Proximal Gradient Descent For the LASSO Problem

Sam Brown, Henry Finnila, and Ryan Voss

June 9, 2025

1 Introduction

LASSO regression is a modeling technique used to remove less explanatory data features and prevent overfitting. Also known as L1 regularization, LASSO makes use of the L1 norm unlike Ridge regression, which uses the L2 norm. In the context of a p-variable problem with N observations, i.e. the data matrix, $X \in \mathbb{R}^{n \times p}$, LASSO regression takes the form.

$$\arg\min_{\beta} \left\{ \frac{1}{2} \|y - X\beta\|_2^2 \; + \; \lambda \|\beta\|_1 \right\}, \beta \in \mathbb{R}' \tag{1}$$

where $\|y - X\beta\|_2^2$ represents the model's loss, i.e., the squared L2 norm of the difference between the true response variable, y, and the model's estimate of the response, $X\beta$. This is also known as the model's residual sum of squares, or sum of squared error. What makes LASSO unique is the $\lambda\|\beta\|_1$ penalty term. λ is chosen through cross-validation, often through the following process:

1. Split the dataset into training and testing groups
2. Define a range of possible λ values to choose from
3. Fit a regression model to each
4. Compare each model to a chosen metric, e.g. Mean Squared Error or BIC

where,

$$\text{Mean Squared Error=MSE=} \sum_{i=1}^{n} (y_i - \hat{y_i})^2$$

where y is the true response variable and $\hat{y}$ is the response variable predicted by the model.

The BIC - Bayes Information Criterion - takes the form

$$\text{BIC=}\; k \ln(n) - 2\ln(\hat{L})$$

where k is the number of predictors in the model with n observations and maximized Likelihood function $\hat{L}$.

The choice of λ ultimately decides how much the model will penalize certain coefficients. A larger λ value shrinks coefficients to a greater magnitude, with the potential to shrink them all the way to zero - effectively removing their respective predictors from the model.

It is worth mentioning LASSO's counterpart, Ridge regression. Ridge regression has the form

$$\arg\min_{\beta} \left\{ \frac{1}{2} \|y - X\beta\|_2^2 \; + \; \frac{\lambda}{2} \|\beta\|_2^2 \right\}, \beta \in \mathbb{R}^{\prime} \tag{2}$$

The only difference between Ridge and LASSO regression is that Ridge applies the L2 norm to the penalty term while LASSO uses the L1 norm. This difference may seem subtle, but it is an essential distinction to draw between the two methods. It allows LASSO regression to shrink coefficients (corresponding to different predictors) all the way to zero, while Ridge can only shrink them asymptotically close. LASSO can therefore be used to remove predictors from the model that don't do a great job at explaining variability in the response variable. Let's take a look at why this is, mathematically.

1.1 Applications of LASSO

- Gene selection in high-dimensional microarray and RNA-seq studies [2].
- Feature selection for macroeconomic and financial time series forecasting [3].
- Sparse signal reconstruction in compressed sensing and image processing [4].
- Variable selection in marketing analytics for customer segmentation and churn prediction [5].
- Predictor identification in environmental and climate modeling [6].

1.2 Lasso vs. Ridge

Ridge solves

$$\arg\min_{\beta} \; \frac{1}{2}\|y - X\beta\|_2^2 + \; \frac{\lambda}{2} \|\beta\|_2^2.$$

The first-order condition is

$$-X^\top (y - X\beta) + \lambda \beta = 0 \quad \Longrightarrow \quad (X^\top X + \lambda I)\,\beta = X^\top y.$$

Since $X^\top X + \lambda I$ is always invertible for $\lambda > 0$, the unique solution $\beta = (X^\top X + \lambda I)^{-1} X^\top y$ has no mechanism to force any coordinate exactly to zero (unless $X^\top y$ lies in a coordinate plane).

Lasso solves

$$\arg\min_{\beta} \; \frac{1}{2}\|y - X\beta\|_2^2 + \; \lambda \|\beta\|_1.$$

At an optimum, $\hat{\beta}$, the subgradient condition is

$$0 \in -X^\top (y - X\hat{\beta}) + \; \lambda \, \partial \|\hat{\beta}\|_1,$$

where for each coordinate j

$$\partial |\beta_j| = \begin{cases} \{\text{sgn}(\beta_j)\}, & \beta_j \neq 0, \\ [-1, 1], & \beta_j = 0. \end{cases}$$

Thus for any j if

$$[X^\top (y - X\hat{\beta})]_j \in [-\lambda, \lambda],$$

we can satisfy the inclusion only by choosing $\hat{\beta}_j = 0$. This "flat" subgradient interval at zero lets Lasso *exactly* zero out weak-signal coefficients.

