Optimization Algorithms from Scratch Project Report

Contributors

- **Sumit Hulke** (2023BCD0026)
- **Suraj Sanjay Harlekar** (2023BCD0038)
- **Aryan Patil** (2023BCD0047)

---

1. Introduction

This project focuses on implementing a complete suite of **optimization algorithms from scratch** using only **NumPy**.
Our goal is to understand:

- How different optimization algorithms work internally
- How they behave on classical test functions
- How fast they converge
- How their convergence trajectories differ when visualized on contour plots

The project is structured modularly, with algorithms defined in Python modules and experiments performed using Jupyter Notebooks.

---

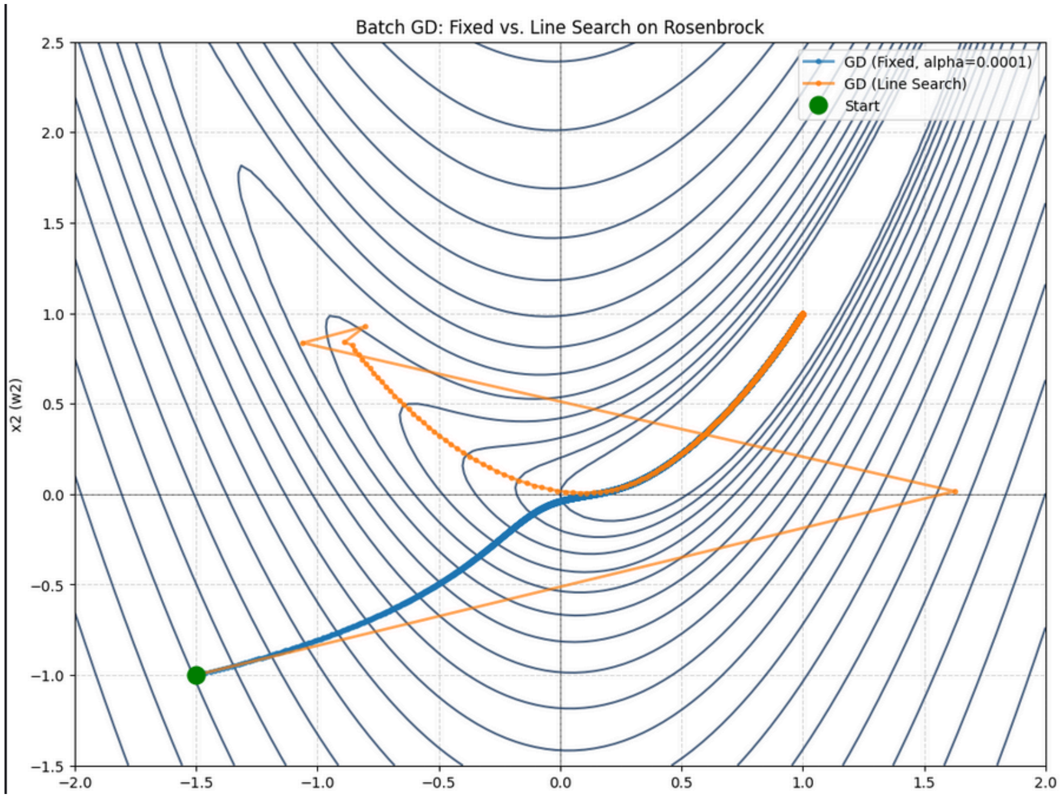
2. Algorithms Implemented

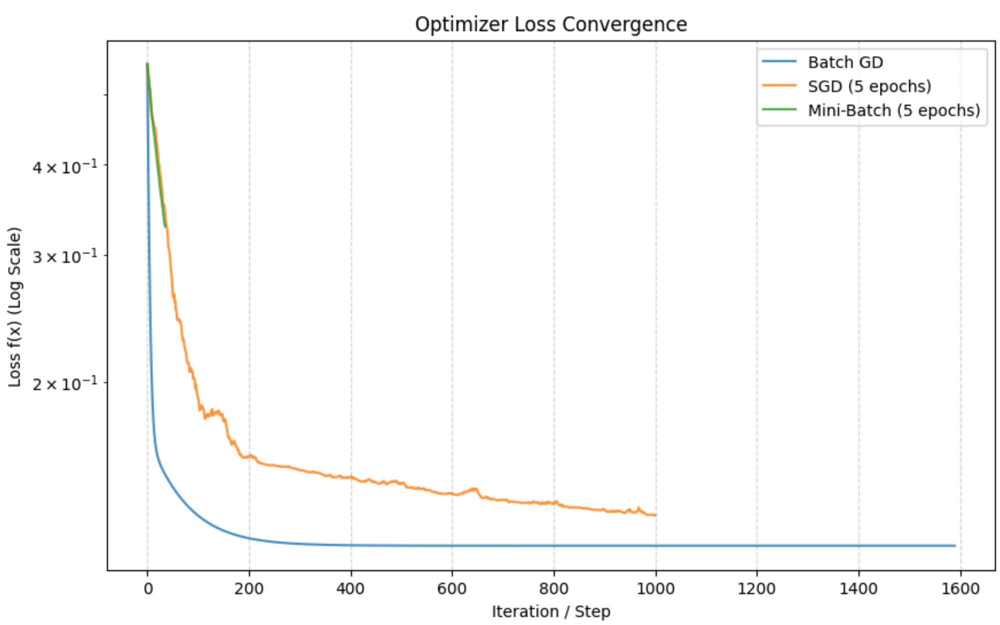
Below is a brief explanation of every algorithm covered in the project.

2.1 First-Order Optimization Methods

Gradient Descent (GD)

- Moves opposite to the gradient direction.
- Highly dependent on learning rate.





Stochastic Gradient Descent (SGD)

- Uses one randomly sampled data point.

- Noisy updates → helps escape local minima.

Mini-Batch SGD

- Uses small batches for smoother, faster convergence.

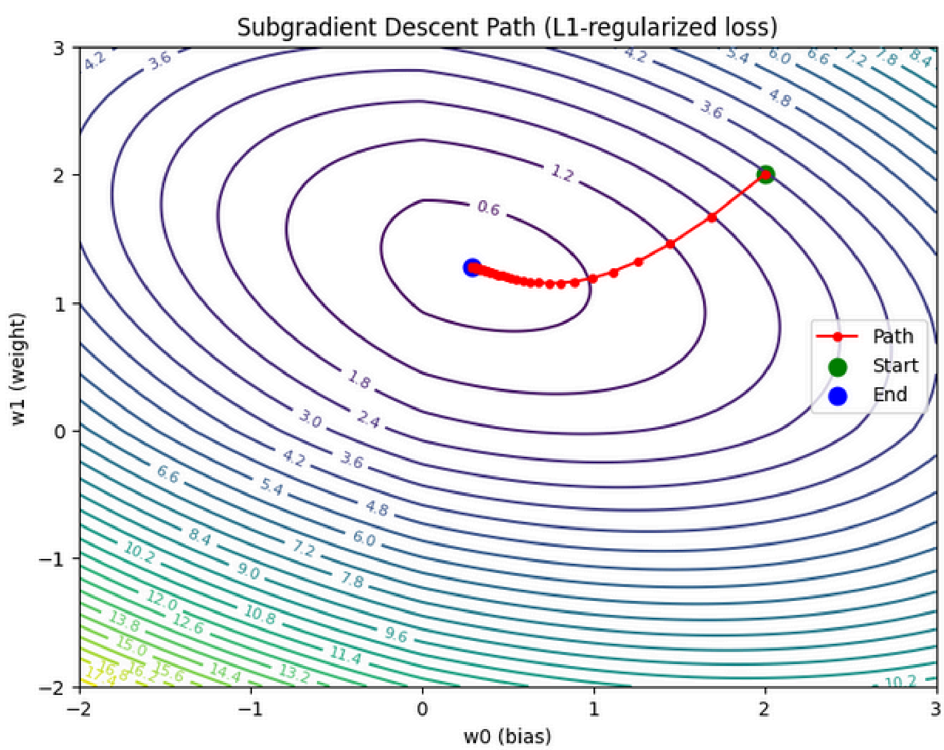
2.2 Momentum-Based Methods

Momentum Gradient Descent

- Adds a velocity term to gradient updates.
- Reduces oscillations, especially in curved valleys.

