University of California, Berkeley

IEOR 290 - Modern Optimization for Statistical Learning

Project Report

Cyril Tamraz

Vasileios Papanikolaou

Thursday, May 10, 2018

Contents

List of Tables

List of Figures

1 Paper Review - Summary

We start our project report by providing a summary of Haihao Lu's paper on "relative-continuity" [1]. For this, we start by recalling some notions on the traditional framework of mirror descent, and then we expand on this framework by relaxing some assumptions and indicating the yielded implications.

1.1 Traditional Framework

We recall the problem statement, assumptions, algorithm, and computational guarantees in the traditional framework, for both the deterministic setting and the stochastic setting.

1.1.1 Deterministic Setting

Problem Formulation. recall that our problem of interest was:

$$\begin{aligned} P : f^* := \min_{\mathbf{x}} & f(\mathbf{x}) \\ \text{s.t.} & \mathbf{x} \in Q \end{aligned} \tag{1.1}$$

Assumptions. the following assumptions are made:

1. Q is a closed convex set of the finite vector space F.
2. $f(.) : Q \to \mathbb{R}$ is a convex function (not necessarily differentiable).
3. $f(.)$ is uniformly Lipschitz on Q:

$$\|\mathbf{g}(\mathbf{x})\|_* \leq M_f, \forall g(\mathbf{x}) \in \partial f(\mathbf{x})$$

Proxy Function. this problem involves choosing a proxy function h such that:

1. $h(.) : F \to \mathbb{R}$ is a differentiable convex function.
2. $h(.)$ is μ_h-strongly convex.

Algorithm. the deterministic mirror descent algorithm looks as follows:

Algorithm 1 Deterministic Mirror Descent

```
Initialize with $x^0 \in Q$
Choose the appropriate proxy $h(.)$
Perform Updates at iteration $i$:
    Compute a subgradient $\mathbf{g}(\mathbf{x}^i) \in \partial f(\mathbf{x}^i)$
    Determine step-size $t_i$
    Update $\mathbf{x}^{i+1} \leftarrow \arg\min_{\mathbf{x}\in Q}\{f(\mathbf{x}^i) + \langle \mathbf{g}(\mathbf{x}^i), \mathbf{x} - \mathbf{x}^i\rangle + \frac{1}{t_i} D_h(\mathbf{x}, \mathbf{x}^i)$
```

Computational Guarantees. after k iterations, the following bound holds:

$$\min_{0\leq i\leq k} f(\mathbf{x}^i) - f(\mathbf{x}) \leq \frac{\frac{1}{2} M_f^2 \sum_{i=0}^k t_i^2 + D_h(\mathbf{x}, \mathbf{x}^0)}{\sum_{i=0}^k t_i} \tag{1.2}$$

which, with the appropriate step-size t_i, is $O\left(\frac{1}{\sqrt{k}}\right)$.

1.1.2 Stochastic Setting

Problem Formulation. recall that our problem of interest was:

$$(\hat{P}_n) : \hat{F}_n^* := \min_{\mathbf{x}} \hat{F}_n(\mathbf{x}) := \frac{1}{n}\sum_{i=1}^{n} f(\mathbf{x}, z_i)$$
$$\text{s.t.} \mathbf{x} \in Q \tag{1.3}$$

Assumptions. the following assumptions are made:

1. Q is a closed convex set of the finite vector space F.
2. $f(.) : Q \to \mathbb{R}$ is a convex function (not necessarily differentiable).
3. Unbiasedness of stochastic subgradients:
$$\mathrm{E}[\bar{\mathbf{g}}(\mathbf{x}^i)|\mathbf{x}^i] \in \partial f(\mathbf{x}^i)$$
4. $f(.)$ is stochastically continuous on Q:
$$\mathrm{E}[\|\mathbf{g}(\mathbf{x})\|_*^2|\mathbf{x}] \leq G_f^2, \forall \mathbf{x} \in Q$$

Proxy Function. this problem involves choosing a proxy function h such that:

1. $h(.) : F \to \mathbb{R}$ is a differentiable convex function.
2. $h(.)$ is 1-strongly convex.

Algorithm. the stochastic mirror descent algorithm looks as follows:

Algorithm 2 Stochastic Mirror Descent

```
**Initialize** with $x^0 \in Q$
**Choose** the appropriate proxy $h(.)$
**Perform Updates** at iteration $i$:
  **Compute** a stochastic subgradient $\bar{\mathbf{g}}(\mathbf{x}^i) \in \partial f(\mathbf{x}^i)$
  **Determine** step-size $t_i$
  **Update** $\mathbf{x}^{i+1} \leftarrow \arg\min_{\mathbf{x}\in Q}\{f(\mathbf{x}^i) + \langle \bar{\mathbf{g}}(\mathbf{x}^i), \mathbf{x} - \mathbf{x}^i\rangle + \frac{1}{t_i} D_h(\mathbf{x}, \mathbf{x}^i)$
```

Computational Guarantees. after k iterations, the following bound holds:

$$\mathrm{E}[f(\bar{\mathbf{x}}^k)] - f^* \leq \frac{3G_f\sqrt{D_{\max}}}{2\sqrt{\mu_h(k+1)}} \tag{1.4}$$

which, with the appropriate step-size t_i, is $O\left(\frac{1}{\sqrt{k}}\right)$. The corresponding sequence of iterates is given by:

$$\bar{\mathbf{x}}^k := \frac{1}{\sum_{i=0}^{k} t_i}\sum_{i=0}^{k} t_i \mathbf{x}^i, \text{ and } D_{\max} := \max_{\mathbf{x},\mathbf{y}\in Q} D_h(\mathbf{x}, \mathbf{y}) \tag{1.5}$$

Moreover, if $f(.)$ is in addition μ_f-strongly convex and $h(.)$ is L_h-smooth, the bound becomes:

$$\mathrm{E}[f(\bar{\mathbf{x}}^k)] - f^* \leq \frac{2G_f^2 L_h}{\mu_f(k+1)\mu_h} \tag{1.6}$$

which, with the appropriate step-size t_i, is $O\left(\frac{1}{k}\right)$. The corresponding sequence of iterates is given by:

$$\bar{\mathbf{x}}^k := \frac{2}{(i+1)(i+2)}\sum_{i=0}^{k}(i+1)\mathbf{x}^i \tag{1.7}$$

1.2 New Framework

The motivation for defining this new framework arises from the fact that many common problems involve an objective function that is neither smooth nor Lipschitz continuous. For instance, consider the below two problems:

$$\textbf{SVM} : f^* = \min_{\mathbf{x}} f(\mathbf{x}) := \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i \mathbf{x}^\top w_i\} + \frac{\lambda}{2}\|x\|_2^2 \tag{1.8}$$

$$\textbf{LASSO} : f^* = \min_{\mathbf{x}} f(\mathbf{x}) := \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{W}\mathbf{x} - \mathbf{y}\|_2^2 + \lambda\|x\|_1 \tag{1.9}$$

We observe that **SVM** is not smooth due to the max term, and not Lipschitz due to the ℓ_2-norm term. Similarly, **LASSO** is not smooth due to the ℓ_1-norm term, and not Lipschitz due to the ℓ_2-norm term. This shows the need to define a new framework to take care of these.

