Hongjue Zhao
3190104515
hongjue0830@zju.edu.cn

Optimization Midterm

Hongjue Zhao

Contents

1 Requirements

Based on small datasets *abalone*, *bodyfat* and *housing*, training a *ridge regression* model with algorithms Gradient Descent, Conjugate Descent, and quasi-Newton method respectively.

Requirements:

1. Implement algorithms (Gradient Descent, Conjugate Descent, and quasi-Newton method) using C/C++ programming language.
2. In this answer sheet, please briefly introduce
   (a) The ridge regression;
   (b) The training algorithms that you implement;
   (c) Experimental settings;
   (d) Experimental results.

   Please illustrate with diagrams, the Mean Squared Error of different algorithms at each iteration, and analyze the results in light of what you have learned in this course.
3. Compress the pdf file of this answer sheet and the C/C++ program into a zip file, and submit it.

Remark. Datasets can be downloaded from link.

2 Introduction to Ridge Regression

Suppose that we have input vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_p)^\top$ and want to predict a real-valued output y. The linear regression model has the from

$$f(\boldsymbol{x}) = \beta + \sum_{j=1}^{p} x_j w_j = \langle \boldsymbol{w}, \boldsymbol{x'} \rangle \,, \tag{2.1}$$

in which $\boldsymbol{w} = (\beta, w_1, \ldots, w_p)^\top \in \mathbb{R}^{p+1}$ and $\boldsymbol{x'} = (1, x_1, \ldots, x_p)^\top \in \mathbb{R}^{p+1}$. The linear model either assumes that the regression function is linear, or that the linear model is reasonable approximation. Here $\boldsymbol{w}$ is unknown parameter vector.

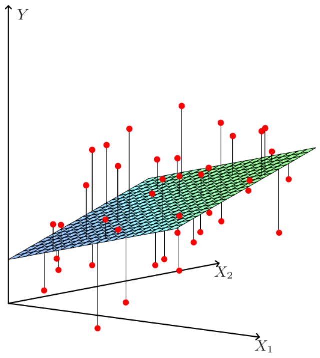

Figure 1: Linear model fitting with $\boldsymbol{x} \in \mathbb{R}^2$.

Based on *Ordinary Least Squares Method* mentioned in Appendix 6.1, we can obtain

$$\hat{\boldsymbol{w}}_{\text{OLS}} = \left(\boldsymbol{X}^\top \boldsymbol{X}\right)^{-1} \boldsymbol{X}^\top \boldsymbol{y}. \tag{2.2}$$

However, according to Eq (2.2), $\hat{\boldsymbol{w}}_{\text{OLS}}$ depends on $\boldsymbol{X}^\top \boldsymbol{X}$. In some cases, $\boldsymbol{X}^\top \boldsymbol{X}$ may be *singular* or *nearly singular*. When there are many correlated variables in a linear regression model, their coefficients can become poorly determined and exhibit high variance. In those cases, we call $\boldsymbol{X}$ *ill-conditioned*. Small changes to elements of $\boldsymbol{X}$ will lead to large changes in $\boldsymbol{X}^\top \boldsymbol{X}$. In addition, $\hat{\boldsymbol{w}}_{\text{OLS}}$ may provide a good fit to the training data, but it will not fit sufficiently well to the test data.

In order to alleviate this problem, we introduce *ridge regression*. Ridge regression shrinks the regression coefficients by imposing a penalty on their size. The ridge coefficients minimize a penalized residual sum of squares,

$$\hat{\boldsymbol{w}}_{\text{ridge}} = \arg\min_{\boldsymbol{w}} \left\{ \sum_{i=1}^{N} \left(y_i - \beta - \sum_{j=1}^{p} x_{ij} w_j \right)^2 + \lambda \sum_{j=1}^{p} w_j^2 \right\}. \tag{2.3}$$

Here $\lambda \geq 0$ is a *complexity parameter* that controls the amount of shrinkage.