Nesterov Accelerated Gradient (NAG)

- Computes gradient at a look-ahead position.
- Results in faster and more stable convergence.



2.3 Adaptive Learning Rate Methods

Adagrad

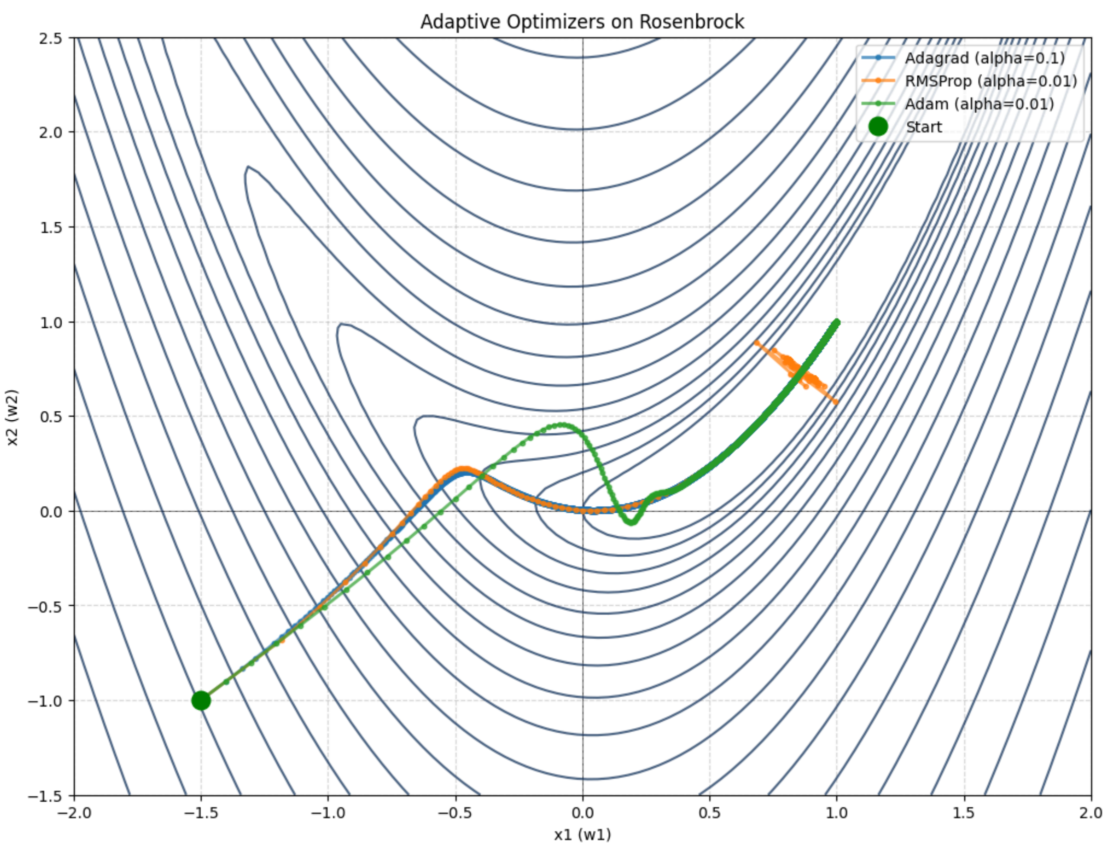
- Adapts learning rates per-parameter based on history.
- Great for sparse features.

RMSProp

- Uses moving average of squared gradients.
- Prevents learning rate from shrinking too fast.

Adam

- Combines Momentum + RMSProp.
- Default choice in deep learning.



