# Optimization Algorithms Project Documentation

## Introduction

This project was completed under using the algorithms taught in class and through the provided presentation materials. The project focuses on implementing and visualizing various first-order and second-order optimization algorithms, applied to the benchmark Rosenbrock function. The goal is to demonstrate the convergence behaviors of these algorithms using both 2D and 3D plots. Every Subfolder had the code and the final image, if the code is executed then the plots can be seen.

**Team Members:**

- Shubhendu Shukla - 2022BCD0054
- Priyanaka Kumari - 2022BCD0057
- Yogesh Kumar Saini - 2022BCD0052

---

## Folder Structure

We have divided the code into two main folders:

### 1. First-Order Optimization Algorithms Folder:

Contains implementations for the following algorithms:

- Adam Algorithm
- Adagrad Algorithm
- Basic Gradient Descent
- EDMA (Exponential Decay Method for Adaptation)
- Gradient Descent with Armijo Line Search
- Linear Regression
- Logistic Regression
- Momentum
- Nesterov Gradient Descent
- RMSProp
- Subgradient Method

### 2. Second-Order Optimization Algorithms Folder:

Contains implementations for the following algorithms:

- BFGS (Broyden-Fletcher-Goldfarb-Shanno)
- Conjugate Gradient Method
- Newton’s Method

# Algorithm Implementations

Each of the algorithms was implemented from scratch, and the Rosenbrock function was used as the benchmark function for testing the optimization methods. Below are descriptions of each algorithm:

## First-Order Optimization Algorithms

1. **Adam Algorithm:**
   - **Description**: Adam is an adaptive learning rate optimization algorithm that computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients.
   - **Use Case**: Suitable for problems with large datasets and high-dimensional parameter spaces.
2. **Adagrad Algorithm:**
   - **Description**: Adagrad adjusts the learning rate for each parameter individually by considering the historical gradient information, allowing it to converge faster for sparse data.
   - **Use Case**: Best for sparse gradient problems, such as those involving natural language processing.
3. **Basic Gradient Descent:**
   - **Description**: A straightforward algorithm that updates parameters by moving in the opposite direction of the gradient, reducing the objective function's value step-by-step.
   - **Use Case**: Useful for simple convex problems with a well-defined gradient.
4. **EDMA (Exponential Decay Method for Adaptation):**
   - **Description**: EDMA adapts the learning rate by applying an exponential decay function, gradually reducing the rate over time to allow for fine-tuning as the optimization progresses.
   - **Use Case**: Effective in problems where the objective function has a sharp descent in the beginning and a more gradual reduction in later stages.
5. **Gradient Descent with Armijo Line Search:**
   - **Description**: This method incorporates a line search using the Armijo condition to find an appropriate step size, helping avoid poor convergence.
   - **Use Case**: Applied to situations where an optimal step size is difficult to find manually.
6. **Linear Regression:**
   - **Description**: A supervised learning algorithm that finds the best-fit line for a given set of data points by minimizing the squared error.
   - **Use Case**: Best for predicting continuous values based on linear relationships between variables.
7. **Logistic Regression:**
   - **Description**: A classification algorithm that predicts binary outcomes using a logistic function, often used in classification tasks.
   - **Use Case**: Commonly used for binary classification problems like spam detection.

8. **Momentum:**
    - **Description**: Momentum helps accelerate gradient descent by adding a fraction of the previous update to the current update, smoothing out oscillations and improving convergence speed.
    - **Use Case**: Effective for problems with steep gradients or noisy data.
9. **Nesterov Gradient Descent (Nesterov Accelerated Gradient):**
    - **Description**: This method improves upon momentum by calculating the gradient at the "look-ahead" point, leading to faster and more accurate updates.
    - **Use Case**: Suitable for faster convergence when dealing with convex optimization problems.
10. **RMSProp:**
    - **Description**: RMSProp adjusts the learning rate by dividing the gradient by the running average of the squared gradient, preventing the learning rate from getting too large or too small.
    - **Use Case**: Ideal for non-stationary objectives, particularly in deep learning.
11. **Subgradient Method:**
    - **Description**: A variant of gradient descent that works for nondifferentiable functions by replacing the gradient with a subgradient.
    - **Use Case**: Used in optimization problems where the objective function is not differentiable.

---

## Second-Order Optimization Algorithms

1. **BFGS (Broyden-Fletcher-Goldfarb-Shanno):**
    - **Description**: BFGS is an iterative method for solving unconstrained nonlinear optimization problems. It approximates the Hessian matrix to speed up convergence.
    - **Use Case**: Often used for problems with a well-behaved objective function and when an accurate Hessian matrix is expensive to compute.
2. **Conjugate Gradient Method:**
    - **Description**: This method solves large systems of linear equations and quadratic optimization problems by generating a sequence of conjugate directions.
    - **Use Case**: Commonly used for large-scale optimization problems and systems of linear equations.
3. **Newton’s Method:**
    - **Description**: Newton’s method uses the second derivative (Hessian matrix) to determine the next parameter update, often leading to faster convergence than gradient descent.
    - **Use Case**: Effective when the objective function is smooth and convex, and the Hessian matrix is computationally feasible.

---

# Benchmark Function: Rosenbrock Function

The **Rosenbrock function** is used as a benchmark for the optimization algorithms. It is defined as:

f(x,y)=(1−x)2+100(y−x2)2f(x, y) = (1 - x)^2 + 100(y - x^2)^2f(x,y)=(1−x)2+100(y−x2)2

This function is commonly used to test optimization algorithms because it has a narrow, curved valley with a global minimum inside the valley. It presents challenges for optimization methods, particularly in terms of convergence speed and robustness.

