Subgradient Methods Applied to LASSO Regression on Life Expectancy Data

Kasey Tian
Electrical and Computer Engineering
Rutgers University
New Brunswick, NJ, USA
kasey.tian@rutgers.edu

Abstract

Subgradient methods are a powerful class of methods to optimize non-differentiable objective functions. One common application is in LASSO regression, which is a regression scheme originally designed for use with linear classifiers. We have applied subgradient methods to LASSO regression to create a linear classifier of life expectancy data to prove the efficacy of this technique.

Index Terms

optimization, subgradients, LASSO, regression

I. Introduction

Subgradient methods are a way to perform an optimization on an objective function which is not fully differentiable. For non-differentiable objective functions, traditional methods, such as gradient descent and Newton's method, are impossible to execute. They can also be combined with a wide variety of other optimization methods, and have far reaching applications. They were originally developed by Shor and others in the Soviet Union in the 1960s and 70s [1] [2].

One common application of subgradient methods is in LASSO (**L**east **A**bsolute **S**hrinkage and **S**election **O**perator) regression. It is a regression analysis method which was first introduced in 1986 [3] in the field of geophysics, but was then independently rediscovered, named, and popularized in 1996 [4]. It is characterized by regularization using the L_1 norm, which involves the absolute value function and is therefore not fully differentiable. This makes it a prime candidate for applying subgradient methods.

We will explore the mathematics of subgradients, subgradient methods, and LASSO in Section II. To demonstrate real implementations of subgradient methods, two examples are included. The first is a simple linear example in Section III. The second is using LASSO regression with a linear classifier to predict life expectancy from World Health Organization (WHO) data [5] in Section IV. We will analyze the results of the LASSO regression in Section V, and then conclude with some brief remarks. The source code for these implementations as well as this paper is available in Appendix A.

II. Mathematics

In this section we will explore the mathematics of subgradients, subgradient methods and LASSO regression.

A. Subgradients

This section is mostly adapted from [6], with some supplementary material from [1].

1) Definitions: A subgradient is defined for some convex function $f : \mathbb{R}^n \to \mathbb{R}$ at a point $x \in \textbf{dom}\, f$ as a vector $g \in \mathbb{R}^n$ such that $\forall\, y \in \textbf{dom}\, f$

$$f(y) \geq f(x) + g^T(y - x) \tag{1}$$

Alternately expressed as

$$f(y) - f(x) \geq g^T(y - x) \tag{2}$$

$$f(x) - f(y) + g^T(y - x) \leq 0 \tag{3}$$

There can be multiple subgradients at a point x, so we will also define the subdifferential $\partial f(x)$ to be the set of all subgradients at x.

$$\partial f(x) = \bigcap_{y \in \textbf{dom}\, f} \left\{ g : f(y) \geq f(x) + g^T(y - x) \right\} \tag{4}$$

If there exists at least one subgradient at a point x, we would say f is subdifferentiable at x. If all points in the domain are subdifferentiable, we say that f is subdifferentiable.

2) Example: Absolute Value: If we consider g in (1) to be a slope, we can visualize a subgradient as being some hyperplane intersecting our function at x for which all values of the function are on or above the plane.

Let us employ this intuition to find a subgradient of the function $f(x) = |x|$ at the point $x = 0$. Graphically, we can see in Fig. 1 that many different lines satisfy this criteron. In fact, we can say that

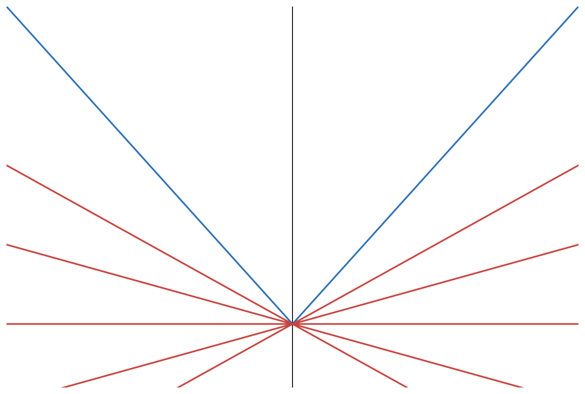


Fig. 1: Absolute value function (blue) with subgradients (red)

any $g \in [-1, 1]$ would be a subgradient, and therefore $\partial f(0) = [-1, 1]$. But what about other points? For a point $x > 0$, we can surmise that the only possible $g = 1$, as any other value will leave some parts of our function beneath the resulting plane. Likewise for $x < 0$, $g = -1$. Using (4), we can say

$$\partial f(x) = \begin{cases} \{-1\} & x < 0 \\ [-1, 1] & x = 0 \\ \{1\} & x > 0 \end{cases} \tag{5}$$

This can be compared against the derivative of $f(x)$.

$$f'(x) = \begin{cases} -1 & x < 0 \\ 1 & x > 0 \end{cases} \tag{6}$$

We find that where the function is differentiable, the subdifferential contains only the gradient. There is only a difference where the function is not differentiable. Here we find that our set ranges between the two derivatives on either side of it. More formally, we can say that $\partial f(x) = [a, b]$ such that

$$a = \lim_{y \to x^-} \frac{f(y) - f(x)}{y - x} \tag{7}$$

$$b = \lim_{y \to x^+} \frac{f(y) - f(x)}{y - x} \tag{8}$$

for one dimensional functions. We can note that for differentiable points $a = b$, so this definition satisfies our observations.