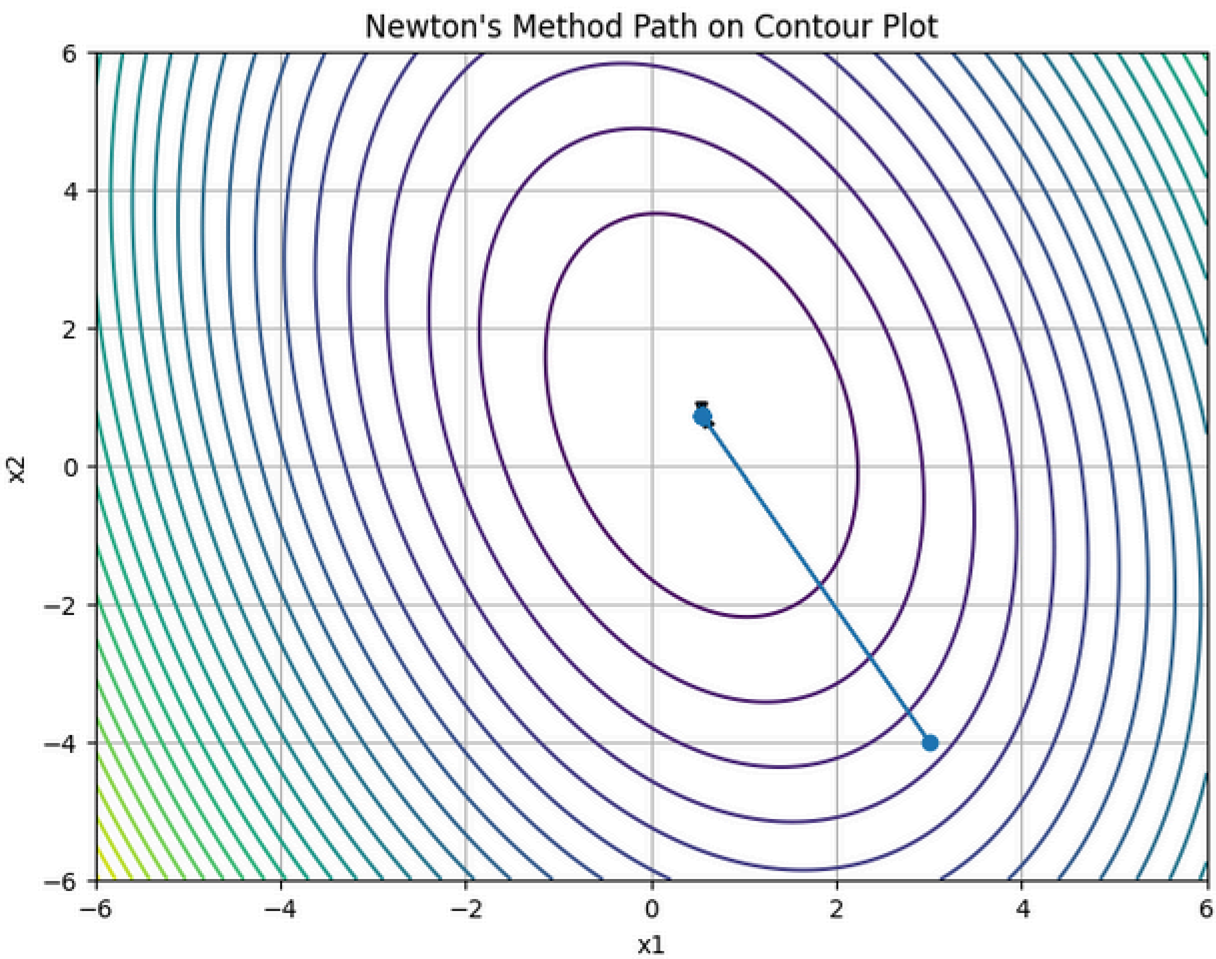
2.4 Second-Order Optimization Methods

Newton’s Method

- Uses Hessian matrix to take curvature-aware steps.
- Extremely fast near minima.

Damped Newton Method

- Adds damping factor when the Hessian is not positive definite.



