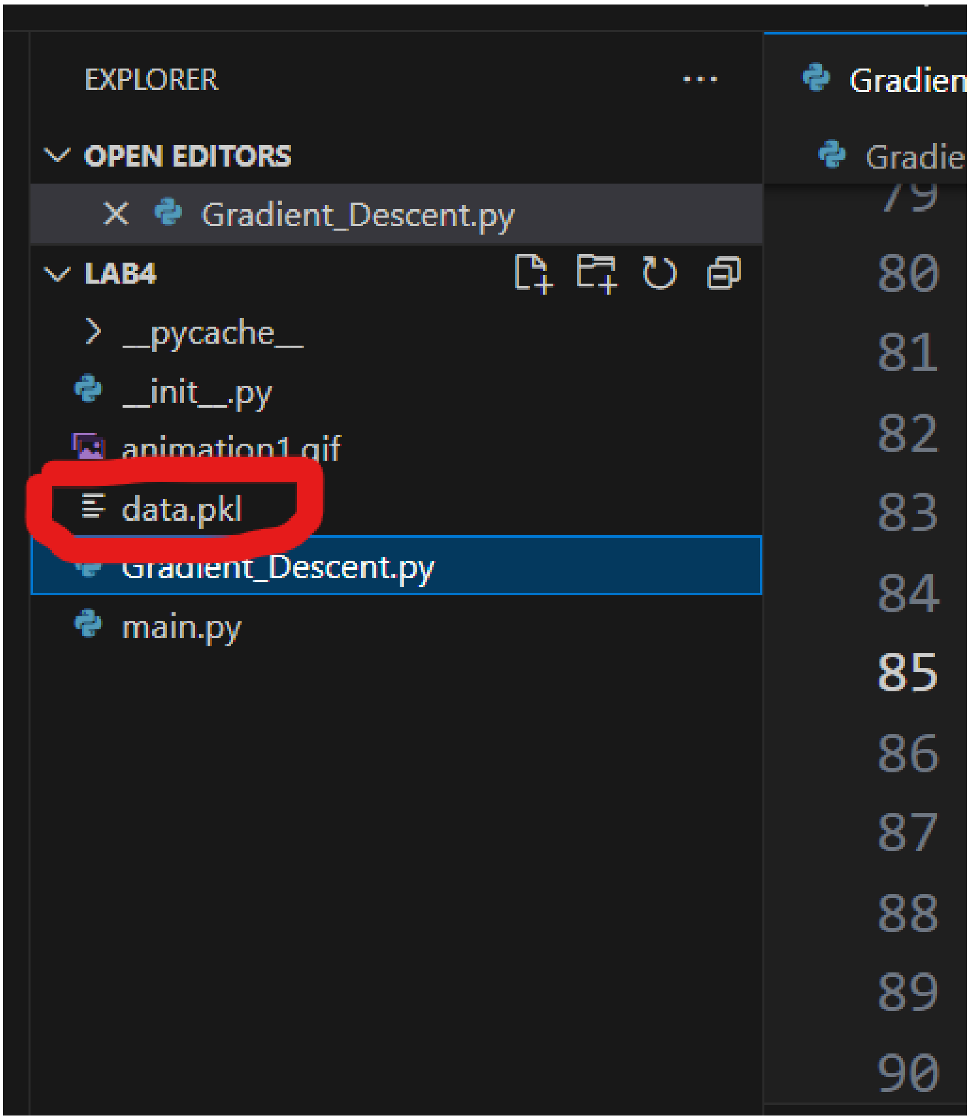
Code file : `Gradient_Descent.py`

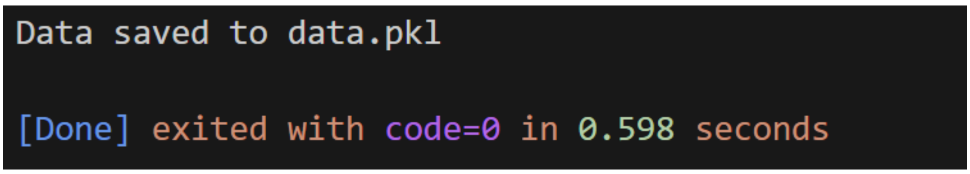
```
import numpy as np
from numdifftools import Gradient
import pickle

def f(arr):
    x,y = arr
    return (1 - x) ** 2 + 100 * (y - x ** 2) ** 2

def Fib(n):
    if n == 0 :
        return [1]
    f = [1,1]
    for i in range(2, n+1):
        f.append( f[-2] + f[-1])
    return f

def Fibonacci(f, x_l, x_u, n):
    fib = Fib(n)
    x1 = x_l + (fib[n-2]/ fib[n])*(x_u - x_l)
    x2 = x_l + (fib[n-1]/ fib[n])*(x_u - x_l)
    while n>1 :
        # print(f"n = {n} | x_l = {x_l} | x_u = {x_u} | x1 = {x1} | x2 = {x2}")
        n = n - 1
        f1 = f(x1)
        f2 = f(x2)
        if f1 < f2 :
            x_u = x2
            x2 = x1
            x1 = x_l + (fib[n-2]/ fib[n])*(x_u - x_l)
```

```
            f2 = f1
            f1 = f(x1)
        elif f1 > f2 :
            x_l = x1
            x1 = x2
            f1 = f2
            x2 = x_l + (fib[n-1]/ fib[n])*(x_u - x_l)
            f2 = f(x2)
        else :
            x_l = x_l + 0.03*(x_u - x_l)
            x1 = x_l + (fib[n-2]/ fib[n])*(x_u - x_l)
            x2 = x_l + (fib[n-1]/ fib[n])*(x_u - x_l)
    return min([x_l , x_u], key = f)
df = Gradient(f)
def GD_(f,df,x0, lr =0.01 ,eps =1e-9 ):
    xk = x0 ; xk1 = x0+ 2*eps