3) Properties: There are a few important properties that we shall take note of. Proofs of these properties can be found in Appendix B.

- If and only if the function is differentiable at x, $\partial f(x) = \{\nabla f(x)\}$
- If and only if x^* is a global minimum, $0 \in \partial f(x^*)$
- $\partial f(x)$ will always be a convex set

B. Subgradient Methods

This section is predominantly adapted from [1] and [2].

1) Iteration Step: Subgradient methods are a family of optimization techniques involving the same basic iteration step

$$x^{(k+1)} = x^{(k)} - \alpha_k g^{(k)} \tag{9}$$

where $g^{(k)}$ is any subgradient at the point $x^{(k)}$ and $\alpha_k > 0$. There are a number of different methods of choosing α_k, which will be further explored in Section II-C. This formula can be seen to be very similar to gradient descent. In fact, can be observed that since the only subgradient at differentiable points is the gradient, all subgradient methods *are* gradient descent in these sections, the only difference from regular gradient descent being our way of choosing step size. One notable difference from gradient descent is that subgradient methods are not descent methods; we are not guaranteed that every step will descend. For that reason we should have some method of tracking our best performing point.

$$f_{\text{best}}^{(k)} = \min(f_{best}^{(k-1)}, f(x^{(k)})) \tag{10}$$

And a corresponding $i_{\text{best}}^{(k)}$ such that $f(x^{(i_{\text{best}}^{(k)})}) = f_{\text{best}}^{(k)}$ This would not be needed in a descent method, because our best performing point is always the last point.

2) Convergence Proof: We are focused on converging towards the optimum point within some error of margin. We begin by making the following assumptions

- There exists a minimizer of f called x^*, for which $f^* = f(x^*)$
- There exists some bounding value G for which $\|g\|_2 \leq G$, $\forall\, g \in \partial f(x)$, $\forall\, x \in \mathbf{dom}\, f$[1][2]
- There exists some known bounding value R such that $R \geq \|x^{(1)} - x^*\|_2$

Next we can write the following based on (9)

$$\|x^{(k+1)} - x^*\|_2^2 = \|x^{(k)} - \alpha_k g^{(k)} - x^*\|_2^2 \tag{11}$$

[1]One way that this assumption can be met is if the function satisfies the Lipschitz condition, but it is not the only way

[2]This assumption helps us greatly in our analysis, but it is not strictly necessary. There exist subgradient methods which can be proven to work even when this assumption does not hold [1]

Focusing on the right side, we can multiply the quadratic out

$$\|x^{(k+1)} - x^*\|_2^2 = \|x^{(k)} - x^*\|_2^2 - 2\alpha_k g^{(k)T}(x^{(k)} - x^*) + \alpha_k^2 \|g^{(k)}\|_2^2 \tag{12}$$

Using the definition of subgradient from (2), we can write

$$\|x^{(k+1)} - x^*\|_2^2 \leq \|x^{(k)} - x^*\|_2^2 - 2\alpha_k (f(x^{(k)}) - f^*) + \alpha_k^2 \|g^{(k)}\|_2^2 \tag{13}$$

Paying particular attention to the term $\|x^{(k)} - x^*\|_2^2$, we can notice we have created a recursive inequality. If we perform a recursive substitution ultil reaching $x^{(1)}$ we would find ourselves with the following inequality

$$\|x^{(k+1)} - x^*\|_2^2 \leq \|x^{(1)} - x^*\|_2^2 - 2\sum_{i=1}^{k} \alpha_i (f(x^{(i)}) - f^*) + \sum_{i=1}^{k} \alpha_i^2 \|g^{(i)}\|_2^2 \tag{14}$$

We know that because of our bounding value $R \geq \|x^{(1)} - x^*\|_2$ and because of the norm, $\|x^{(k+1)} - x^*\|_2^2 \geq 0$, so we can write

$$2\sum_{i=1}^{k} \alpha_i (f(x^{(i)}) - f^*) \leq R^2 + \sum_{i=1}^{k} \alpha_i^2 \|g^{(i)}\|_2^2 \tag{15}$$

It follows readily from (10) that $f_{\text{best}}^{(k)} \leq f(x^{(i)}) \,\forall\, 1 \leq i \leq k$, so we can say

$$2\sum_{i=1}^{k} \alpha_i (f_{\text{best}}^{(k)} - f^*) \leq 2\sum_{i=1}^{k} \alpha_i (f(x^{(i)}) - f^*) \leq R^2 + \sum_{i=1}^{k} \alpha_i^2 \|g^{(i)}\|_2^2 \tag{16}$$

And therefore

$$f_{\text{best}}^{(k)} - f^* \leq \frac{R^2 + \sum_{i=1}^{k} \alpha_i^2 \|g^{(i)}\|_2^2}{2\sum_{i=1}^{k} \alpha_i} \tag{17}$$

We can finish by applying our G bounding condition to $\|g^{(k)}\|_2$, letting us write the final inequality

$$f_{\text{best}}^{(k)} - f^* \leq \frac{R^2 + G^2 \sum_{i=1}^{k} \alpha_i^2}{2\sum_{i=1}^{k} \alpha_i} \tag{18}$$

Given that R, G, and α_k are finite values, we can surmise that $f_{\text{best}}^{(k)}$ approaches f^* within some small error bound that is dependent on α_k and k. It is evident from this that the means we use to choose α_k will have significantly affect the error bound.

