Laboratory 1

Variant 5

Group 5

By Maurizio Andres Carrasquero and Uhma Pawel

1 Introduction

In this report, we describe the implementation of a gradient descent algorithm to minimize a given function f(x,y)=2sin(x)+3cos(y)f(x,y) = 2\sin(x) + 3\cos(y)f(x,y)=2sin(x)+3cos(y). The algorithm follows the straight gradient descent approach, iteratively updating the parameters based on the function's gradient until convergence.

1.1 Gradient Descent Algorithm

Gradient descent is an optimization algorithm commonly used for:

- Minimizing cost functions in machine learning.
- Finding local/global minima of complex mathematical functions
- Training deep learning models efficiently

1.2 Advantages

- Simple to implement and computationally efficient
- Works well for convex functions where the global minimum is guaranteed
- Can handle high-dimensional optimization problems

1.3 Disadvantages

- Convergence can be slow depending on the learning rate
- May get stuck in local minima for non-convex functions
- Choosing an inappropriate learning rate can lead to divergence

2 Implementation

Below, we explain the key steps of our implementation along with relevant code snippets

2.1 Functions and its derivatives

The function f(x,y) and its partial derivates are defined as:
$\partial f/\partial x=2\cos(x)$, $\partial f/\partial y=-3\sin(y)$

```
def function(x, y):
    return 2*np.sin(x)+3*np.cos(y)

def function_der_fx(x):
    """
    derivative of F(X,Y) with respect to Y
    """
    return 2*np.cos(x)

def function_der_fy(y):
    """
    derivative of F(X,Y) with respect to X
    """
    return -3 * np.sin(y)
```

2.2 Gradient descent step calculation

At each iteration, we update x and y using:
$xnew=x-\alpha(\partial fx/\partial x)$,$ynew=y-\alpha(\partial f/\partial y)$

```
def next_x(x,learning_rate):
    return x - function_der_fx(x) * learning_rate

def next_y(y, learning_rate):
    return y - function_der_fy(y) * learning_rate
```

2.3 Gradient descent algorithm

The iterative process stops when the function value changes by less than a defined tolerance or when the maximum iterations are reached

```
def gradient_descent(initial_guess, learning_rate, tol=1e-6, max_iter=1000):
    """
    Gradient descent algorithm

    Parameters:
    - initial_guess
    - learning_rate
    - tol: tolerance
    - max_iter: maximum number of iterations

    """
    iterations = 0
    guess = initial_guess
    diff = np.inf
    path = [initial_guess]
    while iterations < max_iter and diff > tol:
        iterations += 1
        new_guess = next_point(guess,learning_rate)
        diff = abs(function(guess[0],guess[1]) - function(new_guess[0],new_guess[1]))
        path.append(new_guess)
        guess = new_guess

    return guess, iterations, np.array(path)
```

2.4 Visualization of convergence

A 3D plot is used to visualize the path of gradient descent

```
def visualize(path):
    """
    Creates 3D plot of the function
    """
    x = np.linspace(-5, 5, 100)
    y = np.linspace(-5 , 5, 100)
    X, Y = np.meshgrid(x, y)
    Z = function(X, Y)
    fig = plt.figure(figsize=(10, 8))
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_surface(X, Y, Z, cmap=cm.viridis, alpha=0.6)
    x_path = path[:, 0]
    y_path = path[:, 1]
    z_path = function(x_path, y_path)

    ax.scatter(x_path, y_path, z_path, color='r', s=50, label="Iteration Points")
    ax.plot(x_path, y_path, z_path, color='r', linestyle='--', label="Descent Path")

    ax.set_title("Gradient Descent Path")
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.set_zlabel("f(x,y)")
    ax.legend()

    plt.show()
```

3 Discussion

3.1 Impact of different initial vectors

The starting point significantly affects the number of iterations required for convergence.