Rewriting the criterion in Eq (2.3) in matrix form, we can obtain

$$\text{RSS} = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})^\top (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}) + \lambda \boldsymbol{w}^\top \boldsymbol{w}, \tag{2.4}$$

and

$$\begin{aligned}\frac{\partial \mathrm{RSS}}{\partial \boldsymbol{w}} &= -2\boldsymbol{X}^\top(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{w})+2\lambda\boldsymbol{w}\\ \frac{\partial^2 \mathrm{RSS}}{\partial \boldsymbol{w}\partial \boldsymbol{w}^\top} &= 2\boldsymbol{X}^\top\boldsymbol{X}+2\lambda\boldsymbol{I}.\end{aligned} \tag{2.5}$$

Let the first derivative in Eq (2.5) be zero, then the ridge regression solution can be expressed as

$$\hat{\boldsymbol{w}}_{\text{ridge}} = \left(\boldsymbol{X}^T\boldsymbol{X}+\lambda\boldsymbol{I}\right)^{-1}\boldsymbol{X}^\top\boldsymbol{y}, \tag{2.6}$$

where $\boldsymbol{I}$ is the $(p+1)\times(p+1)$ *identity matrix*. The solution adds a positive constant to the diagonal of $\boldsymbol{X}^\top\boldsymbol{X}$ before inversion, which makes this problem nonsigular, even if $\boldsymbol{X}^\top\boldsymbol{X}$ is not of full rank.

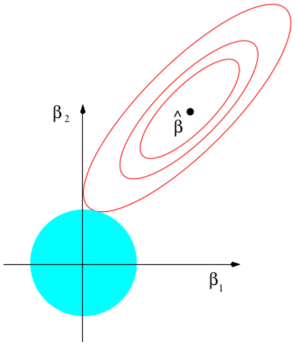

Figure 2: Estimation picture for ridge regression.

3 Optimization Algorithms

3.1 Gradient Method

Gradient Method is one of the most common optimization algorithm in many applications, for instance, Deep Learning and so on. Just as its name, the *antigradient* is choosen to be the search direction of the Gradient Method, since its the locally steepest descent of a differentiable function.

The general algorithm form of Gradient Method is as Algorithm. 1

Algorithm 1: Gradient Method

```
Input  : Objective function $f(\boldsymbol{x})$, gradient function $\boldsymbol{g}(\boldsymbol{x}) = \nabla f(\boldsymbol{x})$ and accuracy $\varepsilon$.
Output: Local minimum of $f(\boldsymbol{x})$: $\boldsymbol{x}^*$.
Initialization: Set $k = 0$ and initialize $\boldsymbol{x}_0 \in \mathbb{R}^n$.
while True do
    Calculate gradient $\boldsymbol{g}_k = \boldsymbol{g}(\boldsymbol{x}^{(k)})$.
    if $\|\boldsymbol{g}_k\| < \varepsilon$ then
        Stop iteration and Let $\boldsymbol{x}^* = \boldsymbol{x}_k$.
    else
        Let $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - h_k\boldsymbol{g}_k$
    end
    $k = k + 1$
end
```

3.2 Conjugate Gradient Method

Conjugate gradient methods was initially developed for minimizing quadratic functions. Consider the problem

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x})$$

with $f(\boldsymbol{x}) = \alpha + \langle \boldsymbol{a}, \boldsymbol{x}\rangle + \frac{1}{2}\langle \boldsymbol{A}\boldsymbol{x}, \boldsymbol{x}\rangle$ and $\boldsymbol{A} = \boldsymbol{A}^\top \succ 0$. We have known that the solution of this problem is $\boldsymbol{x}^* = -\boldsymbol{A}^{-1}\boldsymbol{a}$.

Consider the linear *Krylov* subspaces

$$\mathscr{L}_k = \mathrm{Lin}\{\boldsymbol{A}(\boldsymbol{x}_0 - \boldsymbol{x}^*), \dots, \boldsymbol{A}^k(\boldsymbol{x}_0 - \boldsymbol{x}^*)\}, \quad k \geq 1,$$

where $\boldsymbol{A}^k$ is the k-th power of matrix $\boldsymbol{A}$. A sequence of points $\{\boldsymbol{x}_k\}$ is generated in accordance with the following rule:

$$\boldsymbol{x}_k = \arg\min\{f(\boldsymbol{x}) \mid \boldsymbol{x} \in \boldsymbol{x}_0 + \mathscr{L}_k\}, \quad k \geq 1.$$