C. Choosing Step Size

There are many different methods of choosing the step size $\alpha^{(k)}$. We will be discussing five types in this section.

1) Constant Step Size: The first and simplest method we can use is a constant step size, where for some constant $\alpha > 0$

$$\alpha_k = \alpha \tag{19}$$

Given this, we can substitute (19) into (18) to obtain

$$f_{\text{best}}^{(k)} - f^* \leq \frac{R^2 + G^2 \alpha^2 k}{2\alpha k} \tag{20}$$

To determine the converging behavior we can take a limit

$$\lim_{k\to\infty} \frac{R^2 + G^2\alpha^2 k}{2\alpha k} = \frac{G^2\alpha}{2} \tag{21}$$

Therefore, we can surmise that we can reduce our eventual convergence bound by reducing α, although we should keep in mind that this comes at the expense of needing more steps. This is similar to the type of behavior we expect from other fixed step size methods when employing methods like gradient descent and Newton's method.

2) Constant Step Length: Second is a constant step length, where for some constant $\gamma > 0$

$$\alpha_k = \frac{\gamma}{\|g^{(k)}\|_2} \tag{22}$$

It can be easily shown that for this step size, $\|x^{(k+1)} - x^{(k)}\|_2 = \gamma$, hence the name constant step length. Because $\|g^{(k)}\|_2 \le G$, we can say that $\alpha_k \ge \gamma/G$, which enables us to use (17) to write

$$f_{\text{best}}^{(k)} - f^* \le \frac{R^2 + \gamma^2 k}{2\sum_{i=1}^k \alpha_i} \le \frac{R^2 + \gamma^2 k}{2\gamma k/G} \tag{23}$$

We can once more apply a limit to determine converging behavior

$$\lim_{k\to\infty} \frac{R^2 + \gamma^2 k}{2\gamma k/G} = \frac{G\gamma}{2} \tag{24}$$

3) Square Summable But Not Summable: This is a broad family of step size choosing methods, so long as the following are true

$$\alpha_k \ge 0 \quad \|\alpha\|_2^2 = \sum_{k=1}^{\infty} \alpha_k^2 < \infty \quad \sum_{k=1}^{\infty} \alpha_k = \infty \tag{25}$$

Which, as the name implies, means that the squares are summable to a finite value but the values themselves are not. One typical example of this is $\alpha_k = a/(b+k)$ for some $a > 0$ and $b \ge 0$. Combining (18) and (25), we can derive

$$f_{\text{best}}^{(k)} - f^* \le \frac{R^2 + G^2\|\alpha\|_2^2}{2\sum_{i=1}^k \alpha_i} \tag{26}$$

And once again using a limit to get the converging behavior

$$\lim_{k\to\infty} \frac{R^2 + G^2\|\alpha\|_2^2}{2\sum_{i=1}^k \alpha_i} = \frac{R^2 + G^2\|\alpha\|_2^2}{\infty} = 0 \tag{27}$$

Because $R^2 + G^2\|\alpha\|_2^2 < \infty$. Therefore $f_{\text{best}}^{(\infty)} \to f^*$, with no additional gap between them.

4) Nonsummable Diminishing Step Size: This is another broad family of step size choosing methods, where our criteria are as follows

$$\alpha_k \ge 0 \quad \lim_{k\to\infty} \alpha_k = 0 \quad \sum_{k=1}^{\infty} \alpha_k = \infty \tag{28}$$

As the name suggests, the step sizes eventually diminish to zero and the sum of all of the step sizes does not exist. One typical example is $\alpha_k = a/\sqrt{k}$ for some $a > 0$. Using (18) with (28) and taking limit, we find that the right hand size converges to zero because of the nonsummable denominator

$$\lim_{k\to\infty} \frac{R^2 + G^2\sum_{i=1}^k \alpha_i^2}{2\sum_{i=1}^k \alpha_i} = \frac{R^2 + G^2\sum_{i=1}^k \alpha_i^2}{\infty} = 0 \tag{29}$$

Therefore this type of method also guarantees the eventual convergence $f_{\text{best}}^{(\infty)} \to f^*$, with no gap between them.

