RICE UNIVERSITY

Parallel sparse optimization

by

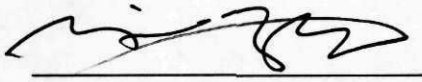
Zhimin Peng

A Thesis Submitted
in Partial Fulfillment of the
Requirements for the Degree

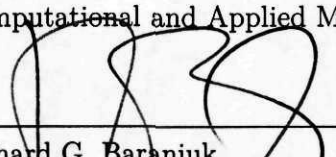
Master of Arts

Approved, Thesis Committee:

Wotao Yin, Chair
Associate Professor
Computational and Applied Mathematics



Yin Zhang
Professor
Computational and Applied Mathematics



Richard G. Baraniuk
Victor E. Cameron Professor
Electrical and Computer Engineering

Houston, Texas

May, 2013

Abstract

Parallel Sparse Optimization

by

Zhimin Peng

Master of Arts (Computational and Applied Mathematics)

Supervisor: Professor Wotao Yin

Rice University

This thesis proposes parallel and distributed algorithms for solving very large-scale sparse optimization problems on computer clusters and clouds. Many modern applications problems from compressive sensing, machine learning and signal and image processing involve large-scale data and can be modeled as sparse optimization problems. Those problems are in such a large-scale that they can no longer be processed on single workstations running single-threaded computing approaches. Moving to parallel/distributed/cloud computing becomes a viable option. I propose two approaches for solving these problems. The first approach is the distributed implementations of a class of efficient proximal linear methods for solving convex optimization problems by taking advantages of the separability of the terms in the objective. The second approach is a parallel greedy coordinate descent method (GRock), which greedily chose several entries to update in parallel in each iteration. I establish the convergence of GRock and explain why it often performs exceptionally well for sparse optimization. Extensive numerical results on a computer cluster and Amazon EC2 demonstrate the efficiency and elasticity of my algorithms.

Keywords: Sparse optimization, parallel and distributed computing, prox-linear methods, GRock.

Acknowledgments

I would like to express my deepest appreciation to all of those who provided me the possibility to complete this thesis. My specific thanks go to the following people.

To Dr. Wotao Yin, thank you for introducing this neat large-scale optimization problem to me. Your encouragement, guidance and support are essential for shaping me into a rigorous mathematical researcher; your enthusiasm, diligence and intelligent have deeply motivated me. Without your guidance and persistent assistance, this dissertation would not be possible.

To Drs. Yin Zhang and Richard Baraniuk, thank you for providing valuable advises and corrections to my thesis. I especially appreciate your support by serving on my committee. Your knowledge and insights greatly enriched my work.

To my collaborator Dr. Ming Yan. Thank you for sharing techniques and thoughts about research. Your patient for my endless disturbing is greatly appreciated.

To Drs. Matthias Heinkenschloss, Jan Hewitt and Danny Sorensen. Thank you for helping me improve my writing and presentation skills.

I would also like to thank the staff and students in the CAAM department. Especially, I owe my deepest gratitude to Etienne Ackermann, who helped me with all kinds of Latex and Matlab problems.

To all of my friends. Thank you for supporting me to get through my time in Rice.

To my family. Your unconditional love and support are greatly appreciated. I would like to dedicate this thesis to my parents.