---

# Example Implementation: RMSProp Optimizer

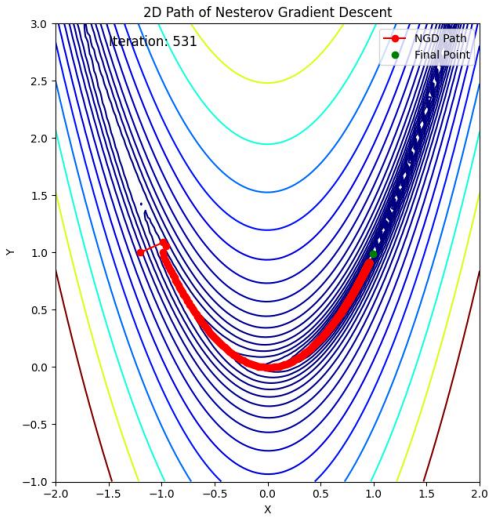
Below is an example of the **RMSProp** implementation that optimizes the Rosenbrock function. The algorithm uses an adaptive learning rate based on the square of past gradients, improving convergence behavior.

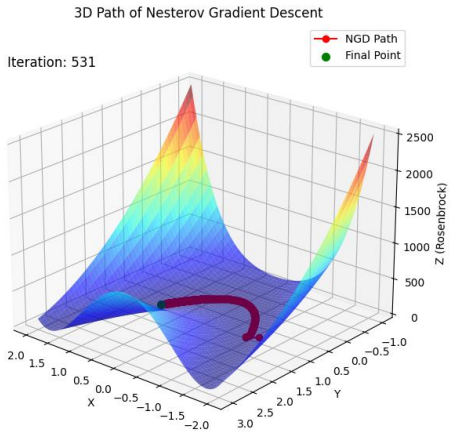
```
python
Copy code
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.animation import FuncAnimation, PillowWriter

class RMSProp:
    def __init__(self, learning_rate=0.01, decay=0.99, epsilon=1e-8,
max_iterations=300, tol=1e-6):
        self.learning_rate = learning_rate
        self.decay = decay
        self.epsilon = epsilon
        self.max_iterations = max_iterations
        self.tol = tol
        self.path = []

    def optimize(self, initial_point, gradient_func):
        current_point = initial_point
        num_iter = 0
        grad_sq = np.zeros_like(current_point)
        consecutive_small_steps = 0

        while num_iter < self.max_iterations:
            self.path.append(current_point.copy())
            grad = gradient_func(current_point)

            if np.linalg.norm(grad) < self.tol:
                print(f"Convergence achieved at iteration {num_iter} due to
small gradient norm.")
                break

            grad_sq = self.decay * grad_sq + (1 - self.decay) * grad**2
            adjusted_grad = grad / (np.sqrt(grad_sq) + self.epsilon)
            next_point = current_point - self.learning_rate * adjusted_grad

            if np.linalg.norm(next_point - current_point) < self.tol:
```

```
                consecutive_small_steps += 1
            else:
                consecutive_small_steps = 0

            if consecutive_small_steps >= 5:
                print(f"Convergence achieved at iteration {num_iter} due to
consecutive small parameter changes.")
                break

            current_point = next_point
            num_iter += 1

        return current_point

def rosenbrock_gradient(point):
    x, y = point
    grad_x = -2 * (1 - x) - 400 * x * (y - x**2)
    grad_y = 200 * (y - x**2)
    return np.array([grad_x, grad_y])

initial_point = np.array([-1.2, 1.0])
rmsprop_optimizer = RMSProp(learning_rate=0.1, decay=0.99)
result = rmsprop_optimizer.optimize(initial_point, rosenbrock_gradient)
path = np.array(rmsprop_optimizer.path)
x_vals, y_vals = path[:, 0], path[:, 1]

x = np.linspace(-2, 2, 100)
y = np.linspace(-1, 3, 100)
X, Y = np.meshgrid(x, y)
Z = (1 - X)**2 + 100 * (Y - X**2)**2

fig = plt.figure(figsize=(14, 6))
ax2d = fig.add_subplot(1, 2, 1)
ax3d = fig.add_subplot(1, 2, 2, projection='3d')

ax2d.contour(X, Y, Z, levels=50)
ax2d.plot(x_vals, y_vals, color='red', label='Optimization Path')
ax2d.set_xlabel('X')
ax2d.set_ylabel('Y')
ax2d.set_title('2D Contour Plot of Optimization Path')

ax3d.plot_surface(X, Y, Z, cmap='viridis', edgecolor='none', alpha=0.6)
ax3d.plot(x_vals, y_vals, rmsprop_optimizer.path, zs=0, color='red',
label='Optimization Path')
ax3d.set_xlabel('X')
ax3d.set_ylabel('Y')
ax3d.set_zlabel('Z')
ax3d.set_title('3D Surface Plot of Optimization Path')

plt.show()
```

This code visualizes the path of optimization on both a 2D contour plot and a 3D surface plot. It tracks the optimization process as the algorithm progresses towards the global minimum of the Rosenbrock function.

---




# Visualization

We used **Matplotlib** for 2D and 3D visualizations to plot the optimization paths for each algorithm. The following visualizations were generated:

1. **2D Contour Plot**: Shows the optimization path over the contour map of the Rosenbrock function.
2. **3D Surface Plot**: Displays the optimization path over the 3D surface of the Rosenbrock function, making it easier to understand how the algorithm converges to the global minimum.

---

# Conclusion

This project successfully implemented and visualized the optimization algorithms discussed. The algorithms' convergence behavior on the Rosenbrock function was observed through 2D and 3D plots. Further work could involve testing on more complex functions and comparing these optimization algorithms on real-world datasets.