5) Nonsummable Diminishing Step Length: Our final category of subgradient method is a step length method, similar to the constant step length method in Section II-C2 in that we take the form $\alpha_k = \gamma_k / \|g^{(k)}\|_2$. Note that there is no longer a singular, constant variable γ, and instead we have a variable γ_k. These γ_k must fulfill the following critera

$$\gamma_k \geq 0 \quad \lim_{k\to\infty} \gamma_k = 0 \quad \sum_{k=1}^{\infty} \gamma_k = \infty \tag{30}$$

D. LASSO Regression

This section is primarily adapted from [4]. As a reminder, LASSO is short for **L**east **A**bsolute **S**hrinkage and **S**election **O**perator. This section works through the mathematics of LASSO. See Section IV for the implementation.

1) Defining our problem: Let us define our problem to be p dimensional with a single scalar outcome. We can analyze n cases of this problem at a time, and define a covariate matrix $X \in \mathbb{R}^{n \times p}$ and results vector $y \in \mathbb{R}^n$ to be the collected inputs and outputs respectively. For a single case $i \in 1, \dots, n$ we can take the input as $X_i \in \mathbb{R}^p$ and the output as $y_i \in \mathbb{R}$. An individual output for case i can be written as $X_{i,j}$ for $j \in 1, \dots, p$. In [4] as well as in our example, we will be using a linear classification scheme, so we have some $\alpha \in \mathbb{R}$, $\beta \in \mathbb{R}^p$ where our classifier takes the form $y = \beta^T x + \alpha$.

2) Least Squares: We can write the LASSO estimate $(\hat{\alpha}, \hat{\beta})$ for our linear classifier as

$$\begin{aligned} (\hat{\alpha}, \hat{\beta}) = \operatorname{argmin} & \left\{ \sum_{i=1}^{n} \left(y_i - \alpha - \sum_{j=1}^{p} \beta_j X_{i,j} \right)^2 \right\} \\ \text{s.t.} & \sum_{j=1}^{p} |\beta_j| \leq t \end{aligned} \tag{31}$$

We can alternately express this in a more compact form

$$\begin{aligned} (\hat{\alpha}, \hat{\beta}) &= \operatorname{argmin} \|y - \alpha - X\beta\|_2^2 \\ \text{s.t.} &\|\beta\|_1 - t \leq 0 \end{aligned} \tag{32}$$

We can first add some additional definitions and assumptions. We assume that X is standardized and we have a vector $\bar{x}$ and a value $\bar{y}$ such that

$$\begin{aligned} \bar{x}_j = \sum_{i=1}^{n} X_{i,j}/n &= 0 \\ \sum_{i=1}^{n} X_{i,j}^2/n &= 1 \\ \bar{y} = \sum_{i=0}^{n} y_i/n &= 0 \end{aligned} \tag{33}$$

Next we can work on determining our optimum $\hat{\alpha}$. We can write an equation based on the portion of (32) inside of the norm and simplify it using substitutions from (33).

$$\begin{aligned}\hat{\alpha} &= \bar{y} - \bar{x}^T\beta \\ \hat{\alpha} &= \bar{y} \\ \hat{\alpha} &= 0\end{aligned} \tag{34}$$

We can see that with the assumption that our inputs and outputs are normalized, $\hat{\alpha}$ effectively disappears, making for much easier analysis. We can use this to write a Lagrangian form of (32), with a value of λ that corresponds to the impact of the L_1 norm penalty. This is directly related to the chosen value of t, but for convenience in the practical implementation we will work with the value of λ directly.

$$\hat{\beta} = \operatorname{argmin}\left\{\frac{1}{n}\|y - X\beta\|_2^2 + \lambda\|\beta\|_1\right\} \tag{35}$$

3) Subgradient of the Objective Function: In order to optimize (35) using a subgradient method we need to determine a subgradient.

$$\begin{aligned}g &= \frac{\partial}{\partial\beta}\left\{\frac{1}{n}\|y - X\beta\|_2^2 + \lambda\|\beta\|_1\right\} \\ &= \frac{2}{n}(y - X\beta)\sum_{i=1}^{n} X_i + \operatorname{sign}(\beta)\lambda\end{aligned} \tag{36}$$

Notice that because of the term $\|\beta\|_1$ the function is not differentiable, so it would be impossible to compute a gradient for it. The subgradient is therefore the only way we have to optimize this function.

4) Orthonormal Covariates and Characterization: In order to help us characterize the behavior of this regression we can find $\hat{\beta}$. we can additionaly assume that the covariates are orthonormal, which enables us to make the statement $X^TX = I$. We can define $\hat{\beta}^o$ that corresponds to the full least squares estimate, with a corresponding $t_o = \|\hat{\beta}^o\|_1$. For any values $t < t_o$ we will observe shrinkage corresponding to the difference between the values. We end up with a value of $\hat{\beta}^o = (X^TX)^{-1}X^Ty$. We can substitute this into (35) to get the following

$$\hat{\beta}_j = \hat{\beta}_j^o \max\left\{0, \left(1 - \frac{n\lambda}{|\hat{\beta}_j^o|}\right)\right\} \tag{37}$$

We find that this means that some β_j will be zero. This is the "selection" of the operation; only the non-zero coefficients will propogate x_j into the LASSO estimate. We can also observe that depending on the value of $\hat{\beta}_j^o$, it is possible for $\hat{\beta}_j$ to have the opposite sign (if it is selected). This interesting observations helps to set LASSO apart from other regression methods. We can also note that the function is definitely not fully differentiable, which means we must use subgradient methods to optimize it.