3.2 Impact of different learning rates

The choices of learning rate ´α ´ greatly influences performance:

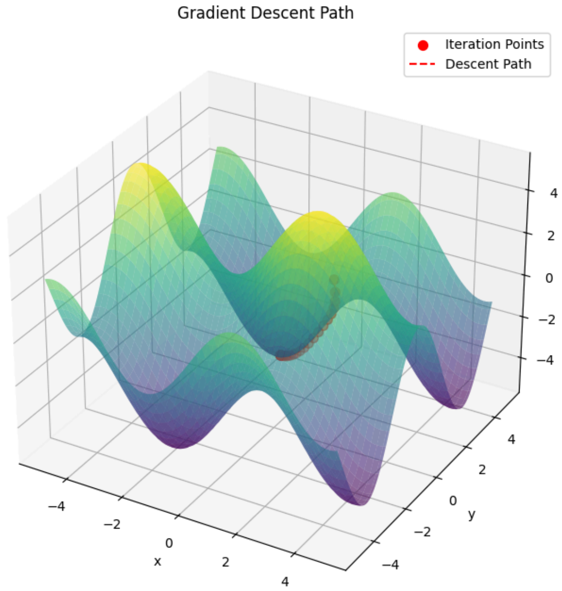
- Small α (0.1) = Converges slowly but safely
- Medium α (0.3-0.5) = Achieves faster convergence
- large α (0.8 or higher) = May lead to divergence

This confirms that the learning rate must be chosen carefully to balance speed and stability

3.3 Test cases and observations

We tested with different initial points and learning rates to observe variations in convergence speed. The results are plotted, showing different descent paths

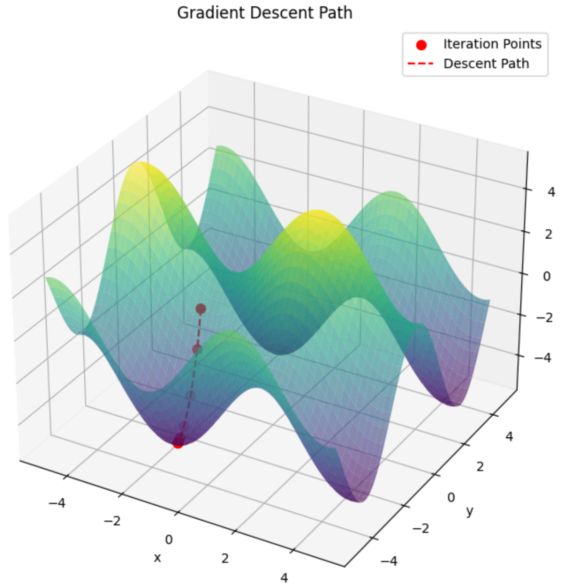
```
Test Case 1: Initial Point = [1, 2], Learning Rate = 0.1
  → Minimum found at: [np.float64(-1.5695759022618714),
np.float64(3.141591772551776)]
  → Iterations: 40
```



In this case the learning mode is moderate, allowing the algorithm to progress in a controlled manner without abrupt desviations

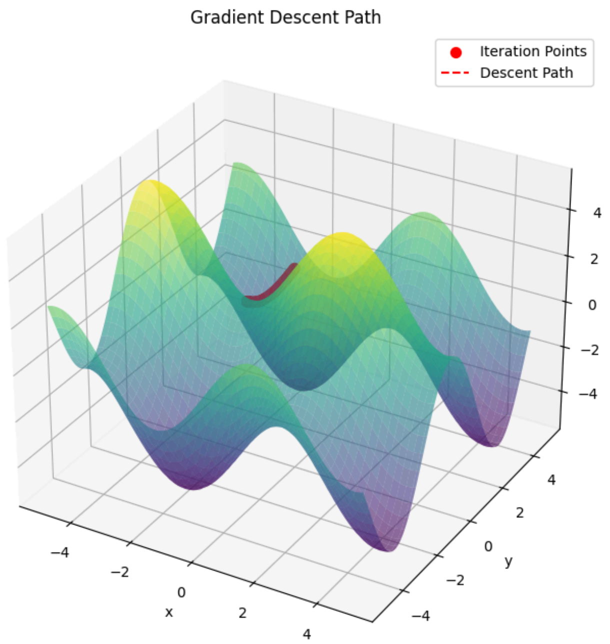
```
Test Case 2: Initial Point = [-2, -1], Learning Rate = 0.2
  → Minimum found at: [np.float64(-1.5713751684887345),
np.float64(-3.1415334814093248)]
  → Iterations: 13
```