There is a problem with the LASSO Optimization problem, however. In the LASSO formula, decomposing

$$\arg\min_{\beta} \frac{1}{2}\|y - X\beta\|_2^2 + \lambda \|\beta\|_1.$$

into functions

$$g(x) = \frac{1}{2}\|y - X\beta\|_2^2$$

and

$$h(x) = \lambda \|\beta\|_1$$

$f(x)$ is differentiable while $h(x)$ is not differentiable. This poses a problem for our optimization. We cannot use standard gradient descent (or adjacent methods) since our function $h(x) = f(x) + g(x)$ contains a function $g(x)$ that's not necessarily differentiable. We need to come up with a unique algorithm to solve LASSO, apart from gradient descent.

2 Proposed Method

2.1 Derivation

To solve the LASSO problem, we will use an algorithm known as proximal or generalized gradient descent. Consider a function $f : \mathbb{R}^n \to \mathbb{R}$ which is not necessarily differentiable, but can be decomposed into the form

$$f(x) = g(x) + h(x),$$

where $g : \mathbb{R}^n \to \mathbb{R}$ is convex and differentiable, and $h : \mathbb{R}^n \to \mathbb{R}$ is convex but not necessarily differentiable. Note that the LASSO objective can easily be decomposed in this manner. Recall that the gradient descent update for a differentiable function g is given by

$$x_{k+1} = x_k - \alpha \nabla g(x_k),$$

for each $k \in \mathbb{N}$ with step size $\alpha \in \mathbb{R}$. One way to derive this formula is through the minimization of a quadratic approximation of g, using I_n rather than the second derivative of g. That is,

$$x_{k+1} = \arg\min_z \left(g(x_k) + \nabla g(x_k)^\top (z - x_k) + \frac{1}{2\alpha}\|z - x_k\|_2^2 \right).$$

To derive a similar update formula for our function f, we will apply this formula to g, without making any changes to h. Then,

$$x_{k+1} = \arg\min_z \left(g(x_k) + \nabla g(x_k)^\top (z - x_k) + \frac{1}{2\alpha}\|z - x_k\|_2^2 + h(z) \right). \tag{3}$$

This is the update function we will use for proximal gradient descent. In its current form, (3) is not very intuitive, so we will write it in a more concise manner.

Claim: The update function in (3) is equivalent to $x_{k+1} = \arg\min_z \left(\frac{1}{2\alpha}\|z - (x_k - \alpha\nabla g(x_k))\|_2^2 + h(z) \right)$.

Before we prove this claim, we must first prove a lemma from linear algebra that will be useful.

Lemma: For $x, y \in \mathbb{R}^n$, $\|x + y\|_2^2 = \|x\|_2^2 + 2x^\top y + \|y\|_2^2$.

Proof: Let $x, y \in \mathbb{R}^n$. Denote the i^{th} element of x and y as x_i and y_i respectively. Then,

$$\begin{aligned}
\|x+y\|_2^2 &= \sum_{i=1}^{n}(x_i+y_i)^2 \\
&= \sum_{i=1}^{n} x_i^2 + 2x_iy_i + y_i^2 \\
&= \sum_{i=1}^{n} x_i^2 + 2\sum_{i=1}^{n} x_iy_i + \sum_{i=1}^{n} y_i^2 \\
&= \|x\|_2^2 + 2x^\top y + \|y\|_2^2.
\end{aligned}$$

■

We are now ready to prove the claim.

Proof: Let $x, z \in \mathbb{R}^n,\ \alpha \in \mathbb{R}$, and $\nabla g : \mathbb{R}^n \to \mathbb{R}^n$. First note that $\|z-(x-\alpha\nabla g(x))\|_2^2 = \|(z-x)+\alpha\nabla g(x)\|_2^2$. By the previous lemma,

$$\begin{aligned}
\frac{1}{2\alpha}\|z-(x-\alpha\nabla g(x))\|_2^2 &= \frac{1}{2\alpha}\left(\|z-x\|_2^2 + 2\alpha\nabla g(x)^\top(z-x) + \|\alpha\nabla g(x)\|_2^2\right) \\
&= \frac{1}{2\alpha}\|z-x\|_2^2 + \nabla g(x)^\top(z-x) + \frac{1}{2\alpha}\|\alpha\nabla g(x)\|_2^2.
\end{aligned}$$