III. Simple Linear Example

A. Linear Function

To gain familiarity with subgradient methods, we can first implement work on a simple example. This example is a linear piecewise function, defined such that the objective function is

$$f(x) = \max(Ax + b) \tag{38}$$

Where $A \in \mathbb{R}^{m\times n}$, $b_i \in \mathbb{R}^m$, and $\max$ picks the maximum value of of a vector. This can be interpreted as m distinct linear equations with inputs of size n being computed and the maximum value being selected.

This function will have discontinuities where the linear functions intersect, as that is where the maximum will change. Also, being a pointwise maxiumum of a series of linear functions, this function will be convex because linear functions are individually convex. These traits together make this a suitable problem to practice subgradient methods on. In order to calculate a subgradient at a point x, we can find one of the constitutent functions that produces the maximum value and then its gradient will be a_i.

B. Source Code

Python was chosen to write a program to simulate this problem and solve it with an example of each of the 5 types of step size choice. It is broken into sections in this report and reformatted from its original source, but the original source code can be found along with all of the original files. See Appendix A.

1) Defining the problem: First we must define the dimensions of this problem $n = 10, m = 25$. Each of the linear equations was generated randomly using unit normal distrubutions, and by using enough seperate linear equations it was highly unlikely that the problem would end up unbounded below (Initially with lower values of m this was a problem, but it was found that higher values resolve this).

```python
#dimensions
n = 10
m = 25
#generate problem
a = np.random.normal(size=(m, n))
b = np.random.normal(size=m)
```

Next are the methods to determine the function value at a point and a subgradient at a point using the methods discussed in Section III-A

```python
def f(x):
    values = np.matvec(a, x) + b
    return np.max(values)
def g(x):
    values = np.matvec(a, x) + b
    return a[np.argmax(values)]
```

C. Step Sizes

To use each of the five different types of step sizes, they were all implemented in their own methods. Where there was a typical example listed for a category, that example was used. In the case of nonsummable diminishing step lengths, there was no typical example provided in [1], so a step length version of the nonsummable diminishing step size was used. It fulfills the criteria in (30), and so is a valid representative of this type of method.

```python
#step sizes
def constant_step_size(g, k):
    alpha = 0.01
    return alpha
def constant_step_len(g, k):
    gamma = 0.01
    norm = np.linalg.vector_norm(g)
    return gamma/norm
def square_sum_not_sum(g, k):
    a = 1
    b = 1
    return a/(b+k)
def nonsum_dim_size(g, k):
    a = 0.11
    return a / math.sqrt(k)
def nonsum_dim_len(g,k):
    a = 0.1
    gammak = a / math.sqrt(k)
    norm = np.linalg.vector_norm(g)
    return gammak/norm
```

D. Subgradient Method

This method uses whichever step size function is provided to execute a fixed number of steps, starting at the starting location.

```
def sgmethod(f, g, x0, step, maxiter):
    xkvec = [x0]
    xbest = x0
    for k in range(1,maxiter+1):
        xk = xkvec[-1]
        subgradient = g(xk)
        alphak = step(subgradient, k)
        xnext = xk - alphak*subgradient
        #keep track of best
        if f(xnext) < f(xbest):
            xbest = xnext
        xkvec.append(xnext)
    print(f"Finished. f* = {f(xbest):.5f}")
    return xkvec, xbest
```

E. Graphing

This final section of code calls the other methods, stores the appropriate data, and then generates the figures. Since the methods we are using do not guarantee finding the optimal f^* in a finite amount of time, the best value out of all of the values from all of the methods is taken as f^*.

```
x0 = np.zeros(n)
maxiter = 100
fstar = math.inf
def format(name, step):
    global fstar
    xkvec, xbest = sgmethod(f, g, x0, step, maxiter)
    thisbest = f(xbest)
    if thisbest < fstar:
        fstar = thisbest
    fkvec = [f(xk) for xk in xkvec]
    sgvec = [np.linalg.vector_norm(g(xk)) for xk in xkvec]
    return name, xkvec, fkvec, sgvec
```

```
formatted = []
formatted.append(format("Constant Size", constant_step_size))
formatted.append(format("Constant Length", constant_step_len))
formatted.append(format("Sq. Sum. not Sum.", square_sum_not_sum))
formatted.append(format("Nonsum. Dim. Size", nonsum_dim_size))
formatted.append(format("Nonsum. Dim. Length", nonsum_dim_len))

karr = [k for k in range(0,maxiter+1)]
fig, ax = plt.subplots()
ax.set_title("Value gap of $f(x^{(k)}) - f^*$ against $k$")
ax.set_ylabel("$f(x^{(k)}) - f^*$")
ax.set_xlabel("$k$")
ax.set_yscale('log')
for entry in formatted:
    gapvec = entry[2] - fstar
    ax.plot(karr, gapvec, label=entry[0])
ax.legend()

fig2, ax2 = plt.subplots()
ax2.set_title("$\|g^{(k)}\|_2$ against $k$ with " + entry[0])
ax2.set_ylabel("$\|g^{(k)}\|_2$")
ax2.set_xlabel("$k$")
ax2.set_yscale('log')
for entry in formatted:
    ax2.plot(karr, entry[3], label=entry[0])
ax2.legend()

plt.show()
```