    dfxk = df(xk)
    path = [xk]
    while np.linalg.norm(dfxk) > eps and np.linalg.norm(xk-xk1) > eps :
        xk1 = xk
        xk = xk - lr*dfxk
        dfxk = df(xk)
        path.append(xk)
    return xk, np.array(path)

def GD(f,df,x0, eps =1e-9 ):
    xk = x0 ; xk1 = x0+ 2*eps
    dfxk = df(xk)
    path = [xk]
    def phi(al):
        return f(xk - al*dfxk)

    while np.linalg.norm(dfxk) > eps and np.linalg.norm(xk-xk1) > eps :
        al = Fibonacci(phi, 0, 10, 40)
        xk1 = xk
        xk = xk - al*dfxk
        dfxk = df(xk)
        path.append(xk)
    return xk, np.array(path)

if __name__ == "__main__":
    arr = np.array([3,2.4])
    xop1, path1 = GD(f,df,arr ,eps =1e-9 )
    xop2, path2 = GD_(f,df,arr ,lr=.001,eps =1e-9 )
    data = {
        'GD pas optimal' :{
            "path" : path1,
            "x_optimal" : xop1
        },
        'GD pas fixe' :{
            "path" : path2,
            "x_optimal" : xop2
        }
    }

    pickle_filename = 'data.pkl'
```

```python

    # Save the data dictionary to a Pickle file
    with open(pickle_filename, 'wb') as pickle_file:
        pickle.dump(data, pickle_file)

    print(f'Data saved to {pickle_filename}')
```

1.1.1 Screenshots





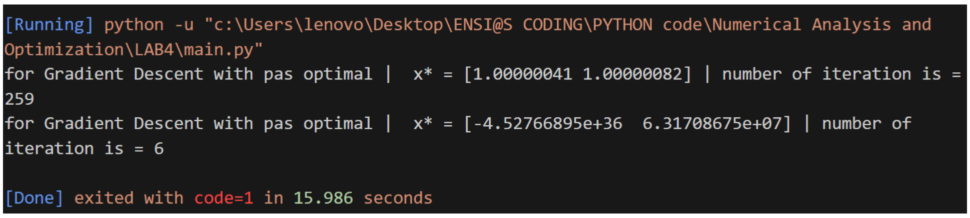
1.2 Animated visualization of gradient descent paths