Therefore,

$$\arg\min_z\left(\frac{1}{2\alpha}\|z-(x-\alpha\nabla g(x))\|_2^2 + h(z)\right) = \arg\min_z\left(\frac{1}{2\alpha}\|z-x\|_2^2 + \nabla g(x)^\top(z-x) + \frac{1}{2\alpha}\|\alpha\nabla g(x)\|_2^2 + h(z)\right).$$

Because we are taking the arg min over z, any terms that do not depend on z will not affect the computation; they change what the minimum value is, but not which value of z achieves the minimum value. Thus, we can drop the $\frac{1}{2\alpha}\|\alpha\nabla g(x)\|_2^2$ term from the RHS to obtain

$$\arg\min_z\left(\frac{1}{2\alpha}\|z-(x-\alpha\nabla g(x))\|_2^2 + h(z)\right) = \arg\min_z\left(\frac{1}{2\alpha}\|z-x\|_2^2 + \nabla g(x)^\top(z-x) + h(z)\right).$$

Additionally, we can introduce terms that don't depend on z without affecting the equality, so

$$\arg\min_z\left(\frac{1}{2\alpha}\|z-(x-\alpha\nabla g(x))\|_2^2 + h(z)\right) = \arg\min_z\left(g(x) + \frac{1}{2\alpha}\|z-x\|_2^2 + \nabla g(x)^\top(z-x) + h(z)\right).$$

Therefore, the update in (3),

$$x_{k+1} = \arg\min_z\left(g(x_k) + \nabla g(x_k)^\top(z-x_k) + \frac{1}{2\alpha}\|z-x_k\|_2^2 + h(z)\right),$$

is equivalent to

$$x_{k+1} = \arg\min_z\left(\frac{1}{2\alpha}\|z-(x_k-\alpha\nabla g(x_k))\|_2^2 + h(z)\right). \tag{4}$$

■

This form allows us to gain some intuition on the motivation for the update function. The first term passed in to the arg min function indicates that we are searching for a $z \in \mathbb{R}^n$ that is close to the output of the update function for $g,\ x_{k+1} = x_k - \alpha\nabla g(x_k)$. The second term enforces that $h(z)$ must be small as well.

Define a function (called a proximal mapping) $\text{prox}_{\alpha,h} : \mathbb{R}^n \to \mathbb{R}^n$ by

$$\text{prox}_{\alpha,h}(x) = \arg\min_z \left(\frac{1}{2\alpha} \|z - x\|_2^2 + h(z) \right). \tag{5}$$

Observe that $\text{prox}_{\alpha,h}$ depends on both h and α, but not g. The update function (4) becomes

$$x_{k+1} = \text{prox}_{\alpha,h}(x_k - \alpha \nabla g(x_k)). \tag{6}$$

To write this in a more familiar form,

$$x_{k+1} = x_k - \alpha G(x_k),$$

where G is the generalized gradient

$$G(x_k) = \frac{x_k - \text{prox}_{\alpha,h}(x_k - \alpha \nabla g(x_k))}{\alpha}.$$

It is not immediately apparent that we have achieved anything useful. From (6), we can see that each iteration of proximal gradient descent requires one evaluation of ∇g and one evaluation of $\text{prox}_{\alpha,h}$. Our overall goal is to solve a single minimization problem, and our proposed method is an iterative algorithm that requires solving a minimization problem on every single step. It turns out, however, that we can find an analytic formula for $\text{prox}_{\alpha,h}$ for many important h functions, including the h induced by the LASSO objective. This allows for cheap evaluation of $\text{prox}_{\alpha,h}$, and our algorithm becomes useful. We will now examine the convergence properties of proximal gradient descent.

2.2 Convergence Properties

Theorem Consider a function $f : \mathbb{R}^n \to \mathbb{R}$. Suppose that f can be decomposed into the form $f(x) = g(x) + h(x)$ such that $g : \mathbb{R}^n \to \mathbb{R}$ is convex, differentiable, and L-smooth (∇g is Lipschitz with constant $L > 0$). Assume that h is convex, and $\text{prox}_{\alpha,h}(x) = \arg\min_z \left(\frac{1}{2\alpha} \|z - x\|_2^2 + h(z) \right)$. Then, proximal gradient descent with fixed step size $\alpha \in \left(0, \frac{1}{L}\right]$ satisfies

$$|f(x_k) - f(x^*)| \leq \frac{\|x_0 - x^*\|_2^2}{2\alpha k}.$$

The proof of this theorem has been omitted. It follows a very similar form as the proof of convergence of standard gradient descent, but using the generalized gradient. Notice that proximal gradient descent has convergence rate $\mathcal{O}(1/k)$, which is the same convergence rate as gradient descent. It is important to keep in mind that this analysis counts iterations, and depending on h, evaluation of $\text{prox}_{\alpha,h}$ may be expensive, so an iteration of proximal gradient descent may be significantly more expensive that an iteration of gradient descent. It turns out that for the LASSO objective, this is not the case, and we can evaluate $\text{prox}_{\alpha,h}$ easily.