F. Figures

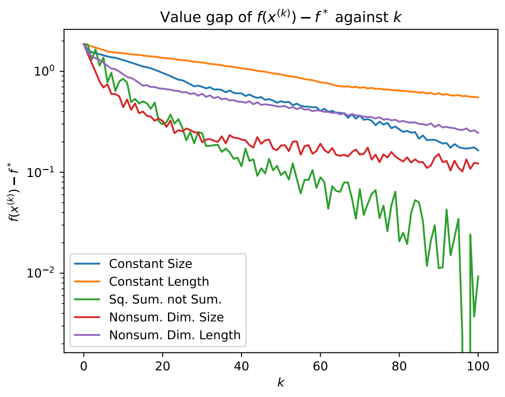
The two figures generated are Fig. 2a and 2b. We can make a few interesting observations that are congruent with what we have previously derived.

1) It is clear that none of these methods are descent methods, because they all increase at one point or another
2) Our last point is often not the most optimal point for a particular subgradient method

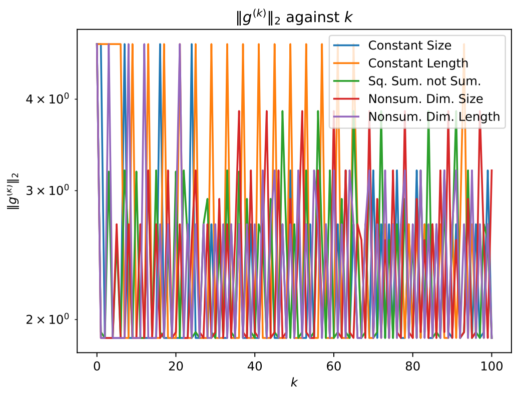
Additionally, we can note that at all points on all functions our subgradient normalized stuck to a few different magnitudes. It stands to reason that there would be exactly 25 such magnitues, each corresponding to one of our linear functions. In this case, it happened that the best performing method was using a square summable but not summable approach, however having run it a few times this is not always the case. This may be because of different function geometries behaving in different ways.

IV. LASSO Implementation

For the implementation of the life expectancy regression, the data was sourced from [5]. In this section we will delve into the code of the LASSO implementation, and in Section V we will review the results. The program was implemented in a Jupyter Python notebook.



(a) Value gap plot



(b) Norm subgradient plot

Fig. 2: Plots for the linear problem

A. *Importing Data and Libraries*

The data was formatted as a CSV file, so Pandas was used in order to import it. The non numerical categories of Country and Status were dropped, since they cannot be used in the linear classifier. There were also a fair number of entries that were empty due to the data being unavailable. As described by [5], this was mostly the case for very small countries. SciKit Learn was used to split the data into a training and testing dataset. While it also includes a LASSO implementation, the point of this entire exercise is to gain familiarity, so LASSO was implemented from scratch.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import math

data = pd.read_csv("Life Expectancy Data.csv")
data = data.drop(["Country", "Status"], axis=1)
data = data.dropna()

X = data.drop("Life expectancy ", axis = 1)
y = data["Life expectancy "]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25)
```

B. *LASSO Regression Object*

For convenience, the LASSO Regression and classifier were wrapped together into an object. This object makes working with different LASSO regression using different step sizes much easier. The `train` and `update_weights` methods are pretty simple, as well as the `predict` method implementing a very simple linear classification, as described in Section II-D1. The method for `subgradient` is a direct adaptation of (36), and `lasso_estimate` derived from (32).

```python
class LassoRegression():
    def __init__(self, name, p, beta, step_size, l1_penalty):
        self.name = name
        self.p = p
        self.beta = beta
        self.l1_penalty = l1_penalty
        self.step_size = step_size

    def train(self, X_train, y_train, maxiter):
        self.X_means = X_train.mean()
        self.X_std = X_train.std()
        self.y_mean = y_train.mean()
        self.y_std = y_train.std()

        estvec = []
        gvec = []
        bestbeta = self.beta
        minest = math.inf
        for k in range(1, maxiter+1):
            est, g = self.update_weights(X_train, y_train, k)
            estvec.append(est)
            gvec.append(np.linalg.norm(g))
            if estvec[-1] < minest:
                minest = estvec[-1]
                bestbeta = self.beta
            if k % 25 == 0:
                print(f"Finished {k} out of {maxiter} iterations on {self.name}")
        self.beta = bestbeta
        return estvec, gvec

    def update_weights(self, X, y, k):
        est = self.lasso_estimate(X, y)
        g = self.subgradient(X, y)
        self.beta -= self.step_size(g, k) * g
        return est

    def predict(self, X):
        norm_X = (X.sub(self.X_means, axis = 1)).div(self.X_std, axis = 1)
        norm_y = norm_X.dot(self.beta)
        return (norm_y * self.y_std) + self.y_mean