1.2.1 Deterministic Setting

Definition 1.1. *(Relative Continuity)*
$f(.)$ is M-relative continuous with respect to the reference function $h(.)$ on Q if, for any $\mathbf{x}, \mathbf{y} \in Q$ with $\mathbf{x} \neq y$, and $\mathbf{g}(\mathbf{x}) \in \partial f(\mathbf{x})$, it holds that:

$$\|\mathbf{g}(\mathbf{x})\|_* \leq \frac{M\sqrt{2D_h(\mathbf{y}, \mathbf{x})}}{\|\mathbf{y} - \mathbf{x}\|} \tag{1.10}$$

which is somehow equivalent to bounding it with the ration of the Bregman distance and the square of the primal norm:

$$\|\mathbf{g}(\mathbf{x})\|_* \leq M^2 \frac{D_h(\mathbf{y}, \mathbf{x})}{\frac{1}{2}\|\mathbf{y} - \mathbf{x}\|^2} \tag{1.11}$$

Definition 1.2. *(Relative Strong Convexity)*
$f(.)$ is μ-strongly convex relative to $h(.)$ on Q if there is a scalar $\mu \geq 0$ such that, for any $\mathbf{x}, \mathbf{y} \in$ int Q and any $\mathbf{g}(\mathbf{x}) \in \partial f(.)$, it holds that:

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \langle \mathbf{g}(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + \mu D_h(\mathbf{y}, \mathbf{x}) \tag{1.12}$$

Assumptions. with the same problem of interest 1.1, the following assumptions are made:

1. Q is a closed convex set of the finite vector space F. **(same as before)**
2. $f(.) : Q \to \mathbb{R}$ is a convex function (not necessarily differentiable). **(same as before)**
3. $f(.)$ is M-relative continuous w.r.t. $h(.)$ on Q. **(modified)**

This shows that we should choose an appropriate proxy function $h(.)$ in accordance with the above assumption 3.

Computational Guarantees. with the algorithm being the same as 1 (with the only difference that we now choose "wisely" the proxy function $h(.)$), after k iterations, the following bound holds:

$$f(\bar{\mathbf{x}}^k) - f(\mathbf{x}) \leq \frac{\frac{1}{2}M^2 \sum_{i=0}^{k} t_i^2 + D_h(\mathbf{x}, \mathbf{x}^0)}{\sum_{i=0}^{k} t_i} \tag{1.13}$$

which, with the appropriate step-size t_i, is $O\left(\frac{1}{\sqrt{k}}\right)$. The corresponding sequence of iterates is given by:

$$\bar{\mathbf{x}}^k := \frac{1}{\sum_{i=0}^k t_i} \sum_{i=0}^k t_i \mathbf{x}^i \tag{1.14}$$

Moreover, if $f(.)$ is in addition μ_f-strongly convex w.r.t. $h(.)$, the bound becomes:

$$f(\bar{\mathbf{x}}^k)] - f^* \leq \frac{2M^2}{\mu(k+1)} \tag{1.15}$$

which, with the appropriate step-size t_i, is $O\left(\frac{1}{k}\right)$. The corresponding sequence of iterates is given by:

$$\bar{\mathbf{x}}^k := \frac{2}{k(k+1)} \sum_{i=0}^k i\mathbf{x}^i \tag{1.16}$$

1.2.2 Stochastic Setting

Definition 1.3. *(Relative Stochastic Continuity)*
$f(.)$ is G-stochastically-relative continuous with respect to the reference function $h(.)$ on Q if f together with the oracle to compute a stochastic subgradient satisfies:

1. *Unbiasedness property:*
$$E[\bar{g}(\mathbf{x})|\mathbf{x}] \in \partial f(\mathbf{x}) \tag{1.17}$$

2. *Boundedness property: $\forall \mathbf{x}, \mathbf{y} \in Q$ with $\mathbf{x} \neq y$, it holds that:*
$$E[\|\bar{\mathbf{g}}(\mathbf{x})\|_*^2|\mathbf{x}] \leq G^2 \frac{D_h(\mathbf{y}, \mathbf{x})}{\frac{1}{2}\|\mathbf{y} - \mathbf{x}\|^2} \tag{1.18}$$

Assumptions. with the same problem of interest 1.1, the following assumptions are made:

1. Q is a closed convex set of the finite vector space F. **(same as before)**
2. $f(.) : Q \to \mathbb{R}$ is a convex function (not necessarily differentiable). **(same as before)**
3. $f(.)$ is G-relative-stochastically continuous w.r.t. $h(.)$ on Q. **(modified)**

This shows that we should choose an appropriate proxy function $h(.)$ in accordance with the above assumption 3.

Computational Guarantees. with the algorithm being the same as 2 (with the only difference that we now choose "wisely" the proxy function $h(.)$), after k iterations, the following bound holds:

$$\mathrm{E}_{\xi_{k-1}}[f(\bar{\mathbf{x}}^k)] - f^* \leq \frac{\frac{1}{2}G^2 \sum_{i=0}^k t_i^2 + D_h(\mathbf{x}, \mathbf{x}^0)}{\sum_{i=0}^k t_i} \tag{1.19}$$

which, with the appropriate step-size t_i, is $O\left(\frac{1}{\sqrt{k}}\right)$. The corresponding sequence of iterates is given by:

$$\bar{\mathbf{x}}^k := \frac{1}{\sum_{i=0}^k t_i} \sum_{i=0}^k t_i \mathbf{x}^i \tag{1.20}$$

Moreover, if $f(.)$ is in addition μ_f-strongly convex w.r.t. $h(.)$, the bound becomes:

$$\mathrm{E}_{\xi_{k-1}}[f(\bar{\mathbf{x}}^k)] - f^* \leq \frac{2G^2}{\mu(k+1)} \tag{1.21}$$

which, with the appropriate step-size t_i, is $O\left(\frac{1}{k}\right)$. The corresponding sequence of iterates is given by:

$$\bar{\mathbf{x}}^k := \frac{2}{k(k+1)} \sum_{i=0}^{k} i\mathbf{x}^i \tag{1.22}$$