For any $k \geq 1$ we have $\mathscr{L}_k = \mathrm{Lin}\{\nabla f(\boldsymbol{x}_0), \dots, \nabla f(\boldsymbol{x}_{k-1})\}$. Here we define $\boldsymbol{\delta}_i = \boldsymbol{x}_{i+1} - \boldsymbol{x}_i$, and it is also clear that $\mathscr{L}_k = \mathrm{Lin}\{\boldsymbol{\delta}_0, \dots, \boldsymbol{\delta}_{k-1}\}$. From this we can derive the algorithm form of Conjugate Gradient Method as Algorithm. 2.

```
Algorithm 2: Conjugate Gradient Method
  Input   : Objective function $f(\boldsymbol{x})$ and accuracy $\varepsilon$.
  Output: Local minimum of $f(\boldsymbol{x})$: $\boldsymbol{x}^*$.
  Initialization: Set $k = 0$, and initialize $\boldsymbol{x}_0 \in \mathbb{R}^n$.
  Compute $f(\boldsymbol{x}_0)$ and $\nabla f(\boldsymbol{x}_0)$ and set $\boldsymbol{p}_0 = \nabla f(\boldsymbol{x}_0)$.
  while True do
      if $\|\nabla f(\boldsymbol{x}_k)\| < \varepsilon$ then
          Stop iteration and Let $\boldsymbol{x}^* = \boldsymbol{x}_k$.
      else
          Let $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - h_k \boldsymbol{p}_k$.
          Compute $f(\boldsymbol{x}_{k+1})$ and $\nabla f(\boldsymbol{x}_{k+1})$.
          Compute coefficient $\beta_k$.
          Define $\boldsymbol{p}_{k+1} = \nabla f(\boldsymbol{x}_{k+1}) - \beta_k \boldsymbol{p}_k$.
      end
      $k = k + 1$
  end
```

There exist many different formulas for the coefficient β_k. There are three most popular expressions.

1. Dai-Yuan:
$$\beta_k = \frac{\|\nabla f(\boldsymbol{x}_{k+1})\|^2}{\langle \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k), \boldsymbol{p}_k\rangle}$$

2. Fletcher–Rieves:
$$\beta_k = -\frac{\|\nabla f(\boldsymbol{x}_{k+1})\|^2}{\|\nabla f(\boldsymbol{x}_k)\|^2}$$

3. Polak–Ribbiere:
$$\beta_k = -\frac{\langle \nabla f(\boldsymbol{x}_{k+1}), \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k)\rangle}{\|\nabla f(\boldsymbol{x}_k)\|^2}$$

In a neighborhood of a strict minimum, the conjugate gradient schemes demonstrate a local

n-step quadratic convergence:

$$\|\boldsymbol{x}_{n+1} - \boldsymbol{x}^*\| \leq \text{const} \cdot \|\boldsymbol{x}_0 - \boldsymbol{x}^*\|^2$$

3.3 Quasi-Newton Method

Quasi-Newton Method, which is also called *variable metirc method*, is an alternative of Newton's Method. The Newton's Method requires Hessian matrix when finding the local minima. Nonetheless, it may be too expensive to calculate Hessian matrix each iteration. In contrast, quasi-Newton methods can achieve similar performance even when the Hessian matrix is unavailable.

For Newton's Method, it follows the following equation to finding the local minima:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - [\nabla^2 f(\boldsymbol{x})]^{-1} \nabla f(\boldsymbol{x}).$$

Since the Hessian matrix is unavailable in some cases, the quasi-Newton methods follow the following equation to finding the local minima:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - h_k \boldsymbol{H}_k \nabla f(\boldsymbol{x}),$$

in which $\boldsymbol{H}_k \to [\nabla^2 f(\boldsymbol{x}^*)]^{-1}$. If $\boldsymbol{H}_{k+1}$ satisfies

$$\boldsymbol{H}_{k+1} = \boldsymbol{H}_{k+1}^\top \succ 0 \quad \text{and} \quad \boldsymbol{H}_{k+1}(\nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k)) = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k,$$