2.3 Proximal Mapping for the LASSO Problem

The LASSO objective function in (1) can be naturally decomposed in to the form we need for proximal gradient descent. Let $g(\beta) = \frac{1}{2}\|y - X\beta\|_2^2$, and $h(\beta) = \lambda \|\beta\|_1$. Notice that both functions are convex, g is differentiable, ∇g is Lipschitz, but h is not differentiable. Adding f and g recovers the LASSO objective. Our goal is to find an analytic formula for $\text{prox}_{\alpha,h}(\beta)$, but we will begin by considering a simpler case. Suppose that $\beta \in \mathbb{R}$, and let $\tilde{h} : \mathbb{R} \to \mathbb{R}$ by $\tilde{h}(\beta) = \lambda|\beta|$. Then,

$$\text{prox}_{\alpha,\tilde{h}}(\beta) = \arg\min_{z \in \mathbb{R}} \left(\frac{1}{2\alpha}(z - \beta)^2 + \lambda|z| \right).$$

Case 1: Suppose that $\beta \geq 0$. Notice that there is no way for a negative z to be the minimizer; choosing $z = 0$ will always give a smaller output than $z < 0$. We can therefore restrict to nonnegative z, and our problem becomes

$$\operatorname{prox}_{\alpha,\tilde{h}}(\beta) = \underset{z\in[0,\infty)}{\arg\min} \left(\frac{1}{2\alpha}(z-\beta)^2 + \lambda z \right).$$

We now have a quadratic function of z. Since the coefficient on z^2 is positive, we have a parabola that opens upward. The minimizer of this parabola will be at the vertex, but we must be careful because we have restricted our search to nonnegative z. If the location of the vertex is nonnegative, we can take that value as our minimizer. If the location of the vertex is negative, our minimizer in the feasible set will be 0. To find the location of the vertex, we set the derivative to 0 and solve for z. This gives

$$\begin{aligned} \frac{1}{\alpha}(z-\beta) + \lambda &= 0 \\ z &= \beta - \alpha\lambda. \end{aligned}$$

Therefore, in the case where $\beta \geq 0$,

$$\operatorname{prox}_{\alpha,\tilde{h}}(\beta) = \begin{cases} \beta - \alpha\lambda & \text{if } \beta \geq \alpha\lambda \\ 0 & \text{else.} \end{cases}$$

Case 2: Now suppose that $\beta < 0$. Here, z must be nonpositive in order to be a minimizer. Thus,

$$\operatorname{prox}_{\alpha,\tilde{h}}(\beta) = \underset{z\in(-\infty,0]}{\arg\min} \left(\frac{1}{2\alpha}(z-\beta)^2 - \lambda z \right).$$

We are again minimizing a quadratic function of z. This time, we will take the location of the vertex as our minimizer if it is nonpositive, and if it is positive, we will take 0. To find the location of the vertex, we again set the derivative equal to 0, giving

$$\begin{aligned} \frac{1}{\alpha}(z-\beta) - \lambda &= 0 \\ z &= \beta + \alpha\lambda. \end{aligned}$$

Hence, in the case $\beta < 0$,

$$\operatorname{prox}_{\alpha,\tilde{h}}(\beta) = \begin{cases} \beta + \alpha\lambda & \text{if } \beta < -\alpha\lambda \\ 0 & \text{else.} \end{cases}$$

We can combine our two cases to get a single piecewise expression for $\operatorname{prox}_{\alpha,\tilde{h}}$. This gives

$$\operatorname{prox}_{\alpha,\tilde{h}}(\beta) = \begin{cases} \beta + \alpha\lambda & \text{if } \beta < -\alpha\lambda \\ 0 & \text{if } -\alpha\lambda < \beta < \alpha\lambda \\ \beta - \alpha\lambda & \text{if } \alpha\lambda < \beta. \end{cases}$$