    def subgradient(self, X, y):
        norm_X = (X.sub(self.X_means, axis = 1)).div(self.X_std, axis = 1)
        norm_y = (y - self.y_mean) / self.y_std
        n = len(norm_y)
        sum = np.sum(norm_X, axis=0)
        g = np.multiply(np.linalg.norm(norm_y-norm_X.dot(self.beta)),sum)*2/n
        g += np.sign(self.beta)*self.l1_penalty
        return g

    def lasso_estimate(self, X, y):
        norm_X = (X.sub(self.X_means, axis = 1)).div(self.X_std, axis = 1)
        norm_y = (y - self.y_mean) / self.y_std
        pred = norm_X.dot(self.beta)
        return np.linalg.norm(norm_y-pred)**2
```

C. Step Sizes

This section was duplicated from Section III-C verbatim, with only some tuning as to the constants chosen.

```
def constant_step_size(g, k):
    alpha = 1
    return alpha

def constant_step_len(g, k):
    gamma = 1
    norm = np.linalg.vector_norm(g)
    return gamma/norm

def square_sum_not_sum(g, k):
    a = 10
    b = 1
    return a/(b+k)

def nonsum_dim_size(g, k):
    a = 2
    return a / math.sqrt(k)

def nonsum_dim_len(g,k):
    a = 2
    gammak = a / math.sqrt(k)
    norm = np.linalg.vector_norm(g)
    return gammak/norm
```

D. Training and Testing the Model

Here the code was very simple, only needing to instatiate the `LassoRegression` object and call the methods on it. While experimenting with different step size constants and λ values the print statement was included to monitor the performance of the classifier on the final data set using the LASSO estiamte.

```
p = len(X_train.columns)
beta = np.random.standard_normal(p)
models = [
    LassoRegression("Constant Size", p, beta.copy(), constant_step_size, 1e-1),
    LassoRegression("Constant Length", p, beta.copy(), constant_step_len, 1e-1),
    LassoRegression("Sq. Sum. not Sum.",p,beta.copy(),square_sum_not_sum,1e-1),
    LassoRegression("Nonsum. Dim. Size", p,beta.copy(),nonsum_dim_size,1e-1),
    LassoRegression("Nonsum. Dim. Length",p,beta.copy(),nonsum_dim_len,1e-1)]
maxiter = 50

train_results = [model.train(X_train, y_train, maxiter) for model in models]
test_lassos = [model.lasso_estimate(X_test, y_test) for model in models]
print(test_lassos)
```

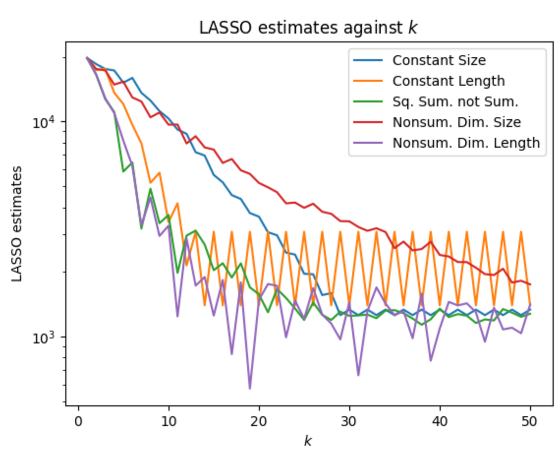
E. Graphing

The final section of code simply generates a graph displaying the LASSO estimate with each iteration, as well as one that displays the norm of the gradients on each iteration. The result is Fig. 3.

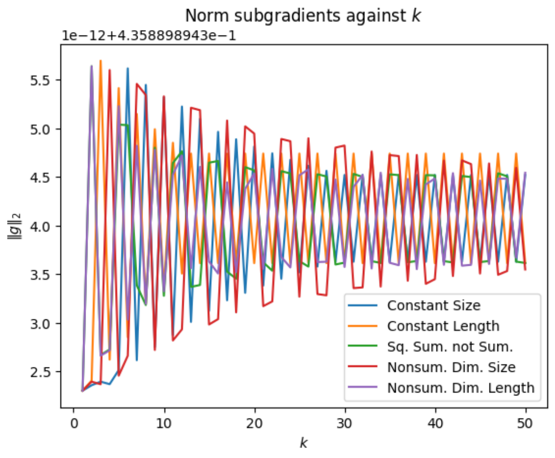
```
karr = list(range(1, maxiter + 1))

fig, ax = plt.subplots()
for i in range(len(models)):
    ax.plot(karr, train_results[i][0], label=models[i].name)
ax.set_title("LASSO estimates against $k$")
ax.set_ylabel("LASSO estimates")
ax.set_xlabel("$k$")
ax.set_yscale('log')
ax.legend()

fig2, ax2 = plt.subplots()
for i in range(len(models)):
    ax2.plot(karr, train_results[i][1], label=models[i].name)
ax2.set_title("Norm subgradients against $k$")
ax2.set_ylabel("$\\|g\\|_2$")
ax2.set_xlabel("$k$")
ax2.legend()