we call it satisfies *quasi-Newton rule*. The general algorithm form of quasi-Newton Method is as Algorithm. 3.

```
Algorithm 3: Quasi-Newton Method
Input   : Objective function $f(\boldsymbol{x})$ and accuracy $\varepsilon$.
Output: Local minimum of $f(\boldsymbol{x})$: $\boldsymbol{x}^*$.
Initialization: Set $k = 0$, $\boldsymbol{H}_0 = \boldsymbol{I}_n$, and initialize $\boldsymbol{x}_0 \in \mathbb{R}^n$.
Compute $f(\boldsymbol{x}_0)$ and $\nabla f(\boldsymbol{x}_0)$.
while True do
    Calculate gradient $\boldsymbol{p}_k = \boldsymbol{H}_k \nabla f(\boldsymbol{x}_k)$.
    if $\|\nabla f(\boldsymbol{x}_k)\| < \varepsilon$ then
        Stop iteration and Let $\boldsymbol{x}^* = \boldsymbol{x}_k$.
    else
        Let $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - h_k \boldsymbol{p}_k$.
        Compute $f(\boldsymbol{x}_{k+1})$ and $\nabla f(\boldsymbol{x}_{k+1})$.
        Update the matrix $\boldsymbol{H}_k$ to $\boldsymbol{H}_{k+1}$.
    end
    $k = k + 1$
end
```

There are several ways to satisfy the quasi-Newton relation. Here we define

$$\Delta \boldsymbol{H}_k = \boldsymbol{H}_{k+1} - \boldsymbol{H}_k, \quad \boldsymbol{\gamma}_k = \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k), \quad \boldsymbol{\delta}_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k.$$

Then three most popular updating methods are as follows:

1. Rank-one correction scheme:

$$\Delta \boldsymbol{H}_k = \frac{(\boldsymbol{\delta}_k - \boldsymbol{H}_k \boldsymbol{\gamma}_k)(\boldsymbol{\delta}_k - \boldsymbol{H}_k \boldsymbol{\gamma}_k)^\top}{\langle \boldsymbol{\delta}_k - \boldsymbol{H}_k \boldsymbol{\gamma}_k, \boldsymbol{\gamma}_k \rangle}$$

2. Davidon–Fletcher–Powell scheme (DFP):

$$\Delta \boldsymbol{H}_k = \frac{\boldsymbol{\delta}_k \boldsymbol{\delta}_k^\top}{\langle \boldsymbol{\gamma}_k, \boldsymbol{\delta}_k \rangle} - \frac{\boldsymbol{H}_k \boldsymbol{\gamma}_k \boldsymbol{\gamma}_k^\top \boldsymbol{H}_k}{\langle \boldsymbol{H}_k \boldsymbol{\gamma}_k, \boldsymbol{\gamma}_k \rangle}$$

3. Broyden–Fletcher–Goldfarb–Shanno scheme (BFGS):

$$\Delta \boldsymbol{H}_k = \beta_k \frac{\boldsymbol{\delta}_k \boldsymbol{\delta}_k^\top}{\langle \boldsymbol{\gamma}_k, \boldsymbol{\delta}_k \rangle} - \frac{\boldsymbol{H}_k \boldsymbol{\gamma}_k \boldsymbol{\delta}_k^\top + \boldsymbol{\delta}_k \boldsymbol{\gamma}_k^\top \boldsymbol{H}_k}{\langle \boldsymbol{\gamma}_k, \boldsymbol{\delta}_k \rangle}$$

where $\beta_k = 1 + \langle \boldsymbol{H}_k \boldsymbol{\gamma}_k, \boldsymbol{\gamma}_k \rangle / \langle \boldsymbol{\gamma}_k, \boldsymbol{\delta}_k \rangle$.

In a neighborhood of strict minimum, quasi-Newton methods have a *superlinear* rate of convergence: for any $\boldsymbol{x}_0 \in \mathbb{R}^n$ there exists a number N such that for all $k \geq N$ we have

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}^*\| \leq \text{const} \cdot \|\boldsymbol{x}_k - \boldsymbol{x}^*\| \cdot \|\boldsymbol{x}_{k-n} - \boldsymbol{x}^*\|$$

Generally speaking, in a neighborhood of a strict minimum, the quasi-Newton methods demonstrate the highest rate of convergence. However, the Conjugate Gradient Methods have the advantage of cheap iteration. As far as the global convergence is concerned, these schemes are not better than the simplest Gradient Method.

3.4 Step-Length Selection

Gradient Method, Conjugate Gradient Method and Quasi-Newton Method are all *the first-order optimization methods*, which can be generally expressed as

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + h_k \boldsymbol{p}_k,$$

in which $\boldsymbol{p}_k$ is the *search direction* which is related to $\nabla f(\boldsymbol{x}_k)$ and h_k is the step size. For the step size, we usually hope it satisfies

$$h_k = \min_h f(\boldsymbol{x}_k + h_k \boldsymbol{p}_k).$$

When the objective function is quadratic, we can derive h_k analytically.

