Mathematical Derivation

ML-HW1
2025 Fall

313553058 Ian Tsai

1 Overview

- Mathematical derivations for LSE, Steepest Descent (L1 Regularization), and Newton's Method.
- Code implementation mapping to show where each formula appears in regression_solver.py.
- Unit test validation, ensuring each function works correctly.
- GitLink: **https://github.com/IaTsai/ML-HW1**

2 Formula Mapping

- **LSE formula** → `LeastSquaresEstimation()`
- **L1 Regularized Gradient update in Steepest Descent** → `SteepestDescent_L1()`
- **Newton's Method** → `NewtonMethod()`
- **Error computation** → `ComputeError()`

3 Closed-form Least Squares Estimation (LSE) Approach

3.1 Mathematical Derivation

To prevent overfitting, we introduce a regularization term λI into the Least Squares Estimation (LSE) objective function. The goal is to find the coefficient vector $\mathbf{w}$ that minimizes the squared error between the predicted and actual values, while also penalizing large weights.

$$J(\mathbf{w}) = \|\mathbf{A}\mathbf{w} - \mathbf{b}\|^2 + \lambda\|\mathbf{w}\|^2$$

Expanding the squared norm:

$$J(\mathbf{w}) = (\mathbf{A}\mathbf{w} - \mathbf{b})^T(\mathbf{A}\mathbf{w} - \mathbf{b}) + \lambda\mathbf{w}^T\mathbf{w}$$

Taking the derivative with respect to $\mathbf{w}$:

$$\nabla J(\mathbf{w}) = 2\mathbf{A}^T\mathbf{A}\mathbf{w} - 2\mathbf{A}^T\mathbf{b} + 2\lambda\mathbf{w}$$

Setting $\nabla J(\mathbf{w}) = 0$ for minimization:

$$\mathbf{A}^T\mathbf{A}\mathbf{w} + \lambda\mathbf{I}\mathbf{w} = \mathbf{A}^T\mathbf{b}$$

Solving for $\mathbf{w}$:

$$\boxed{\mathbf{w} = (\mathbf{A}^T\mathbf{A} + \lambda\mathbf{I})^{-1}\mathbf{A}^T\mathbf{b}}$$

3.2 Corresponding Code (regression_solver.py)

```
def LeastSquaresEstimation(A, b, lambda_val):
    ATA = A.Transpose().Multiply(A)
    ATb = A.Transpose().Multiply(b)
    ATA_lI = ATA.Add(Matrix.Identity(A.n), lambda_val)
    return ATA_lI.Inverse().Multiply(ATb)
```

4 Steepest Descent Method (L1 Regularization)

4.1 Mathematical Derivation

In standard gradient descent, the update rule is:

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta \nabla J(\mathbf{w}^{(k)})$$

For L2 regularization, the gradient is:

$$\nabla J(\mathbf{w}) = 2\mathbf{A}^T(\mathbf{A}\mathbf{w} - \mathbf{b}) + 2\lambda\mathbf{w}$$

However, in L1 Regularization, we introduce an absolute penalty on $\mathbf{w}$ instead of a squared penalty. The cost function becomes:

$$J(\mathbf{w}) = \|\mathbf{A}\mathbf{w} - \mathbf{b}\|^2 + \lambda\|\mathbf{w}\|_1$$

Taking the derivative:

$$\nabla J(\mathbf{w}) = 2\mathbf{A}^T(\mathbf{A}\mathbf{w} - \mathbf{b}) + \lambda \cdot \operatorname{sign}(\mathbf{w})$$

where

$$\operatorname{sign}(w_i) = \begin{cases} 1, & w_i > 0 \\ -1, & w_i < 0 \\ 0, & w_i = 0 \end{cases}$$

Thus, the L1 regularized steepest descent update rule becomes:

$$\boxed{\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta \left(2\mathbf{A}^T(\mathbf{A}\mathbf{w}^{(k)} - \mathbf{b}) + \lambda \cdot \operatorname{sign}(\mathbf{w}^{(k)})\right)}$$

4.2 Corresponding Code (regression_solver.py)

```
def SteepestDescent_L1(A, b, lr=0.00001, tol=1e-6,
                       max_iter=1000000, lambda_val=0):
    w = Matrix.Zeros(A.n, 1)
    for _ in range(max_iter):
        residual = A.Multiply(w).Add(b, -1)
        gradient = A.Transpose().Multiply(residual).Add(
            Matrix([[lambda_val * np.sign(w.matrix[i][0])]
                    for i in range(A.n)]), 1)
        w_new = w.Add(gradient, -lr)
        if w_new.NormDifference(w) < tol:
            break
        w = w_new
    return w
```

5 Newton's Method

5.1 Mathematical Derivation

Newton's Method finds $\mathbf{w}$ using the Hessian matrix H and gradient $\nabla J(\mathbf{w})$. For the standard least squares objective (without regularization):

$$J(\mathbf{w}) = \|\mathbf{A}\mathbf{w} - \mathbf{b}\|^2$$

Taking the gradient:

$$\nabla J(\mathbf{w}) = 2\mathbf{A}^T(\mathbf{A}\mathbf{w} - \mathbf{b})$$

Taking the Hessian:

$$H = \nabla^2 J(\mathbf{w}) = 2\mathbf{A}^T\mathbf{A}$$

Newton's update step:

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - H^{-1}\nabla J(\mathbf{w}^{(k)})$$

For quadratic functions, Newton's method converges in one step:

$$\boxed{\mathbf{w} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}}$$

5.2 Corresponding Code (regression_solver.py)

```
def NewtonMethod(A, b):
    H = A.Transpose().Multiply(A)
    grad = A.Transpose().Multiply(b)
    return H.Inverse().Multiply(grad)
```

6 Summary

The three methods implemented in this homework demonstrate different approaches to polynomial regression:

1. **Closed-form LSE with L2 Regularization**: Direct solution with regularization to prevent overfitting
2. **Steepest Descent with L1 Regularization**: Iterative method that promotes sparsity in the solution
3. **Newton's Method**: Fast convergence for quadratic objectives without regularization

Each method has its trade-offs in terms of computational cost, convergence rate, and regularization effects.