2.5 Quasi-Newton Methods

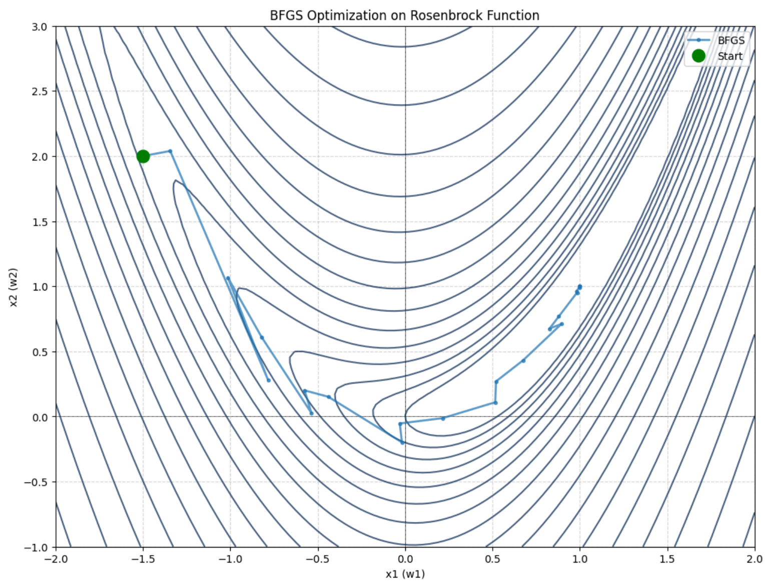
BFGS

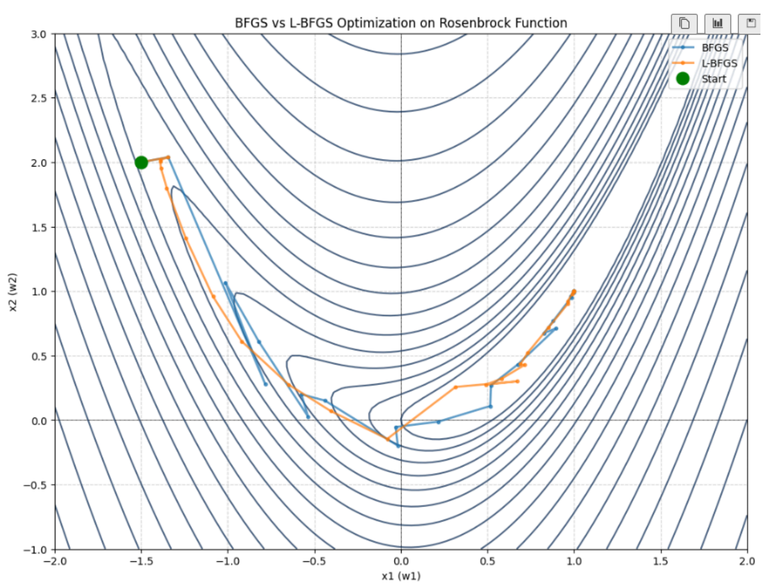
- Approximates the inverse Hessian using gradient differences.

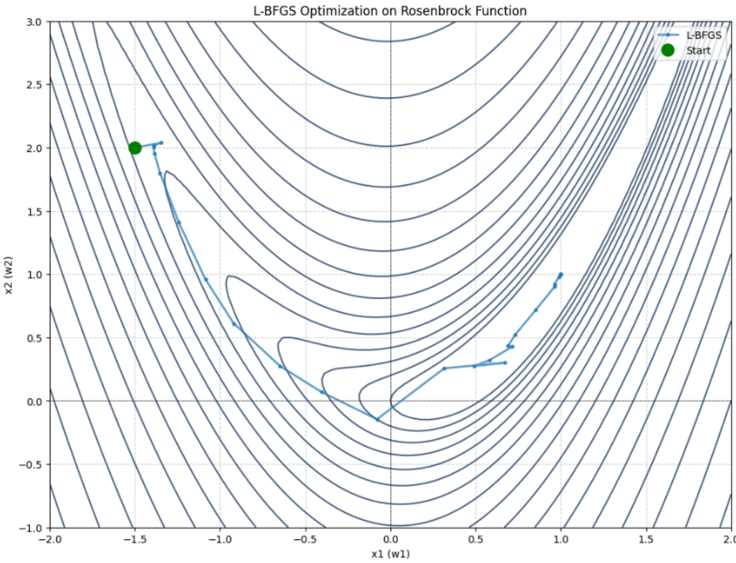
- Faster than vanilla GD, without requiring true Hessians.

L-BFGS

- Limited memory BFGS (stores only a few past updates).
- Suitable for high-dimensional problems.