Theorem 1. If f is convex quadratic

$$f(\boldsymbol{x}) = \frac{1}{2} \boldsymbol{x}^\top \boldsymbol{Q} \boldsymbol{x} - \boldsymbol{b}^\top \boldsymbol{x},$$

Its one-dimensional minimizer along $\boldsymbol{x}_k + h_k \boldsymbol{p}_k$ is given by

$$h_k = -\frac{\nabla f(\boldsymbol{x}_k)^\top \boldsymbol{p}_k}{\boldsymbol{p}_k^\top \boldsymbol{Q} \boldsymbol{p}_k}$$

The proof of Theorem. 1 can be found in Appendix. 6.2.

4 Experiments

4.1 Experiment Settings

Generally 3 data sets (*abalone*, *bodyfat* and *housing*) are chosen to test the optimization methods. For all datasets, 80% data is used for training and remaining data is used for testing. In order to improve the performance, the scaled data is used.

First I build up the linear model using C++ which can be find in LinearModel.h and LinearModel.cpp. Before training, the weights of this linear model are initialized randomly.

Then three optimization methods are implemented in Optimizer.h and Optimizer.cpp. For all methods, the stop criteria are designed based on the *first-order optimality condition*: the iteration will last untill the gradient of objective function is small enough ($\nabla f(\boldsymbol{x}) \leq \varepsilon$). Here I set ε to 10^{-10}.

For the rigde regression expressed in Equation. 2.3, I choose $\lambda = 0.0,\ 0.1,\ 0.5,\ 1.0,\ 5.0,\ 10.0$ to test the optimization methods. For all optimization methods, step sizes are chosen according to Theorem. 1.

In the whole work, the Eigen library is used to perform matrix operations.

4.2 Experiment Results

The MSE losses on testing dataset of three optimization methods are shown in Fig. 3, Fig. 4 and Fig. 5. From three figures we can find that Conjugate Gradient Methods and Quasi-Newton Methods converge much more quickly than Gradient Method. Even though for all algorithms we choose the best step size during the iterations, Gradient Method still needs more than 10000 iterations to reach designed accuracy. From these experiments we can compare the rate of convergence of three methods directly, and this also satisfies analysis. The reason why Conjugate Gradient Methods and quasi-Newton methods demonstrate similar performance may be related to the best step size.

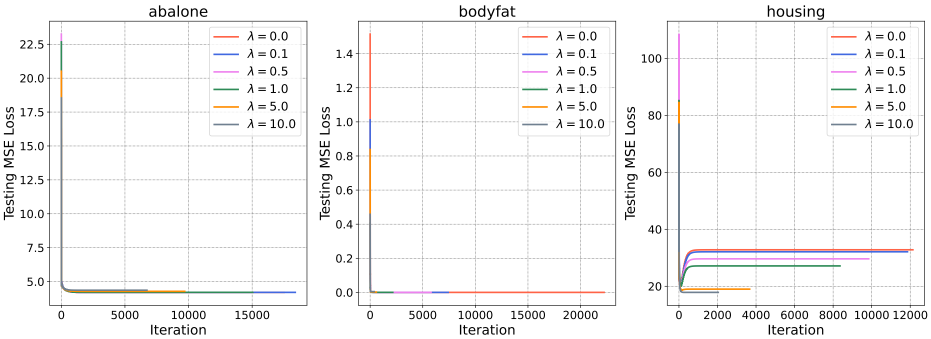

Figure 3: Testing MSE Loss of Gradient Methods on 3 datasets.

5 Conclusion

In this work, I introduce the rigde regression and implement Gradient Method, Conjugate Gradient Methods and quasi-Newton methods. Then I test them on three different datasets and show related results. I also have a better understanding through exactly coding. I really hope I can use them in my future work.

