Laboratory 1

Variant 5

Group 103

By Bora ILCI & Kaan Emre KARA

Introduction

This project implements the gradient descent algorithm to minimize the function $f(x,y) = 2sin(x) + 3cos(y)$. The implementation visualizes the optimization path in 3D space, allowing us to observe how different initial points and learning rates affect the convergence behavior.

Gradient descent is a first-order iterative optimization algorithm used to find the local minimum of a differentiable function. The algorithm works by taking steps proportional to the negative of the gradient of the function at the current point.

Advantages:

- Simple to implement and understand
- Works well for convex optimization problems
- Requires only first-order derivatives

Disadvantages:

- Convergence speed is sensitive to the learning rate
- May get trapped in local minima for non-convex functions
- Can exhibit zigzagging behavior in narrow valley

Implementation

The gradient descent implementation iteratively updates the position by moving in the direction of the negative gradient:

```
def gradient_descent(initial_guess, learning_rate, tol=1e-6, max_iter=1000):
    x, y = initial_guess
    iterations = 0
    path = [(x, y)]
    for _ in range(max_iter):
        grad_x = 2 * np.cos(x)
        grad_y = -3 * np.sin(y)
        grad_norm = np.sqrt(grad_x**2 + grad_y**2)
        if grad_norm < tol:
            break
        x = x - learning_rate * grad_x
        y = y - learning_rate * grad_y
        path.append((x, y))
        iterations += 1
    return (x, y), iterations, path
```

The algorithm computes the gradient at each step, checks if it's below the tolerance (convergence criterion), and updates position by moving in the negative gradient direction scaled by the learning rate.

Visualization

The code implements an innovative multi-page visualization system that allows navigation between different test cases using arrow keys:

```
def visualize_multi_page(results_list):
    fig = plt.figure(figsize=(16, 7))
    n_pages = len(results_list)
    axes_list = []
    surfs = []

    # Creating all subplots
    for i in range(n_pages):
        ax1 = fig.add_subplot(1, 2, 1, projection="3d")
        ax2 = fig.add_subplot(1, 2, 2, projection="3d")
        # ...
```

Each page shows two different viewing angles of the same optimization run, allowing better understanding of the 3D descent path.

Test Cases

The implementation includes four test cases:

```
# Example usage with two different initial guesses:
initial_guess_1 = [2.0, 2.0]
initial_guess_1_2 = [-2.0, 1.0]
learning_rate_1 = 0.1
# Example usage with two different learning rates:
initial_guess_2 = [2.0, 2.0]
initial_guess_2_2 = [2.0, 2.0]
learning_rate_2 = 1
learning_rate_2_2 = 0.5
```

Discussion

Impact of Different Initial Vectors

The initial position significantly affects the convergence path and sometimes the final solution:

1. **Starting from [2.0, 2.0] vs [-2.0, 1.0] with the same learning rate (0.1)**:
    - Different initial positions result in different paths through the function landscape
    - Despite different starting points, the algorithm should converge to the same local minimum if it's within the basin of attraction
    - The number of iterations required varies based on how far the initial position is from the nearest minimum
    - The function f(x,y) = 2*sin(x)* + *3*cos(y) has multiple local minima due to its periodic nature, so different starting points can lead to different local minima

Impact of Different Learning Rates

The learning rate dramatically affects convergence speed and stability:

1. **Learning rate comparison (0.1 vs 0.5 vs 1.0) from the same starting point [2.0, 2.0]**:
    - **Small learning rate (0.1)**: Takes more iterations but follows a smoother path
    - **Medium learning rate (0.5)**: Better balance between speed and stability
    - **Large learning rate (1.0)**: Converges in fewer iterations but risks overshooting or oscillating around the minimum

    A larger learning rate can significantly reduce the number of iterations needed, but if too large, it may cause the algorithm to diverge or oscillate. This demonstrates the importance of learning rate tuning in gradient-based optimization.

Mathematical Insights

For the function $f(x, y) = 2sin(x) + 3cos(y)$, the gradient is $[2cos(x), -3sin(y)]$:

- Local minima occur when the gradient equals zero: $cos(x) = 0$ and $sin(y) = 0$
- This happens at $x = \pm\pi/2 + n\pi$ and $y = n\pi$
- The global minimum is approximately at $x = -\pi/2$ and $y = 0$ (or equivalent points due to periodicity)

Conclusion

This implementation demonstrates how gradient descent works for multivariate optimization problems.

Key learnings include:

1. The delicate balance between learning rate and convergence speed/stability
2. The importance of initial point selection in functions with multiple local minima
3. How periodic functions create multiple equivalent minima in the solution space

Several challenges were encountered:

1. Visualizing 3D descent paths clearly without text overlapping
2. Creating an interactive multi-page visualization system
3. Selecting appropriate test cases to demonstrate algorithmic behavior

The implementation could be enhanced by:

1. Adding adaptive learning rate strategies (e.g., line search, momentum)
2. Including more diverse initial points to better explore the function landscape
3. Implementing different descent algorithms for comparison (conjugate gradient, Newton's method)
4. Adding contour plots to better visualize the function topology
5. Improving text label positioning to avoid overlapping in visualizations