Table of Contents

List of Illustrations vi

List of Tables viii

1 Introduction 1

1.1 Compressive Sensing . . . 2

1.1.1 Greedy algorithms . . . 3

1.1.2 ℓ_1 relaxation . . . 4

1.2 Machine Learning . . . 6

1.2.1 Sparse linear regression . . . 6

1.2.2 Sparse classification methods . . . 7

1.3 Parallel and Distributed Algorithms . . . 8

2 Problem Statement 11

2.1 Regularizer . . . 12

2.2 Loss function . . . 13

2.3 Problem Formulation . . . 14

2.3.1 Non-smoothness . . . 16

2.3.2 Large Dataset . . . 17

3 Parallel Algorithms 22

3.1 Parallel Prox-linear Algorithms . . . 23

3.1.1 Example: distributed LASSO . . . 25

3.1.2 Example: distributed sparse logistic regression . . . 27

3.1.3 Convergence analysis . . . 28

3.2 Parallel Greedy Coordinate Descent Methods 29
3.2.1 Algorithm 29
3.2.2 GRock demonstration 31
3.2.3 Convergence analysis 33
3.3 Parallel Alternating Direction Method of Multipliers 39
3.3.1 Parallel the original ADMM 40
3.3.2 Apply ADMM to the primal problem 41
3.3.3 Apply ADMM to the dual problem 42
3.3.4 An example: LASSO 45
3.3.5 Convergence analysis 46

4 Numerical Experiments 48
4.1 LASSO 48
4.1.1 Greedy rocks for sparse optimization 48
4.1.2 Synthetic small datasets 52
4.1.3 Synthetic large datasets 55
4.2 Sparse Logistic Regression 56
4.2.1 Synthetic small datasets 56
4.2.2 Real datasets 58

5 Conclusion and Future Work 60
5.1 Conclusion 60
5.2 Future Work 61
5.2.1 Convergence rate for GRock 61
5.2.2 Asynchronous algorithms 61
5.2.3 Regularization with linear transform 62

Bibliography 63

Illustrations

2.1 Regularizers. ℓ_1 norm is a piecewise linear function, and it is non-smooth; elastic net function gives an upper bound for ℓ_1 norm, which is also non-smooth; Huber norm lies between ℓ_1 norm and ℓ_2 norm, which can be regarded as a smooth approximation of ℓ_1 norm. 13

2.2 Proximal operators. $\text{shrink}(x, 1)$ is the proximal operator for ℓ_1 norm with $\lambda = 1$; $0.5\,\text{shrink}(x, 1)$ gives the proximal operator for elastic net function with $\lambda = \alpha = 1$; proximal operator for Huber norm with $\lambda = \mu = 1$ is given by $\mathbf{prox}_{Huber}(x)$. 17

2.3 Data distribution scenarios. Left: row block partition; middle: column block partition; right: general block partition 19

3.1 Example I. GRock with $P = 1$ needs two steps to converges to the optimal solution; GRock with $P = 2$ converges to the optimal solution with only one step. 32

3.2 Example II. GRock with $P = 1$ converges to the optimal solution; GRock with $P = 3$ diverges. 33

4.1 A comparison of different coordinate descent methods. The left figure illustrates the convergence with respect to objective function; the right figure demonstrate the convergence with respect to the relative error. 49

4.2 A comparison of different coordinate descent methods with different sparsity level and dynamic range of $\mathbf{x}$; the ith column corresponding to the range of 10^i; the color represents number of iterations. 50
4.3 A comparison of different coordinate descent methods with different sparsity level and dynamic range of $\mathbf{x}$; the ith column corresponding to the range of 10^i; the color represents number of iterations. 51
4.4 A comparison of different coordinate descent methods and FISTA. . . 52
4.5 Cores vs iterations 53
4.6 Cores vs time 54
4.7 Cores vs communication overhead percentage 54
4.8 Dataset I 57
4.9 Dataset II 58

Tables

2.1 problem scale quantification 18

4.1 datasets tested on Rice cluster 52
4.2 large dataset time results 56
4.3 Logistic regression datasets 57
4.4 Parallel time results for the UCI adult dataset 59

Chapter 1

Introduction

Technological advances in data gathering have led to a rapid proliferation of big data in diverse areas such as the Internet, engineering, climate studies, cosmology, and medicine. In order to interpret this massive amount of data, new computational approaches are being introduced to let scientists and engineers analyze their data in a parallel and distributed manner. Among these approaches, structured solutions, and in particular sparse solutions, have become enormously important.

Many applications from compressive sensing, machine learning and image processing are modeled as sparse optimization[1] problems. Big data involved in those problems result in large-scale sparse optimization problems which can no longer be processed on single workstations running single-threaded computing approaches. Moving to parallel/distributed/cloud computing becomes a viable option.