6 Appendix

6.1 Ordinary Least Squares Method

Generally speaking we have a training dataset with training data $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)$ from which we would like to estimate the parameter vector $\boldsymbol{w}$, and $\boldsymbol{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip})^\top$. If we use *Ordinary*

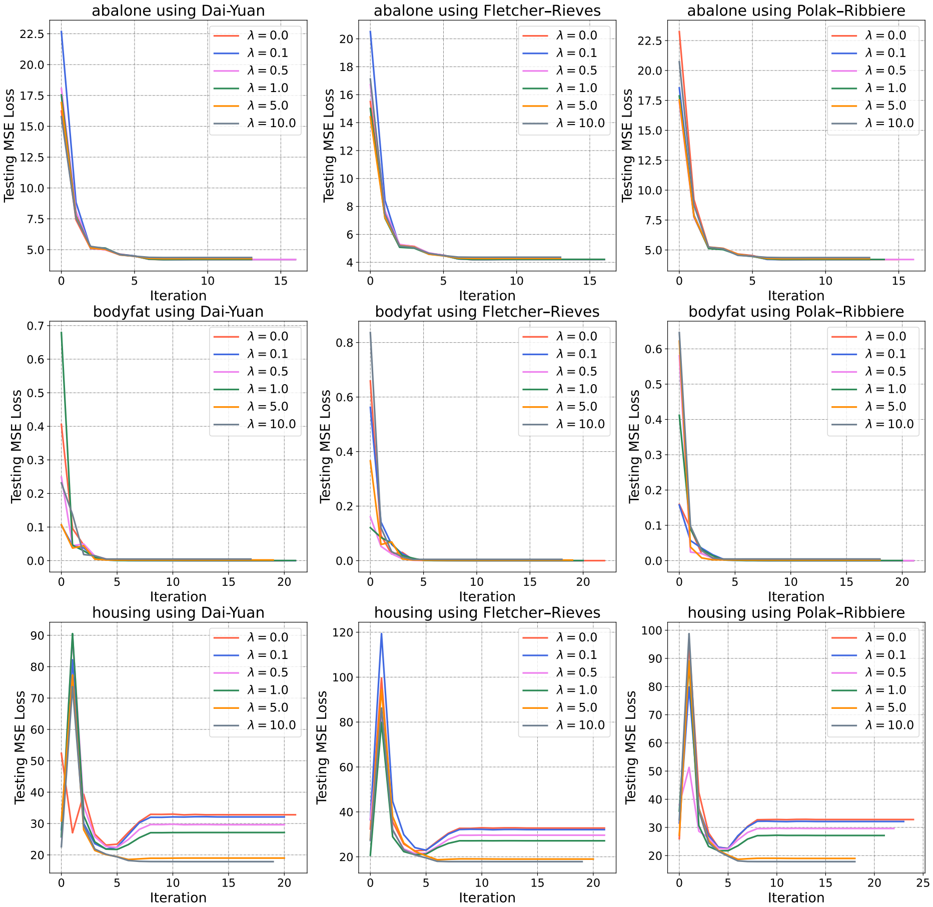

Figure 4: Testing MSE Loss of Conjugate Gradient Methods on 3 datasets.