1.3 Takeaways from the Paper

We start by summarizing the bounds in table 1.1:

	No Added Assumption	$f(.)$ **Strongly Convex**
$f(.)$ **Lipschitz**	$O\left(\frac{1}{\sqrt{k}}\right)$	$O\left(\frac{1}{k}\right)$
$f(.)$ **Relatively Continuous w.r.t.** $h(.)$	$O\left(\frac{1}{\sqrt{k}}\right)$	$O\left(\frac{1}{k}\right)$
$f(.)$ **Stochastically Continuous**	$O\left(\frac{1}{\sqrt{k}}\right)$	$O\left(\frac{1}{k}\right)$
$f(.)$ **Relatively Stochastically Continuous w.r.t.** $h(.)$	$O\left(\frac{1}{\sqrt{k}}\right)$	$O\left(\frac{1}{k}\right)$

Table 1.1: Traditional/ New Framework Bounds

Which shows that, with the new framework i.e. relaxed assumptions, the notion of relatively continuity permits a bound that is as good as that from the traditional framework i.e. with smoothness assumption. Moreover, the new framework uses functional links between the objective function and the proxy function, which may be extended for a wide variety of notions (e.g. relatively smoothness, as introduced in paper [2]).

Promising Future Research We have identified the following promising further research that arise from this paper:

1. find even more functional properties to extend the notion of relativity on.
2. find a *universal* procedure to choose the best proxy function $h(.)$.
3. conduct thorough analysis on the cases where this method outperforms others (e.g. ISTA, splitting subgradient-type method, quasi-Newton method, etc.) in terms of computational complexity (same bound, but multiplicative parameters?).
4. analyze this method in sparse vector concepts (e.g. LASSO).

We will be focusing on 3. and 4. in what follows.

2 Solving LASSO with Relative Continuity

In this section, we try to solve a LASSO regression problem by using the notion of relative continuity. Thus, our problem of interest is 1.9, reformulated (by interchanging variable names for convenience) as:

$$\textbf{LASSO} : \min_{\mathbf{w}\in\mathbb{R}^d} f(\mathbf{w}) = \frac{1}{n}\|\mathbf{X}\mathbf{w}-\mathbf{y}\|_2^2 + \lambda\|\mathbf{w}\|_1 \tag{2.1}$$

where we recall that f is **not** smooth (non-differentiable due to ℓ_1-norm) and non-Lipschitz (due to ℓ_2-norm).

2.1 Subgradient Analysis

We start by computing a subgradient of the objective function, and try to bound it, as this will give us the desired information in order to design our algorithm.

2.1.1 Computing a Subgradient

We propose computing a subgradient of our objective function as follows:

- gradient of the MSE term:
$$\frac{2}{n}(\mathbf{X}^\top\mathbf{X}\mathbf{w} - \mathbf{X}^\top\mathbf{y})$$
- subgradient of the regularizer term:
$$\lambda\mathbf{q},\ \text{where } q_i = \begin{cases} +1 & w_i \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

Thus, a subgradient of our objective at any $\mathbf{x}\in\mathbb{R}$ is given by:

$$\mathbf{g} = \frac{2}{n}(\mathbf{X}^\top\mathbf{X}\mathbf{w} - \mathbf{X}^\top\mathbf{y}) + \lambda\mathbf{q}$$

2.1.2 Bounding $||\mathbf{g}||_2$

Observe that:

$$\begin{aligned}
||\mathbf{g}||_2^2 &= \left\|\frac{2}{n}(\mathbf{X}^\top\mathbf{X}\mathbf{w} - \mathbf{X}^\top\mathbf{y}) + \lambda\mathbf{q}\right\|_2^2 \\
&\leq \frac{4}{n^2}\|\mathbf{X}^\top\mathbf{X}\mathbf{w}\|_2^2 + \frac{4}{n^2}\|\mathbf{X}^\top\mathbf{y}\|_2^2 + \|\lambda\mathbf{q}\|_2^2 \\
&\quad + \frac{8}{n^2}\|\mathbf{X}^\top\mathbf{X}\mathbf{w}\|_2\|\mathbf{X}^\top\mathbf{y}\|_2 + \frac{4}{n}\|\mathbf{X}^\top\mathbf{X}\mathbf{w}\|_2\|\lambda\mathbf{q}\|_2 + \frac{4}{n}\|\mathbf{X}^\top\mathbf{y}\|_2\|\lambda\mathbf{q}\|_2 \\
&\leq \frac{4}{n^2}\|\mathbf{X}^\top\mathbf{X}\|_2^2\|\mathbf{w}\|_2^2 + \frac{4}{n^2}\|\mathbf{X}^\top\mathbf{y}\|_2^2 + \lambda^2 n \\
&\quad + \frac{8}{n^2}\|\mathbf{X}^\top\mathbf{X}\|_2\|\mathbf{w}\|_2\|\mathbf{X}^\top\mathbf{y}\|_2 + \frac{4}{n}\|\mathbf{X}^\top\mathbf{X}\|_2\|\mathbf{w}\|_2\lambda\sqrt{n} + \frac{4}{n}\|\mathbf{X}^\top\mathbf{y}\|_2\lambda\sqrt{n} \\
&\leq \frac{4}{n^2}\sigma^2\|\mathbf{w}\|_2^2 + \frac{4}{n^2}r^2 + \lambda^2 n + \frac{8}{n^2}\sigma\|\mathbf{w}\|_2 r + \frac{4}{\sqrt{n}}\sigma\|\mathbf{w}\|_2\lambda + \frac{4}{\sqrt{n}}r\lambda \\
&\quad,\ \text{where } \sigma = \sigma_{\max}(\mathbf{X}^\top\mathbf{X}),\ \text{and } r = \|\mathbf{X}^\top\mathbf{y}\|_2
\end{aligned}$$

Thus, we obtain the below final bound:

$$||\mathbf{g}||_2^2 = \left(\frac{4}{n^2}\sigma^2\right)\|\mathbf{w}\|_2^2 + \left(\frac{8}{n^2}\sigma r + \frac{4}{\sqrt{n}}\sigma\lambda\right)\|\mathbf{w}\|_2 + \left(\frac{4r}{n} + \lambda\sqrt{n}\right)^2 \tag{2.2}$$

2.2 Finding the Appropriate Proxy Function

Using proposition 5.2. of paper [1], we set our proxy function as:

$$h(\mathbf{w}) := \frac{\alpha}{4}\|\mathbf{w}\|_2^4 + \frac{\beta}{3}\|\mathbf{w}\|_2^3 + \frac{\gamma}{2}\|\mathbf{w}\|_2^2 \tag{2.3}$$

which has the below gradient:

$$\nabla h(\mathbf{w}) := \alpha\|\mathbf{w}\|_2^2\mathbf{w} + \beta\|\mathbf{w}\|_2\mathbf{w} + \gamma\mathbf{w} \tag{2.4}$$