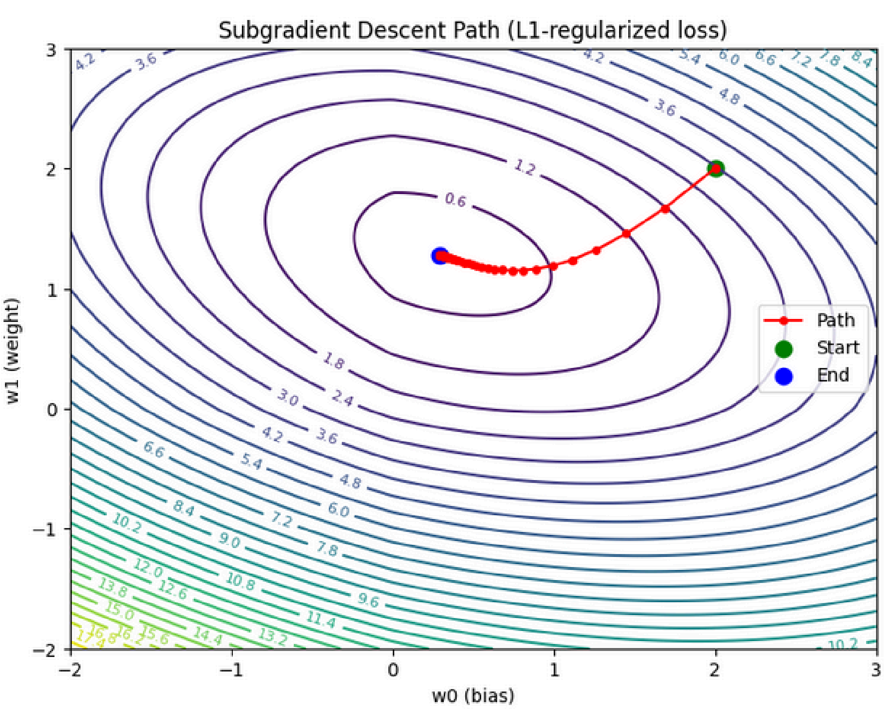






2.6 Non-Differentiable Optimization

Sub-Gradient Method

- Works when objective is not differentiable (e.g., L1 norm).
- Uses any valid sub-gradient at non-smooth points.
- 

---

3. Regression Models Implemented

We also implemented regression models (from scratch + using optimizers).

3.1 Linear Regression

- Uses closed-form solution and GD.
- Dataset: **California Housing**.

3.2 Ridge Regression (L2 Regularization)

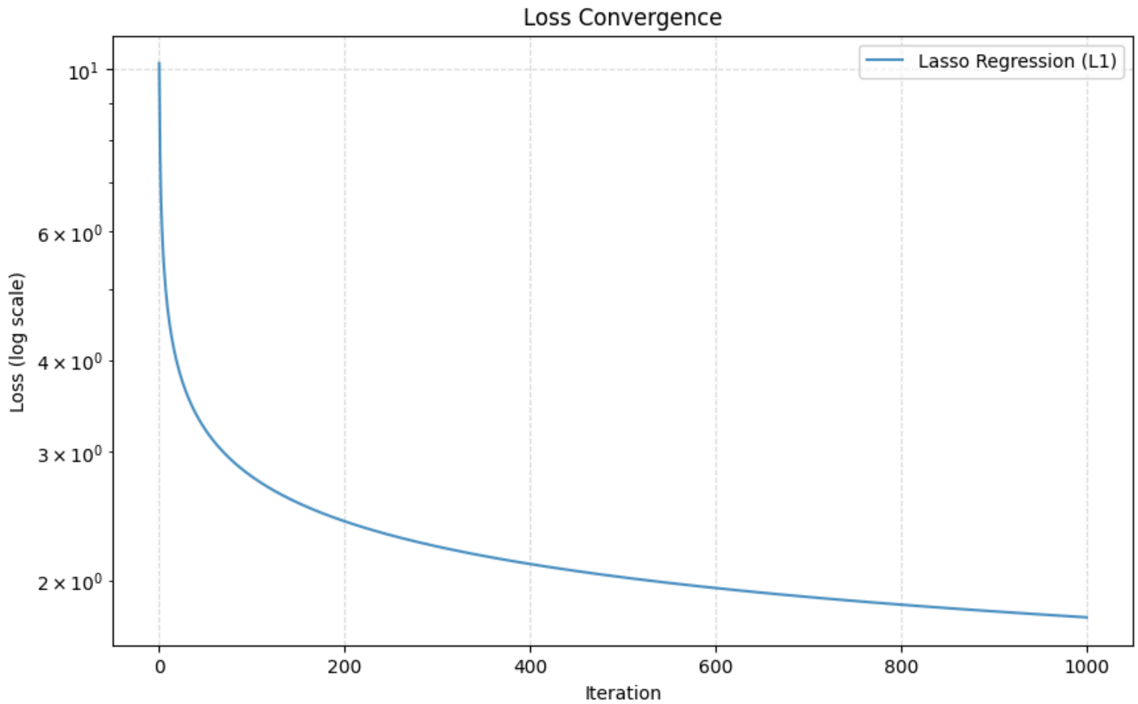
- Adds L2 penalty to control model complexity.