Code file : main.py

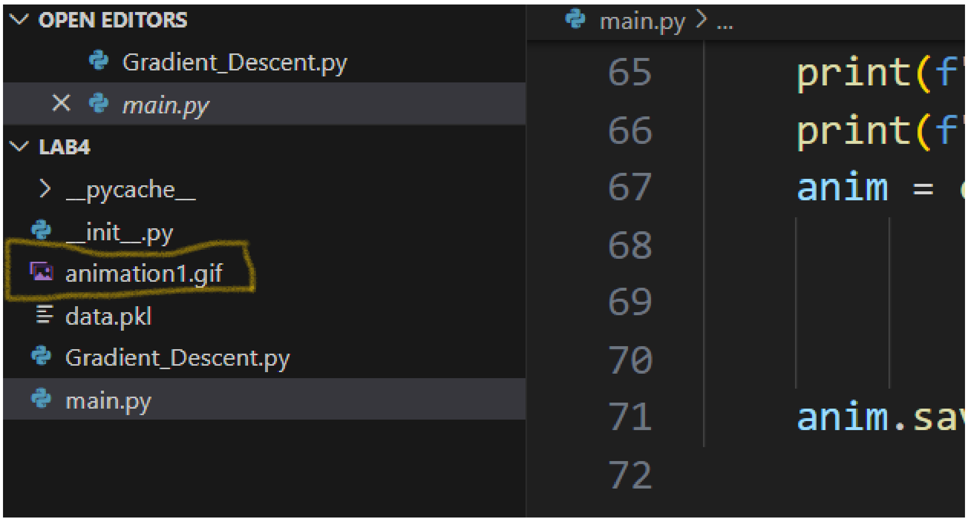
```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.animation import FuncAnimation
from matplotlib.animation import PillowWriter
from Gradient_Descent import f
import pickle
args = {
    'f': f, 'paths' :0
}

def creat_animation(f, paths, minimum,
                     x_lim, y_lim,
                     colors, labels, n_seconds=7,
                       figsize = (14,16)):
    try :
        path_length = max(len(path) for path in paths)
        n_points = 300
        x = np.linspace(*x_lim, n_points)
        y = np.linspace(*y_lim, n_points)
        X, Y = np.meshgrid(x, y)
        Z = f([X,Y])
        fig, ax = plt.subplots(figsize=figsize)
        ax.contour(X, Y, Z, 90, cmap="jet")
        scatters = [ax.scatter(None,
                               None,
                               label=label,
                               c=c) for c, label in zip(colors, labels)]
        ax.legend(prop={"size": 25})

        ax.plot(*minimum, "rD")
        #ax.plot(optimal_points[:,0], optimal_points[:,1], colors)

        def animate(i):
            for path, scatter in zip(paths, scatters):
                scatter.set_offsets(path[:i, :])

            ax.set_title(str(i))
        ms_per_frame = 1000 * n_seconds / path_length
        anim = FuncAnimation(fig, animate, frames=path_length, interval
           ↪ =ms_per_frame)
        plt.show()
    except Exception as e :
        print(e)
    return anim
if __name__ == "__main__":
    pickle_filename = 'data.pkl'

    # Read the data from the Pickle file
    with open(pickle_filename, 'rb') as pickle_file:
        data = pickle.load(pickle_file)

    labels = list(data.keys())
    paths =[
        list(list(data.values())[0].values())[0],
        list(list(data.values())[1].values())[0]
```

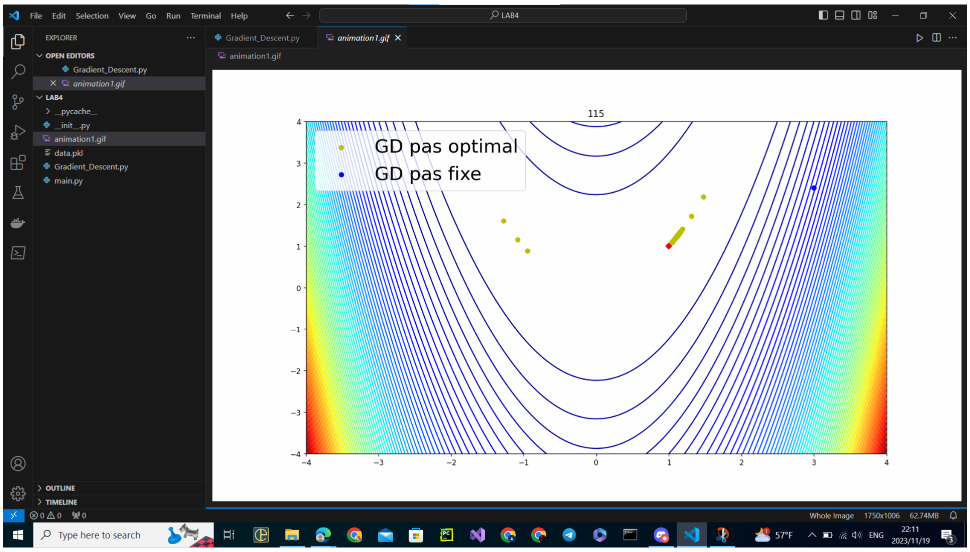
```python
        ]

    optimal_points = [
        paths[0][-1],
        paths[1][-1]
        ]
    paths[ 0] = np.array(list(paths[0][:1000:10]) + list(paths[0][1000::100]))
    colors = ['y', 'b']
    minimum = (1,1)
    x_lim, y_lim = (-4,4),(-4,4)
    print(f"for Gradient Descent with pas optimal |  x* = {optimal_points[0]} | number of iteration is = {len(paths[0])}")
    print(f"for Gradient Descent with pas optimal |  x* = {optimal_points[1]} | number of iteration is = {len(paths[1])}")
    anim = creat_animation(f, paths, minimum,
                        x_lim, y_lim,
                        colors, labels, n_seconds=5,
                         figsize = (14,16))
    anim.save('animation1.gif', writer=PillowWriter(fps=30))
```

1.2.1 Screenshots



```
[Running] python -u "c:\Users\lenovo\Desktop\ENSI@S CODING\PYTHON code\Numerical Analysis and Optimization\LAB4\main.py"
for Gradient Descent with pas optimal |  x* = [1.00000041 1.00000082] | number of iteration is = 259
for Gradient Descent with pas optimal |  x* = [-4.52766895e+36  6.31708675e+07] | number of iteration is = 6

[Done] exited with code=1 in 15.986 seconds
```




115
GD pas optimal
GD pas fixe