We can now return to our original goal: finding a solution to the proximal operator applied to the LASSO objective function. Let $\beta \in \mathbb{R}^p$ and $h : \mathbb{R}^p \to \mathbb{R}$ by $h(\beta) = \lambda\|\beta\|_1$. Then,

$$\operatorname{prox}_{\alpha,h}(\beta) = \underset{z\in\mathbb{R}^p}{\arg\min} \left(\frac{1}{2\alpha}\|z-\beta\|_2^2 + \lambda\|z\|_1 \right).$$

Expanding the norms and manipulating the RHS, we get

$$\begin{aligned} \operatorname{prox}_{\alpha,h}(\beta) &= \underset{z}{\arg\min} \left(\frac{1}{2\alpha}\sum_{i=1}^{p}(z_i-\beta_i)^2 + \lambda\sum_{i=1}^{n}|z_i| \right) \\ &= \underset{z}{\arg\min} \left(\sum_{i=1}^{p}\left[\frac{1}{2\alpha}(z_i-\beta_i)^2 + \lambda|z_i| \right] \right). \end{aligned}$$

Note that each term the i^{th} term in the sum depends only on z_i and β_i. We can therefore choose each z_i to minimize its respective term. Each of these subproblems is exactly $z_i = \text{prox}_{\alpha,\tilde{h}}(\beta_i)$, which we now have a formula for. Therefore,

$$\text{prox}_{\alpha,h}(\beta) = S_{\alpha\lambda}(B), \tag{7}$$

where $S_{\alpha\lambda}$ is the soft thresholding operator defined element wise by

$$[S_{\alpha\lambda}(\beta)]_i = \begin{cases} \beta_i + \alpha\lambda & \text{if } \beta_i < -\alpha\lambda \\ 0 & \text{if } -\alpha\lambda < \beta_i < \alpha\lambda \\ \beta_i - \alpha\lambda & \text{if } \alpha\lambda < \beta_i, \end{cases}$$

for $i = 1, \cdots p$. The algorithm resulting from the application of proximal gradient descent to the LASSO problem is often called the Iterative Soft Thresholding Algorithm (ISTA).

3 Implementation

To demonstrate the implementation of proximal gradient descent, we will use the well-known LASSO (Least Absolute Shrinkage and Selection Operator) problem. The lasso problem requires the solving of the following optimization problem

$$\arg\min_{\beta} \left\{ \frac{1}{2} \|y - X\beta\|_2^2 \; + \; \lambda \, \|\beta\|_1 \right\}, \beta \in \mathbb{R}^{'}. \tag{8}$$

We will need to use proximal gradient descent as the term $\lambda \, \|\beta\|_1$ is not differentiable. We will use python and PyTorch to implement proximal gradient descent to solve this Lasso problem.

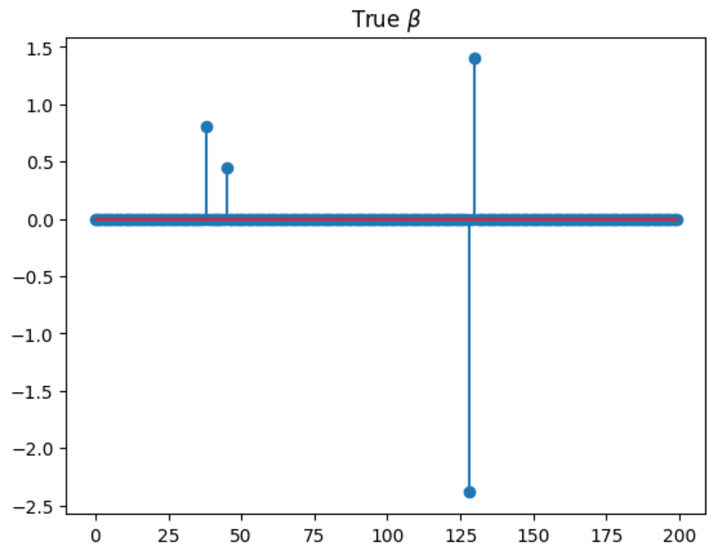
3.1 Data Initialization

For this implementation, we will use synthetic data for simplicity. The following code allows us to set up the synthetic data.

```python
import torch
import matplotlib.pyplot as plt

torch.manual_seed(1)
# synthetic data
N =  40 # training examples
d = 200 # number of features

nnz = 4 # non zero components

m = torch.randperm(d) # selecting components at random and assign them with values
betaTrue = torch.zeros(d)
betaTrue[m[:nnz]] = 5 * torch.randn(nnz)

X = torch.randn(N, d)
ones = torch.ones(X.size(0), 1)  # create a column of ones n rows, 1 col
X_new = torch.cat((ones, X), dim=1)  # concatenate along the column (dim=1)

noise = 0.1 * torch.randn(N)

y = X @ betaTrue + noise # target
```