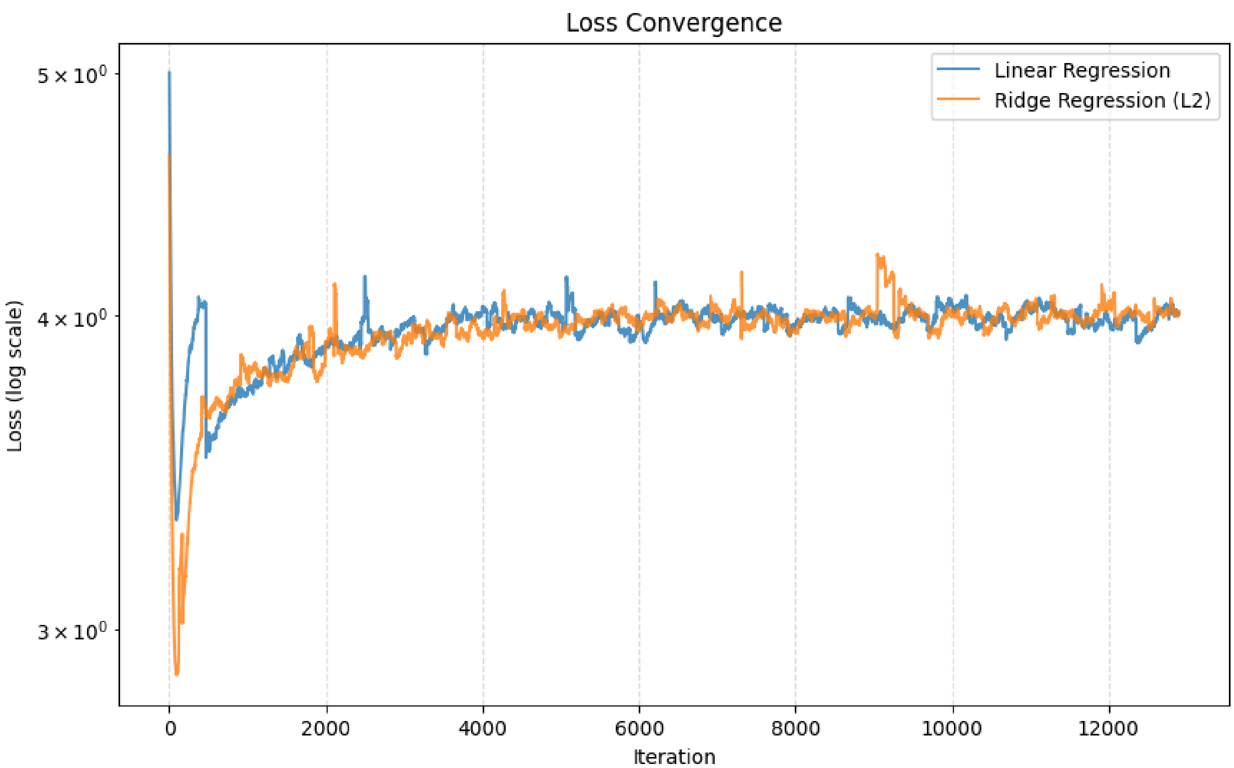
3.3 Lasso Regression (L1 Regularization)