2.3 Solving the Inner Minimization Problem

Recall that each iteration requires solving the below minimization problem:

$$\mathbf{w}^{i+1} \leftarrow \arg\min_{\mathbf{w}\in\mathbb{R}^d}\{f(\mathbf{w}^i) + \langle\bar{\mathbf{g}}(\mathbf{w}^i), \mathbf{w} - \mathbf{w}^i\rangle + \frac{1}{t_i}D_h(\mathbf{w}, \mathbf{w}^i)\} \tag{2.5}$$

which actually reduces to solving:

$$\mathbf{w}^{i+1} \leftarrow \arg\min_{\mathbf{w}\in\mathbb{R}^d}\{\bar{\mathbf{g}}(\mathbf{w}^i)^\top\mathbf{w} + \frac{1}{t_i}(h(\mathbf{w}) - \nabla h(\mathbf{w}^i)^\top\mathbf{w})\} \tag{2.6}$$

We observe that this can be written as:

$$\begin{aligned}
&\arg\min_{\mathbf{w}\in\mathbb{R}^d}\bar{\mathbf{g}}(\mathbf{w}^i)^\top\mathbf{w} + \frac{1}{t_i}(h(\mathbf{w}) - \nabla h(\mathbf{w}^i)^\top\mathbf{w}) \\
=&\arg\min_{\mathbf{w}\in\mathbb{R}^d}(t_i\bar{\mathbf{g}}(\mathbf{w}^i) - \nabla h(\mathbf{w}^i))^\top\mathbf{w} + h(\mathbf{w}) \\
=&\arg\min_{\mathbf{w}\in\mathbb{R}^d}\langle\mathbf{k}, \mathbf{w}\rangle + \sum_{i=0}^{2}\frac{a_i}{i+2}\|\mathbf{w}\|_2^{i+2} \\
&\text{, where } \mathbf{k} = t_i\bar{\mathbf{g}}(\mathbf{w}^i) - \nabla h(\mathbf{w}^i) \text{ is independent of } \mathbf{w}, \\
&\text{, and } a_2 = \alpha, a_1 = \beta, a_0 = \gamma
\end{aligned}$$

The First-Order Optimality Condition on the above minimization is given below:

$$\mathbf{k} + \left(\sum_{i=0}^{2}a_i||\mathbf{w}||_2^i\right)\mathbf{w} = \mathbf{0}$$

which leads us to $\mathbf{w} = -\theta\mathbf{k}$, with $\theta \geq 0$ being the solution to the below equation:

$$\sum_{i=0}^{2}a_i\|\mathbf{k}\|_2^i\theta^{i+1} - 1 = 0$$

which expands in our case to:

$$\alpha\|\mathbf{k}\|_2^2\theta^3 + \beta\|\mathbf{k}\|_2\theta^2 + \gamma\theta - 1 = 0$$

Thus, each of our iterations will involve finding the biggest (actually only) real root of the above cubic polynomial.

3 Results + Compare/ Contrast with ISTA

In this section, we provide various plots on the performance of mirror descent with relative continuity RC and ISTA, as well as the way both methods *promote* sparsity, in the context of LASSO. We first note that the LASSO objective function is **not** strongly convex; it is however **locally** strongly convex for small λ, as the quadratic term takes *most* of the objective value. This means that mirror descent will relative continuity will perform worse for larger λ, as it appears in figure 3.1.

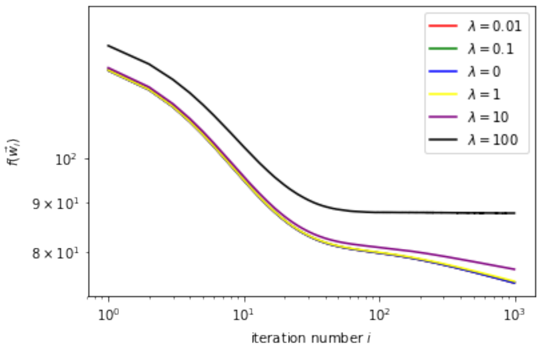

Figure 3.1: Mirror Decent with Relative Continuity on LASSO - Varying λ

Moreover, ISTA's convergence bound is **not** affected (at least even for large λ) by the non-strong convexity of our objective function. Thus, we should expect the convergence bounds of table 3.1:

	Small λ	**Large** λ
RC	$O\left(\frac{1}{k}\right)$	$O\left(\frac{1}{\sqrt{k}}\right)$
ISTA	$O\left(\frac{1}{k}\right)$	$O\left(\frac{1}{k}\right)$

Table 3.1: RC VS ISTA (in terms of λ)

3.1 RC VS ISTA - Convergence Rate

We elaborate here on the previous claim, and provide in figures 3.2,3.3,3.4 the convergence rates of RC and ISTA on LASSO (on the sklearn Boston dataset) for $\lambda \in \{0.01, 100, 10000\}$.

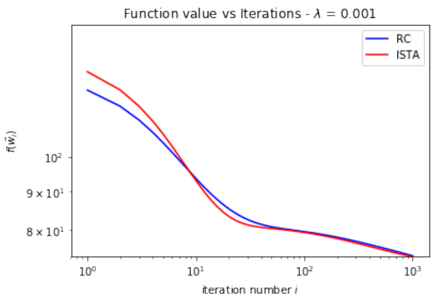

Figure 3.2: Convergence Contrast - $\lambda = 0.001$

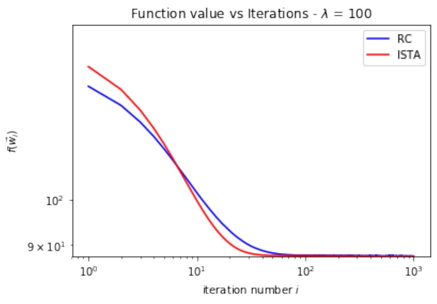

Figure 3.3: Convergence Contrast - $\lambda = 100$

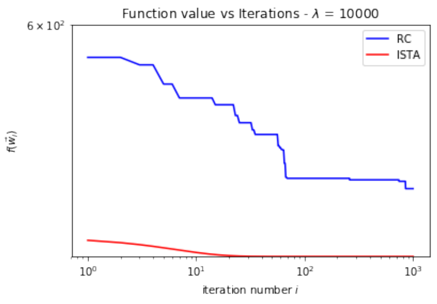

Figure 3.4: Convergence Contrast - $\lambda = 10000$

We observe as expected that, for *relatively* small λ, both algorithms perform as well, while RC starts performing bad as the LASSO regularizer becomes significant in the objective, which thus stops being locally strongly convex.