In case 2 The convergence is fast without large jumps



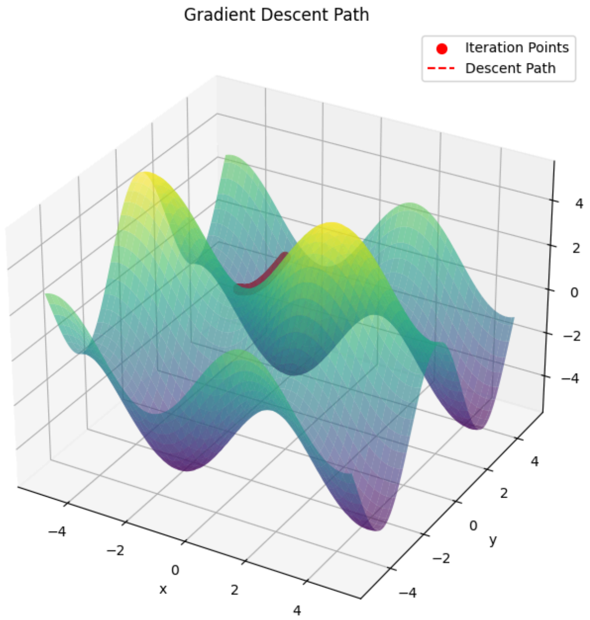

```
Test Case 3: Initial Point = [3, -3], Learning Rate = 0.5
  → Minimum found at: [np.float64(4.71238898038469),
np.float64(-3.1418655377017464)]
  → Iterations: 9
```

In case 3 the learning rate is high but not excessive, allowing efficient convergence



```
Test Case 4: Initial Point = [0, 0], Learning Rate = 0.01
  → Minimum found at: [np.float64(-1.5659045116042036), np.float64(0.0)]
  → Iterations: 298
```

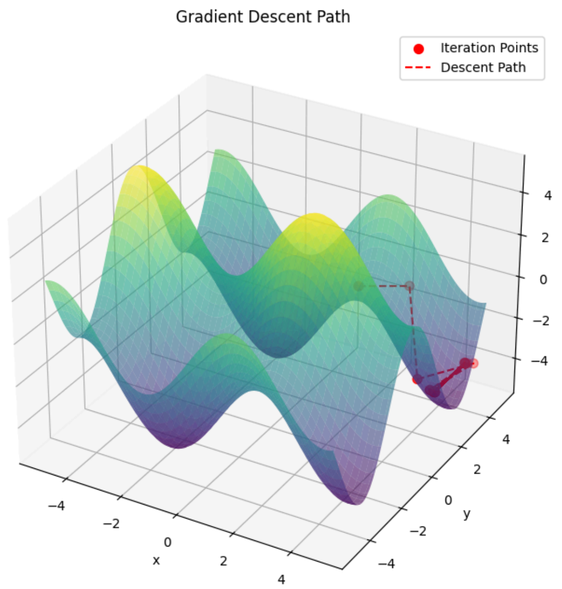
In case 4, although the minimum is correctly found, the algorithm takes too long to reach it due to small steps sizes



Ç

```
Test Case 5: Initial Point = [2, 2], Learning Rate = 0.8
  → Minimum found at: [np.float64(4.713022779559163),
np.float64(4.168330944991768)]
  → Iterations: 15
```

In case 5, the learning rate is high, but in this case, it does not cause instability



4 Conclusion

From this task, we learned that:

- Gradient descent effectively minimize functions, but its performance depends on initialization and parameter tuning
- Initial points affect convergence speed, starting closer to the minimum helps
- Learning rate selection is crucial: small rates lead to slow convergence, while larger ones can cause instability
- Visualization helps analyze optimization behavior and path traversal

4.1 Challenges faced

- Turning the learning rate to prevent divergence
- Handling cases where the algorithm converges too slowly
- Implementing visualization effectively

4.2 Future improvements

- Implementing momentum-based gradient descent to speed up convergence
- Using adaptive learning rates to improve efficiency
- Applying this approach to more complex functions and real-world datasets