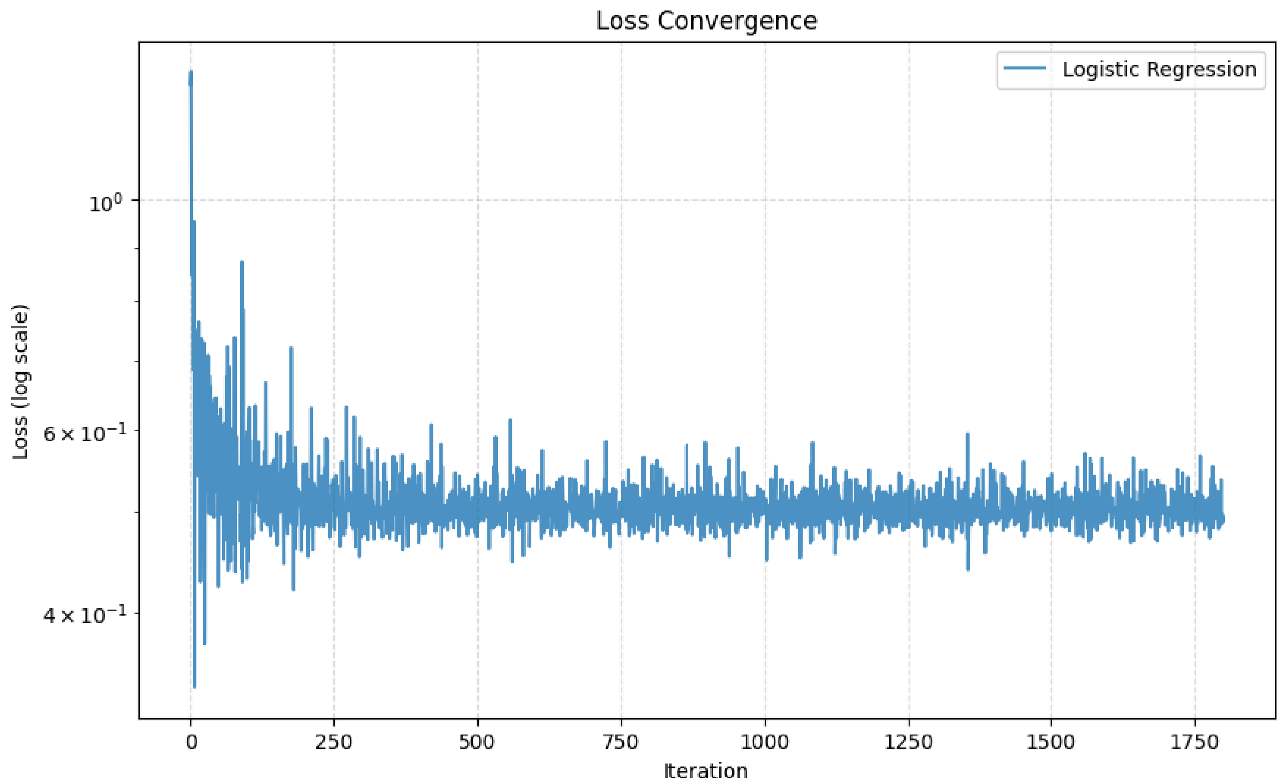
- Uses subgradient method.
- Produces sparse parameter vectors.

3.4 Logistic Regression

- Binary classification using sigmoid + cross-entropy loss.
- Dataset: **Breast Cancer Wisconsin**.


Loss Convergence
Lasso Regression (L1)
Loss (log scale)
Iteration
10^1
6×10^0
4×10^0
3×10^0
2×10^0
0
200
400
600
800
1000


Loss Convergence
Linear Regression
Ridge Regression (L2)
Loss (log scale)
Iteration
5×10^0
4×10^0
3×10^0
0
2000
4000
6000
8000
10000
12000



---

4. Project Structure Summary

- **optimizers/** → All algorithm implementations
- **notebooks/** → Experiments, contour plots, convergence graphs
- **utils/** → Test functions, plotting helpers

---

5. Conclusion

This project provided deep insight into:

- Gradient-based optimization
- Curvature-aware second-order methods
- Adaptive learning rate strategies
- Quasi-Newton approximations

- Regression model training from scratch