plt.show()
```

V. Numerical Results

We can examine our results a little more closely. We can note that just like our results in Section III-F, the most recent value is not always the best value, because we are using subgradient methods, which are not descent methods. We can also see oscillitory behavior at the end of the curves. This is presumably oscillating because one or more of the values of $\hat{\beta}_j$ are bouncing around zero on every iteration. Regardless, this is why we made sure during training to save the best $\hat{\beta}$, to ensure the best possible performance. We can also compare the convergence behaviors of our different methods. We see that nonsummable diminishing size converges the most slowly, followed by constant step size. The other methods converge at roughly the same speed, but nonsummable diminishing step length manages to find the best solution out of all the methods. Constant length seems to plateau with large oscillations at a worse solution than the other methods in this case, but this was not the case everytime that the program was run. Some of



(a) LASSO estimate during optimization



(b) LASSO norm subgradient plot

Fig. 3: Plots for the LASSO regression

these methods might be able to have better performance depending on the starting point randomly chosen and with finer tuning of the constants, but with limited time for this project these were the values we settled on.

VI. Conclusion

Subgradients and subgradient methods are a very versatile way of optimizing a wide variety of problems where traditional gradient-based methods cannot be used. A great example of an application is a LASSO regression scheme, which employs the non-differentiable L_1 norm. As demonstrated in this project, using these two together is an effective way to build a linear classifier for a given dataset. This project proved difficult at times, and certainly provided a challenge, but was ultimately very informative and educational.

Appendix

A. *Source Code Availablility*

The source code for this project, including the code for this LaTeX document, is available on GitHub: https://github.com/Derpanieux/Convex-Optimization-Term-Project.

B. *Proving Subgradient Properties*

1) When the Function is Differentiable: We can begin by supposing that f is differentiable at x, and therefore $\nabla f(x)$ exists. Let us write a definition for each of its elements

$$\nabla f_i(x) = \lim_{y \to x} \frac{f(y) - f(x)}{y_i - x_i} \tag{39}$$

And let us also rewrite (3) with an elementwise sum

$$f(x) - f(y) + \sum_{i=1}^{n} g_i^T (y_i - x_i) \leq 0 \tag{40}$$

We can combine (39) and (40)

$$\lim_{y \to x} f(x) - f(y) + \sum_{i=1}^{n} \nabla f_i(x)^T (y_i - x_i) \leq 0 \tag{41}$$

Substitute and simplify and we find that

$$\lim_{y \to x} f(x) - f(y) + \sum_{i=1}^{n} \nabla f_i(x)^T (y_i - x_i) = 0 \tag{42}$$

If we are interested in trying an alternate $g \neq \nabla f_i(x)$, we can say that a particular g_i would be invalid if

$$g_i(y_i - x_i) > \nabla f_i(x)(y_i - x_i) \tag{43}$$

If $g_i > \nabla f_i(x)$ and $y_i > x_i$, this condition is fulfilled and therefore no values of $g_i > \nabla f_i(x)$ are valid. Likewise, if $g_i < \nabla f_i(x)$ and $y_i < x_i$ this condition is fulfilled and therefore no values of $g_i < \nabla f_i(x)$ are valid. Therefore the only valid values for every $g_i = \nabla f_i(x)$, so the only valid $g = \nabla f(x)$ if $\nabla f(x)$ exists.

2) Global Minimum: If x^* is a global minimum it must be true that $\forall\, y \in \mathbf{dom}\, f$

$$f(x^*) \leq f(y) \tag{44}$$

If we have g equal to the zero vector in (1), we end up removing the second term, so we end up with

$$f(y) \geq f(x) \tag{45}$$

We can trivially redefine $x = x^*$ to derive (44). Therefore x^* is optimal if and only if the zero vecctor is included in its subdifferential.

3) Convexity: We can view (3) as defining a halfspace in terms of x, y, f. From (4) we can see that $\partial f(x)$ is composed of intersections of these halfspaces. Since halfspaces are always convex and the intersection of convex sets is always convex, $\partial f(x)$ must be a convex set.

References

[1] S. Boyd and J. Park, "Subgradient methods," May 2014. [Online]. Available: https://web.stanford.edu/class/ee364b/lectures/subgrad_method_notes.pdf

[2] S. Boyd, L. Xiao, and A. Mutapcic, "Subgradient methods," October 2003. [Online]. Available: https://web.mit.edu/6.976/www/notes/subgrad_method.pdf

[3] F. Santosa and W. W. Symes, "Linear inversion of band-limited reflection seismograms," *SIAM Journal on Scientific and Statistical Computing*, vol. 7, no. 4, pp. 1307–1330, 1986. [Online]. Available: https://doi.org/10.1137/0907087

[4] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 12 2018. [Online]. Available: https://doi.org/10.1111/j.2517-6161.1996.tb02080.x

[5] K. Jarashi, D. Russel, and D. Wang, "Life Expectancy (WHO)." [Online]. Available: https://www.kaggle.com/datasets/kumarajarshi/life-expectancy-who

[6] S. Boyd and L. Vandenberghe, "Subgradients," April 2008. [Online]. Available: https://see.stanford.edu/materials/lsocoee364b/01-subgradients_notes.pdf