3.2 RC VS ISTA - Sparsity

We start by providing below the progression of the 12 coefficients of our weight vector for *small* $\lambda = 0.01$. We observe in figures 3.5,3.6,3.7,3.8,3.9,3.10,3.11,3.12,3.13,3.14,3.15,3.16, that, for RC, **all** coefficients do not cancel out, while ISTA cancels out 2 of them.

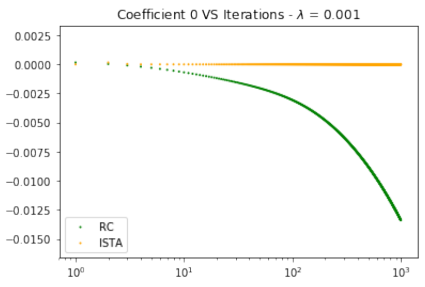

Figure 3.5: Sparsity Contrast - $\lambda = 0.001$

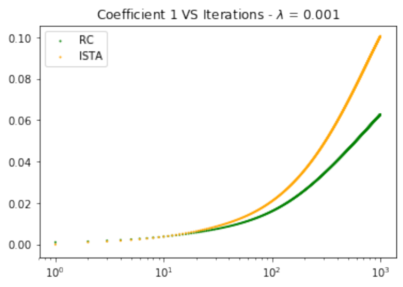

Figure 3.6: Sparsity Contrast - $\lambda = 0.001$

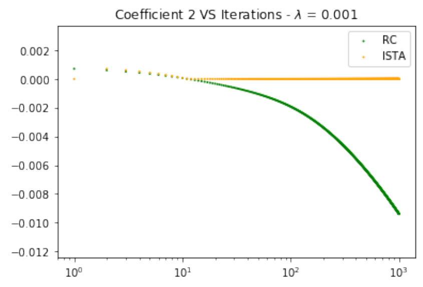

Figure 3.7: Sparsity Contrast - $\lambda = 0.001$

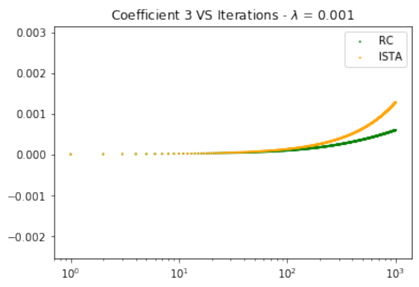

Figure 3.8: Sparsity Contrast - $\lambda = 0.001$

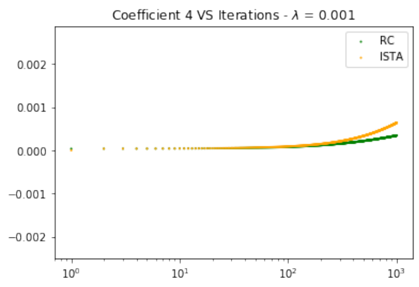

Figure 3.9: Sparsity Contrast - $\lambda = 0.001$

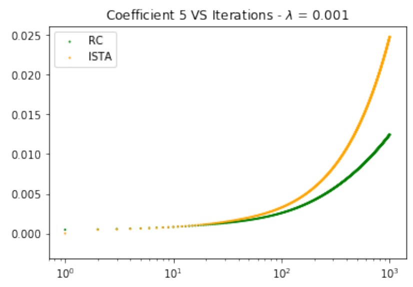

Figure 3.10: Sparsity Contrast - $\lambda = 0.001$

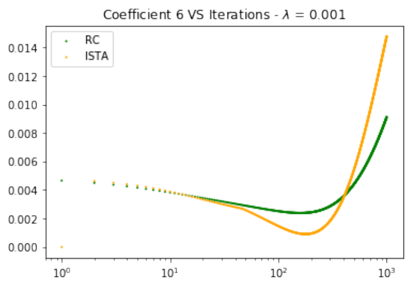

Figure 3.11: Sparsity Contrast - $\lambda = 0.001$

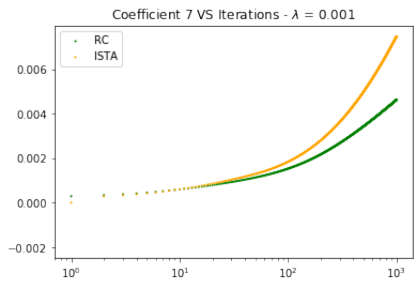

Figure 3.12: Sparsity Contrast - $\lambda = 0.001$

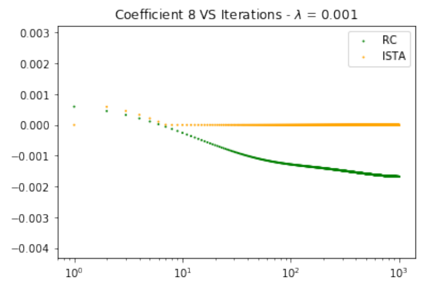

Figure 3.13: Sparsity Contrast - $\lambda = 0.001$

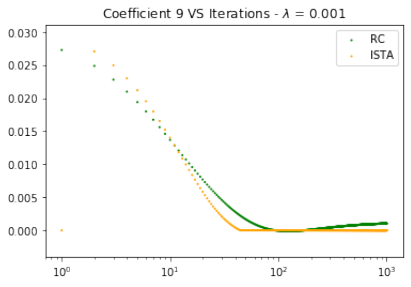

Figure 3.14: Sparsity Contrast - $\lambda = 0.001$

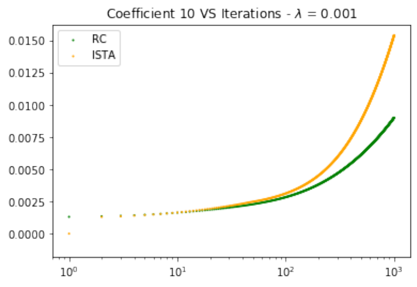

Figure 3.15: Sparsity Contrast - $\lambda = 0.001$

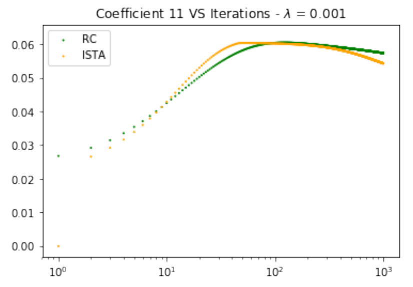

Figure 3.16: Sparsity Contrast - $\lambda = 0.001$

We then provide below the progression of the 12 coefficients of our weight vector for *moderate* $\lambda = 100$. We observe in figures 3.17,3.18,3.19,3.20,3.21,3.22,3.23,3.24,3.25,3.26,3.27,3.28, that both algorithms *eventually* set up the appropriate coefficients to zero, although RC shows some pseudo-periodic oscillations till an asymptotic zero for these coefficients. That is, RC does **not** promote (fast) sparsity.