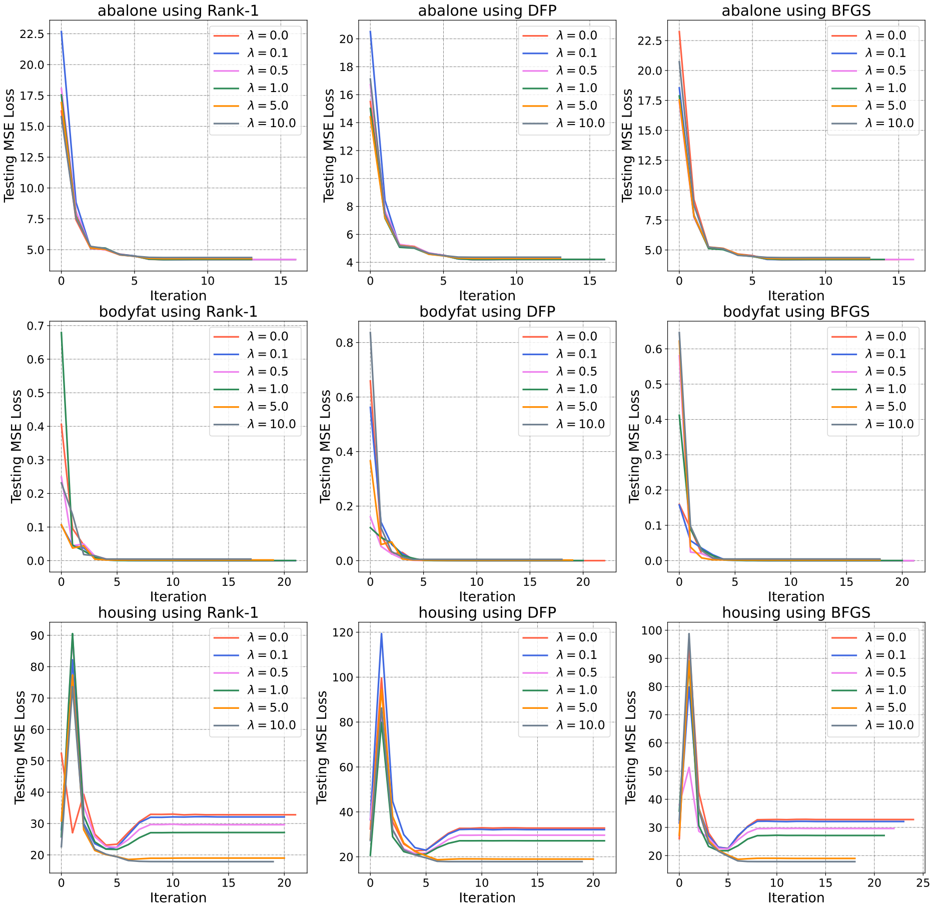

Figure 5: Testing MSE Loss of Quasi-Newton Methods on 3 datasets.

Least Squares (OLS) method, we will pick the $\boldsymbol{w}$ which can minimize *the residual sum of squares* (RSS)

$$\begin{aligned}\mathrm{RSS}(\boldsymbol{w}) &= \sum_{i=1}^{N} (y_i - f(\boldsymbol{x}_i))^2 \\ &= \sum_{i=1}^{N} \left(y_i - \beta - \sum_{j=1}^{p} x_{ij} w_j \right)^2 .\end{aligned} \tag{6.1}$$

Denote by $\boldsymbol{X} \in \mathbb{R}^{N\times(p+1)}$ the matrix with each row an input vector, and similarly, let $\boldsymbol{y} \in \mathbb{R}^N$ be the output vector in this training dataset. Then we can rewritte Eq (6.1) in as

$$\mathrm{RSS}(\boldsymbol{w}) = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})^\top (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}) \tag{6.2}$$

This is a quadratic function. Differentiating with respect to $\boldsymbol{w}$ we can get

$$\begin{aligned}\frac{\partial \mathrm{RSS}}{\partial \boldsymbol{w}} &= -2\boldsymbol{X}^\top (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}) \\ \frac{\partial^2 \mathrm{RSS}}{\partial \boldsymbol{w} \partial \boldsymbol{w}^\top} &= 2\boldsymbol{X}^\top \boldsymbol{X}.\end{aligned} \tag{6.3}$$

Suppose that $\boldsymbol{X}$ has full column rank. Therefore, $\boldsymbol{X}^\top \boldsymbol{X}$ is positive definite. We can set the first derivative to 0

$$\boldsymbol{X}^\top (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) = 0 \tag{6.4}$$

to get the unique solution

$$\hat{\boldsymbol{w}}_{\mathrm{OLS}} = \left(\boldsymbol{X}^\top \boldsymbol{X}\right)^{-1} \boldsymbol{X}^\top \boldsymbol{y}. \tag{6.5}$$

6.2 Proof of Theorems

Proof of Theorem. 1. For $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{Q}\boldsymbol{x} - \boldsymbol{b}^\top \boldsymbol{x}$,

$$f(\boldsymbol{x} + h\boldsymbol{p}_k) = \frac{1}{2}(\boldsymbol{x} + h\boldsymbol{p}_k)^\top \boldsymbol{Q}(\boldsymbol{x} + h\boldsymbol{p}_k) - \boldsymbol{b}^\top (\boldsymbol{x} + h\boldsymbol{p}_k)$$

with respect to h, and setting the derivative to zero, we obtain

$$h = -\frac{\nabla f(\boldsymbol{x}_k)^\top \boldsymbol{p}_k}{\boldsymbol{p}_k^\top \boldsymbol{Q}\boldsymbol{p}_k}.$$

Therefore we get the one-dimensional minimizer along $\boldsymbol{x}_k + h\boldsymbol{p}_k$. □