Listing 1: Synthetic data creation

The Lasso problem is particularly effective at finding sparse solutions, making it well-suited for the synthetic data we generated, which was intentionally designed with sparsity in mind. The variable nnz controls how many truly "significant" predictors are present among all the features. To better reflect real-world scenarios, we also introduce a small amount of noise to each y value. This setup effectively simulates typical Lasso use cases, such as identifying a meaningful signal within noisy data or isolating the most important predictors from a high-dimensional feature set.
To visualize the structure of the true underlying model, we plot the coefficient vector β_{True}. As shown, only a small number of components are nonzero, reflecting the sparse nature of the data and motivating the use of Lasso for recovery.



3.2 Defining the Proximal Operator

To solve the Lasso problem, we must first implement the proximal operator for the L1 norm, which allows us to minimize the non-differentiable penalty term in the objective function. The proximal operator for the L1 norm is a transformation that shrinks each component of a vector toward zero by a fixed amount, effectively promoting sparsity. We implement this operation with the `soft_threshold` function, which moves each value closer to zero by a fixed amount `alpha`, setting small values exactly to zero. This is the key mechanism by which Lasso zeroes out unimportant coefficients. The `proxL1Norm` function applies this soft-thresholding step and optionally skips the first element—typically the intercept—so that it is not penalized, as is common practice in linear regression settings.

In section 2.3 we showed the prox operator to be equal to the soft thresholding operator, which is defined by

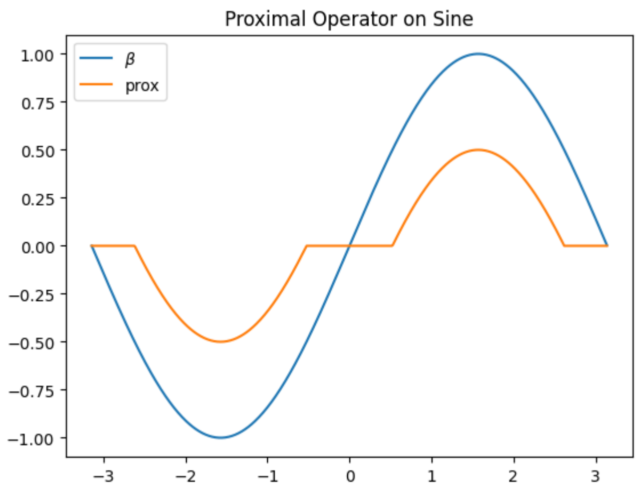
$$[S_{\alpha\lambda}(\beta)]_i = \begin{cases} \beta_i + \alpha\lambda & \text{if } \beta_i < -\alpha\lambda \\ 0 & \text{if } -\alpha\lambda < \beta_i < \alpha\lambda \\ \beta_i - \alpha\lambda & \text{if } \alpha\lambda < \beta_i, \end{cases}$$

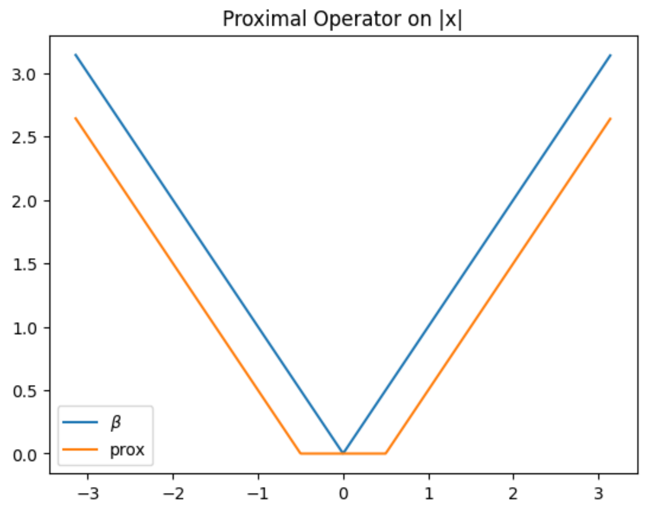
This function shifts each component of β toward zero by a fixed threshold $\alpha\lambda$, and sets it to zero when the magnitude is smaller than the threshold. This is the key operation behind the proximal operator for the L1 norm and is responsible for inducing sparsity in Lasso.
Our implementation of this proximal operator, shown below, uses PyTorch's `clamp` function to efficiently express this behavior. The absolute value of each element is reduced by `alpha`, and any values below zero are clipped to zero. The sign is then restored to recover the correct direction of shrinkage. The result exactly matches the piecewise definition of $S_{\alpha\lambda}$.

```
def soft_threshold(beta, alpha): # Beta is values you want to shrink, alpha is "step
    size" of shrinkage
    return torch.sign(beta) * torch.clamp(torch.abs(beta) - alpha, min=0.0) # moves
        all values closer to zero and once they get close enough to zero, make them
        zero (clamp)

def proxL1Norm(betaHat, alpha, penalizeALL=True): #betaHat is current estimate, alpha
    is step size again
    out = soft_threshold(betaHat, alpha)
    if not penalizeALL: # makes sure intercept is not penalized for linear regression
        cases
        out[0] = betaHat[0]
    return out
```