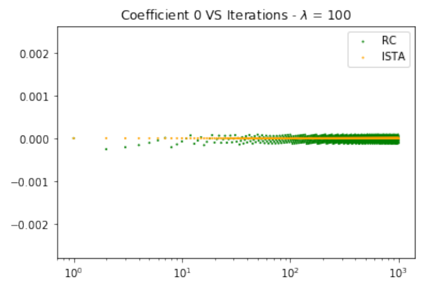

Figure 3.17: Sparsity Contrast - $\lambda = 100$

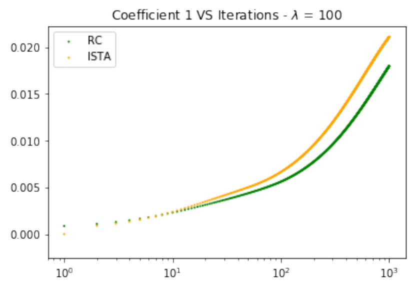

Figure 3.18: Sparsity Contrast - $\lambda = 100$

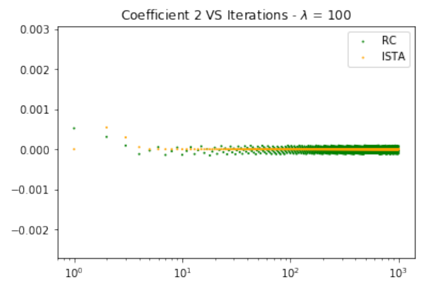

Figure 3.19: Sparsity Contrast - $\lambda = 100$

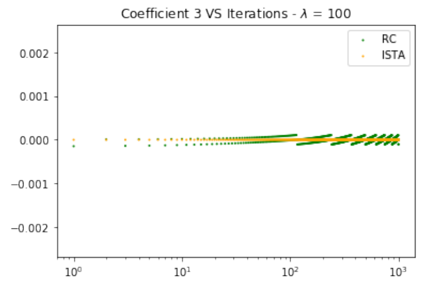

Figure 3.20: Sparsity Contrast - $\lambda = 100$

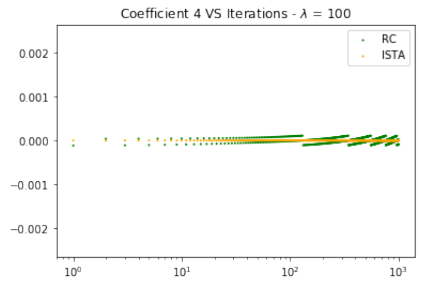

Figure 3.21: Sparsity Contrast - $\lambda = 100$

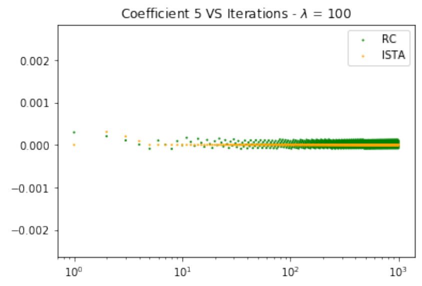

Figure 3.22: Sparsity Contrast - $\lambda = 100$

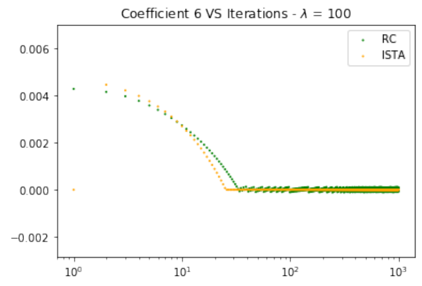

Figure 3.23: Sparsity Contrast - $\lambda = 100$

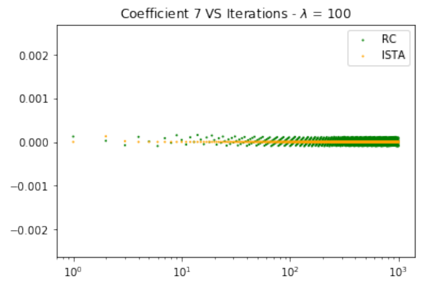

Figure 3.24: Sparsity Contrast - $\lambda = 100$

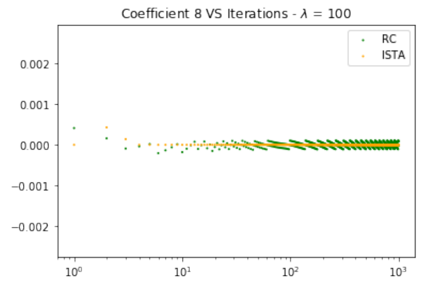

Figure 3.25: Sparsity Contrast - $\lambda = 100$

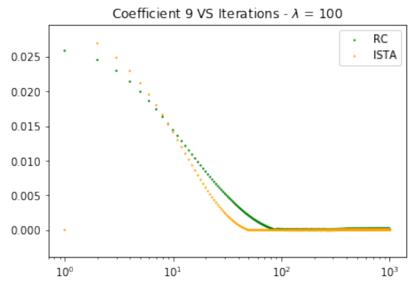

Figure 3.26: Sparsity Contrast - $\lambda = 100$

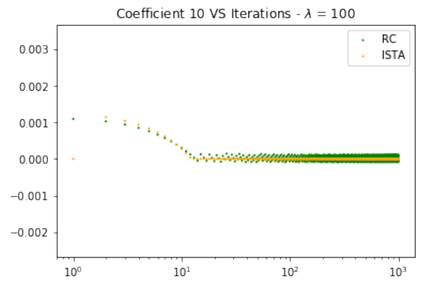

Figure 3.27: Sparsity Contrast - $\lambda = 100$

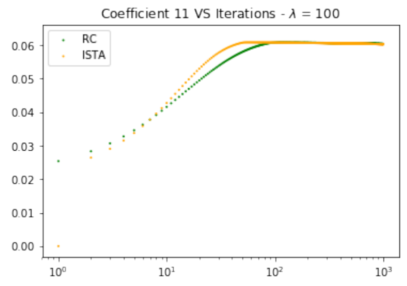

Figure 3.28: Sparsity Contrast - $\lambda = 100$

We finally provide below the progression of the 12 coefficients of our weight vector for *large* $\lambda = 10000$. We observe in figures 3.29,3.30,3.31,3.32,3.33,3.34,3.35,3.36,3.37,3.38,3.39,3.40, that while ISTA *clearly* promotes sparsity, RC becomes even more unstable (similarly to its convergence rate).

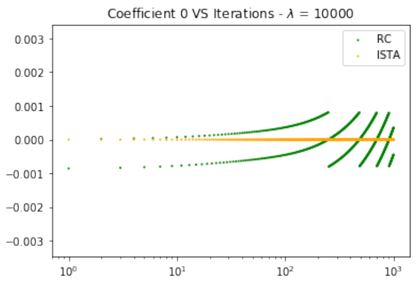

Figure 3.29: Sparsity Contrast - $\lambda = 10000$

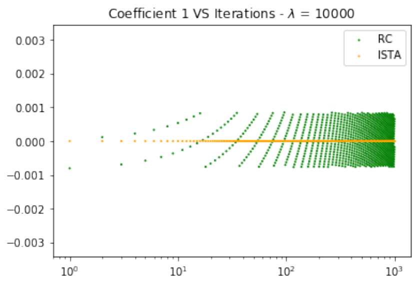

Figure 3.30: Sparsity Contrast - $\lambda = 10000$

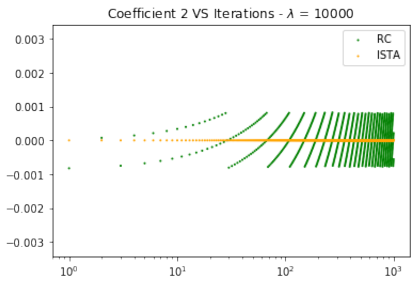

Figure 3.31: Sparsity Contrast - $\lambda = 10000$

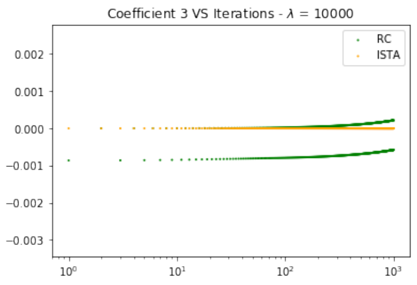

Figure 3.32: Sparsity Contrast - $\lambda = 10000$

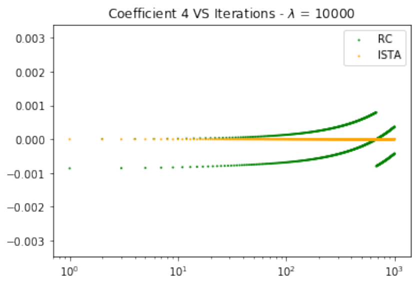

Figure 3.33: Sparsity Contrast - $\lambda = 10000$

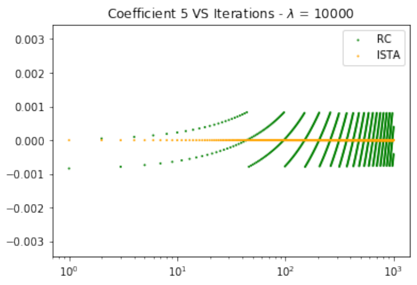

Figure 3.34: Sparsity Contrast - $\lambda = 10000$

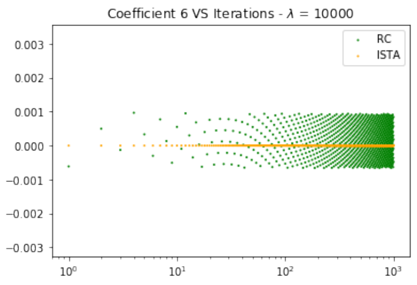

Figure 3.35: Sparsity Contrast - $\lambda = 10000$

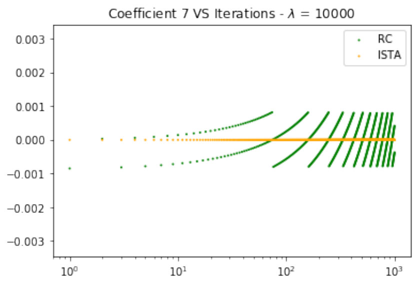

Figure 3.36: Sparsity Contrast - $\lambda = 10000$

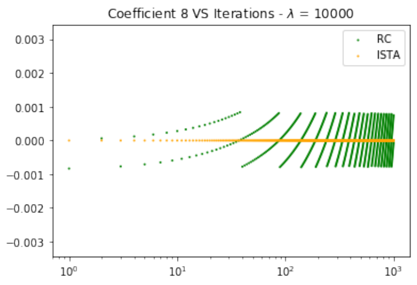

Figure 3.37: Sparsity Contrast - $\lambda = 10000$

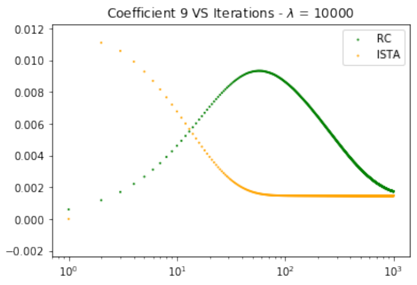

Figure 3.38: Sparsity Contrast - $\lambda = 10000$

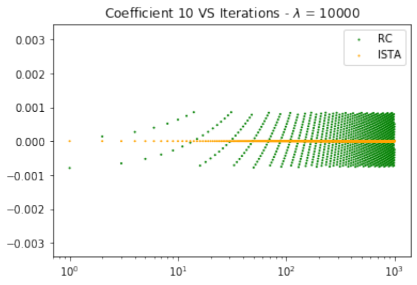

Figure 3.39: Sparsity Contrast - $\lambda = 10000$

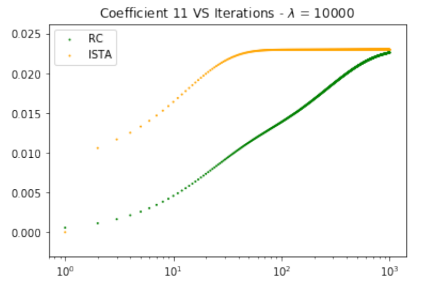

Figure 3.40: Sparsity Contrast - $\lambda = 10000$

References

[1] Haihao Lu, *“Relative-Continuity” for Non-Lipschitz Non-Smooth Convex Optimization using Stochastic (or Deterministic) Mirror Descent*. arXiv:1710.04718v2 [math.OC] 20 Oct 2017

[2] Haihao Lu, Robert M. Freund, Yurii Nesterov, *Relatively-Smooth Convex Optimization by First-Order Methods, and Applications*. June 19, 2017.