Listing 2: Proximal Operator

To test our implementation of the proximal operator, we can apply it to simple functions.





We observe that a single iteration of the proximal operator shifts the values of the function toward zero, and sets them exactly to zero if they are sufficiently close. These figures were generated using $\lambda = 0.5$.

3.3 Implementing Proximal Gradient Descent

Now that we have constructed the data and implemented the proximal operator, we are ready to implement a function to perform the descent of the proximal gradient.

```
def solveLasso_proxGrad(X, y, lmbda):
    maxIter = 300
    alpha = 0.005

    N, d = X.shape # N is number of data rows, d is number of features
    beta = torch.zeros(d, dtype=torch.float32) # intitialize coefficients

    costFunVals = torch.zeros(maxIter) # store cost at each iteration

    for t in range(maxIter):
        # gradient of the squared loss
        grad = X.T @ (X @ beta - y)

        #proximal gradient update for L1-penalized loss
        beta = proxL1Norm(beta - alpha * grad, alpha * lmbda)

        # compute Lasso objective
        residual = X @ beta - y
        cost = 0.5 * torch.norm(residual)**2 + lmbda * torch.sum(torch.abs(beta))
        costFunVals[t] = cost

        print(f"Iteration: {t}, Objective function value: {cost.item():.4f}")

    return beta, costFunVals
```

Listing 3: Proximal Gradient Descent

The core part of this function is the following line.

```
beta = proxL1Norm(beta - alpha * grad, alpha * lmbda)
```

This is the update step that is used for proximal gradient descent in this instance. It consists of two stages: a standard gradient descent step on the smooth squared loss, followed by the application of the proximal operator to handle the non-smooth L1 regularization. The update step can be formally written as:

$$\beta^{(k+1)} = \text{prox}_{\alpha,h}\left(\beta^{(k)} - \alpha_k \nabla f\left(\beta^{(k)}\right)\right).$$

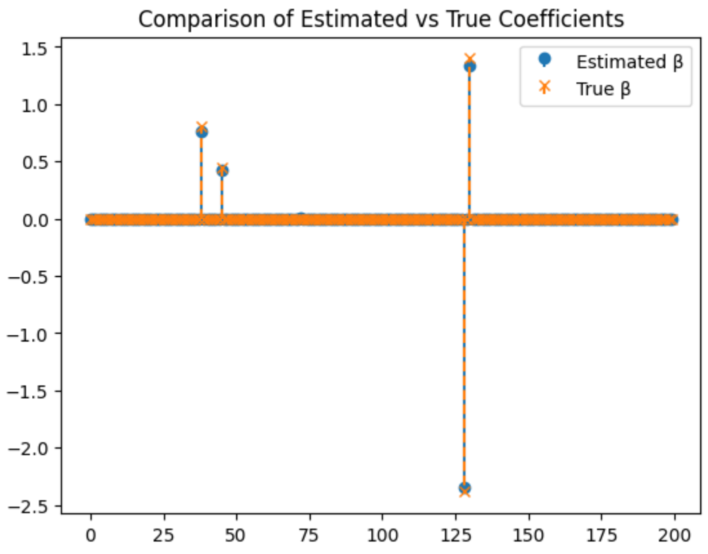
The second argument passed into the `proxL1Norm` function is the scaled regularization parameter $\alpha \cdot \lambda$. As shown in Equation (7), the soft-thresholding operator shifts each component of β toward zero by exactly this amount.

3.4 Testing Implementation

To test our proximal gradient descent code, we will use it to solve the synthetic data we introduced in section 3.1. We will call the function with our feature matrix, target matrix, and λ value.

```
lmbda = 2
beta, costFunVals = solveLasso_proxGrad(X, y, lmbda)
```

The function outputs the values for β and the cost function values. To test the correctness of our algorithm we can plot the true and predicted β values on a stem plot.