A Python Code for Relative Continuity

A.1 Python code for evaluating the LASSO objective

```python
# this function evaluates the LASSO objective function at w for a given regulatizer lamb
# inputs:
#      _ X,y (feature matrix and label vector)
#      _ w (point to evaluate at)
#      _ lamb (regularizer)
def f(w,X,y,lamb):
    N = X.shape[0]
    val = (1/N)*(LA.norm(np.matmul(X,w)-y)**2) + lamb*LA.norm(w,ord = 1)

    return val
```

A.2 Python code for getting a LASSO subgradient

```python
# this function returns a subgradient of the LASSO objective function at w for a given regulatizer lamb
# inputs:
#      _ X,y (feature matrix and label vector)
#      _ w (point to evaluate at)
#      _ lamb (regularizer)
def g(w,X,y,lamb):
    # a subgradient of the mse
    N = X.shape[0]
    Df = (2/N)*(np.matmul(np.matmul(X.T,X),w) - np.matmul(X.T,y))
    # a subgradient of the regularization part
    q = np.zeros(w.shape)
    q[w >= 0] = 1
    q[w < 0] = -1

    return Df + lamb*q
```

A.3 Python code for computing α, β, γ

```python
# this function computes the bound parameters alpha, beta, gamma
# inputs:
#      _ X,y (feature matrix and label vector)
#      _ lamb (regularizer)
def compute_bound_params(X,y,lamb):
    N = X.shape[0]
    # Computing the relevant quantities
    _, sigmas, _ = LA.svd(np.matmul(X.T,X)) # Eig of X.T@X
    r = LA.norm(np.matmul(X.T,y)) # Norm of X.T@Y
    # Defining sigma,ro,gamma
    alpha = (4/(N**2))*(sigmas[0]**2)
    beta = (8/(N**2))*sigmas[0]*r + (4/np.sqrt(N))*sigmas[0]*lamb
    gamma = ( (4*r/N) + lamb*np.sqrt(N) )**2

    return alpha,beta,gamma
```

A.4 Python code for the inner-minimization problem

```python
# this function computes the inner arg-min
# inputs:
#      _ X,y (feature matrix and label vector)
#      _ w_i (point to evaluate at)
#      _ lamb (regularizer)
#      _ ro,sig,gamma (bound params)
#      _ step (step size to take)
def find_w_new(w_i,X,y,lamb,alpha,beta,gamma,ti):
    Dhi = (alpha*LA.norm(w_i)**2 + beta*LA.norm(w_i) + gamma)*w_i
    gi = g(w_i,X,y,lamb)
    k = ti*gi - Dhi
    a = alpha*(LA.norm(k)**2)
    b = beta*LA.norm(k)
    c = gamma
    d = -1
    # here we find a root of the cubic polynomial
    p = [a,b,c,d]
    roots = np.roots(p)
    roots = roots[~np.iscomplex(roots)]
    roots = np.real(roots)
    theta_star = np.max(roots)

    w_ip1 = -theta_star*k

    return w_ip1
```

A.5 Python code for the mirror descent algorithm

```python
# this function performs mirror descent with provided parameters
# inputs:
#       - lambda (regularization parameter)
#       - n_iter (number of iterations for which to run the descent algorithm)
#       - t (step size)
# outputs:
#       - fs (history of function evaluations)
#       - wis (history of weights)
def mirror_descent(lamb, n_iter, t):
    w_i = np.zeros((X.shape[1],1)) # initial point
    fs = []
    wis = []
    for i in range(n_iter):
        w_i = find_w_new(w_i,X,y,lamb,alpha,beta,gamma,t)
        fs.append(f(w_i,X,y,lamb))
        wis.append(w_i)
    return fs, wis
```

B Python Code for ISTA

B.1 Python code for all the ISTA framework

```
class ISTA:

    def __init__(self,lamda):

        self.lamda = lamda


    def evaluate_f(self,X,y,w):

        return (1/X.shape[0]*LA.norm(y-X@w)**2 + self.lamda*LA.norm(w,ord=1))

    def gradf(self,X,y,w):

        return (1/X.shape[0]*(2*X.T@X@w - 2*X.T@y))

    def solve_subproblem(self,X,y,w_k,L):
        '''
        L is the the smoothness coefficient of f
        '''
        c = w_k - 1/L*self.gradf(X,y,w_k)
        l = self.lamda/L
        w_next =np.sign(c)*np.maximum(c-l*np.ones(c.shape[0]),0)
        return (w_next)


    def evaluate_L(self,X):
        _,sigmas,_ = LA.svd(X.T@X)
        return(2/X.shape[0]*(sigmas[0]))

    def fit(self,X,y,w_init = np.zeros(X.shape[1]),num_iter = 1000):
        '''
        Fitting the data(solving the optim)
        '''
        L = self.evaluate_L(X)
        weights = []
        function_value = []
        w = w_init
        weights.append(w)
        function_value.append(self.evaluate_f(X,y,w))

        for i in range(num_iter):

            w = self.solve_subproblem(X,y,w,L)
            weights.append(w)
            function_value.append(self.evaluate_f(X,y,w))

        self.w = weights[-1]
        self.wis = np.array(weights)

        return (weights,function_value)

    def predict(self,X_new):

        return (X_new@self.w)
```

C Python Code for Various Plots

C.1 Python code for comparing/ contrasting RC and ISTA

```python
boston = load_boston()
lambs = [0.001,0.01,0.1,1,10,100]
for lamb in lambs:
    X = boston.data
    y = boston.target
    y = y.reshape((y.shape[0],1))
    # solve RC
    alpha,beta,gamma = compute_bound_params(X,y2,lamb)
    n_iter = 1000
    t = 6500
    fs, wis = mirror_descent(lamb,n_iter,t)
    wis = np.array(wis)
    # solve ISTA
    y = boston.target
    ista = ISTA(lamda = lamb)
    weights,function_value = ista.fit(X,y)
    # plot comparison
    iterations = np.arange(len(function_value))
    plt.loglog(fs, color='blue', label = 'RC')
    plt.loglog(function_value, color='red', label = 'ISTA')
    plt.xlabel('iteration number ' + r'$i$')
    plt.ylabel(r'$f(\vec w_i)$')
    plt.ylim((0,150))
    plt.title('Function value vs Iterations - ' + r'$\lambda$' + ' = ' + str(lamb))
    plt.legend()
    plt.show()
    # other plot comparison
    itr = [i for i in range(1,1002)]
    s = [1 for i in range(1001)]
    for i in range(13):
        plt.title('Coefficient ' + str(i) + ' VS Iterations - ' + r'$\lambda$' + ' = ' + str(lamb))
        ax = plt.gca()
        ax.scatter(itr[:-1],wis[:,i],s=s, color='green', label = 'RC')
        ax.scatter(itr,ista.wis[:,i],s=s, color='orange', label = 'ISTA')
        ax.set_xscale('log')
        plt.legend()
        plt.show()
```