The results indicate that our implementation effectively denoised the signal and identified the key predictors, recovering the sparse structure of the true β vector.

3.5 Lasso Regression Using `scikit-learn`

While our earlier implementation demonstrates how Lasso regression can be solved from scratch using proximal gradient descent, practical applications typically rely on optimized libraries. In this section, we introduce a new dataset and show how to apply the `Lasso` class from the `scikit-learn` library to efficiently perform Lasso regression. This built-in implementation is highly optimized and offers convenient features such as cross-validation, standardized preprocessing, and robust performance for real-world data.

We use a dataset from `scikit-learn` called `california_housing`. This dataset is chosen for its convenience and compatibility with `scikit-learn`, allowing us to bypass many preprocessing steps and focus on the implementation of Lasso regression. While the dataset is not specifically tailored to highlight the strengths of Lasso (and other models may perform better), it serves well for illustrative purposes. To perform Lasso regression, we first import the necessary libraries:

```
from sklearn.datasets import load_diabetes, fetch_california_housing
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import Lasso
```

We then perform necessary data preprocessing steps.

```
hous = fetch_california_housing()
X = hous.data
y = hous.target

#split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state
    =42)

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Finally, we will initialize the Lasso model, fit it on the training sets, use it to predict the test set, and evaluate it using Mean Squared Error (MSE).

```
# Initialize Lasso model
lasso = Lasso(alpha=1.0)  # alpha is the regularization strength (have used lambda in
    other examples)

#fit model
lasso.fit(X_train_scaled, y_train)

# predict on the test set
y_pred = lasso.predict(X_test_scaled)

# Evaluate the model

mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

4 Discussion

Proximal gradient descent (ISTA) for LASSO combines a simple gradient step on the smooth squared-error term with a closed-form soft-thresholding update on the non-smooth L_1 term. This yields several key strengths and weaknesses:

Strengths

- **Sparsity promotion.** The soft-thresholding operator can set coefficients to zero - reducing model size
- **Simplicity and ease of implementation.** Each iteration requires only a matrix–vector multiplication and the element-wise proximal map, so it's not too difficult to implement/code
- **Guaranteed convergence.** Under standard assumptions (convexity, Lipschitz gradient), ISTA converges at a $\mathcal{O}(1/k)$ rate (equivalent to basic gradient descent).

Weaknesses

- **Slow convergence.** The $\mathcal{O}(1/k)$ rate can be slow in practice. Many iterations may be needed to reach accurate solutions.
- **Step-size sensitivity.** Choosing the step size α requires knowledge (or estimation) of the Lipschitz constant L of ∇g. Too large a step causes divergence; too small makes progress painfully slow. This makes tuning the parameters one of the most crucial, yet difficult parts of the process.

Overall, ISTA is quite a useful algorithm for sparse regression, but can take a while to reach optimal convergence and requires very finely tuned parameters to run well.

References

[1] R. Tibshirani, Convex Optimization: Proximal Gradient Methods, Lecture 8, Carnegie Mellon Univ., Pittsburgh, PA, USA, 2015. [Online]. Available: https://www.stat.cmu.edu/ ryantibs/convexopt-S15/lectures/08-prox-grad.pdf

[2] F. Hu, R. Zhao, and J. Xu, "Supervised group Lasso with applications to microarray data analysis," *BMC Bioinformatics*, vol. 9, p. 10, 2008.

[3] A. Bonavito and L. Morelli, "Forecasting macroeconomic time series: Lasso-based approaches," *Econometric Reviews*, vol. 33, no. 5–6, pp. 594–616, 2014.

[4] D. L. Donoho, "Compressed sensing," *IEEE Trans. Inf. Theory*, vol. 52, no. 4, pp. 1289–1306, Apr. 2006.

[5] L. Wang, H. Li, and M. Zhang, "Customer churn prediction based on Lasso and random forest models," in *Proc. 2018 IEEE Int. Conf. Cloud Comput. Big Data Anal.*, Chengdu, China, 2018, pp. 231–235.

[6] P. Liang, K. Fraedrich, and R. Schnur, "Predictor selection for downscaling GCM data with Lasso," *J. Geophys. Res. Atmos.*, vol. 117, no. D11, 2012.

Team Contributions

Each team member contributed approximately equally to the project. For the report, the work was divided as follows.

- Henry: Section 1 (Introduction) and Section 4 (Discussion)
- Ryan: Section 2 (Proposed Method)
- Sam: Section 3 (Implementation)

We each made and presented the slides that correspond to these sections.