This thesis introduces two novel approaches for solving large-scale sparse optimization problems that leverage the vast amounts of available computing resources. Our two approaches are motivated by two structures present in sparse optimization: separable objective functions and data "near-orthogonality", which are described in the next chapter. In the first approach, we parallelize the existing prox-linear algorithms

[1]Note that sparse optimization does not necessarily involve sparse linear algebra.

by taking advantages of the separability of the terms in the objective. The second approach is developed based on greedy coordinate-block (GRock) update. At each iteration, it selects a few blocks of variables and then a few variables in each selected block, both by greedy means, and updates the latter in parallel. In fact, under certain orthogonality conditions (such as the RIP [10] and incoherence conditions [41]), certain greedy rules are guaranteed to select the coordinates corresponding to nonzero entries in the final solution.

In this chapter, I will briefly review the sparse optimization problems arising from compressive sensing (see section 1.1) and machine learning (see section 1.2). Section 1.3 looks back some of the recent works for parallel sparse optimization problems.

1.1 Compressive Sensing

Compressive sensing, a new and efficient way for signal acquisition [16, 9], is regarded as one of the most popular sparse optimization problems in the past decade. Compared to the conventional Shannon sampling theorem which says that signals can only be recovered by using a sampling rate larger than twice the maximum frequency of the signal, compressive sensing asserts that certain signals and images can be recovered from far fewer samples or measurements based on the following two tenets [11]:

- **Sparsity**. Signals can be sparsely represented in a certain basis Ψ;
- **Incoherence**. Sparsely represented signal in Ψ has a very dense representation in the acquisition domain Φ.

Let $\mathbf{x} \in \mathbb{R}^n$ be an unknown signal, compressive sensing recovers $\mathbf{x}$ from the observation $\mathbf{b}$ by solving

$$\begin{aligned} &\min_{\mathbf{x}} \|\mathbf{x}\|_0 \\ &\text{s.t. } \mathbf{A}\mathbf{x} = \mathbf{b}, \end{aligned} \tag{1.1}$$

where $\|\mathbf{x}\|_0$ is the ℓ_0 norm which is defined as the number of nonzero elements of $\mathbf{x}$. The sensing matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ can be constructed by setting $\mathbf{A} = \mathbf{R}\Phi\Psi$ where $\mathbf{R} \in \mathbb{R}^{m \times n}$ is the matrix that extracts m samples from measurements Φ, and Ψ is the basis under which $\mathbf{b}$ has a sparse representation. Model (1.1) recovers the sparsest solution $\mathbf{x}$ from the linear system. Various algorithms have been developed to solve (1.1). In section 1.1.1, some of the greedy algorithms to solve (1.1) are briefly reviewed. In section 1.1.2, we introduce some of the recent algorithms that solve a convex relaxation problem of (1.1).

1.1.1 Greedy algorithms

Greedy algorithms try to identify the support of $\mathbf{x}$ iteratively, after which the pseudo-inverse of the corresponding columns can be used to compute the value of nonzero entries. Orthogonal Matching Pursuit (OMP) introduced by Tropp [42] is one of the first greedy algorithms used for compressive sensing. It iteratively selects one column of matrix $\mathbf{A}$ which is mostly correlated to the residual $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}$, and the index of the column is added to the support set; $\mathbf{x}$ is updated on the support by solving a least square problem. Stagewise Orthogonal Matching Pursuit (StOMP) developed by Donoho and his collaborators [17] is a variation of the OMP algorithm. Instead

of simply selecting the mostly correlated column, StOMP selects those columns such that the inner products are larger than a given threshold. It then solves a least square problem to update the nonzero entries of x as well as the residual $\mathbf{r}$. Needell and Tropp introduced Compressive Sampling Matching Pursuit (CoSaMP) in 2009 [31] which is another variation of the OMP algorithm. It selects $2K$ (where K is the number of nonzero entries) columns from $\mathbf{A}$ that are mostly correlated to the current residual, then residual $\mathbf{r}$ and variable $\mathbf{x}$ is updated by solving a least square problem on the current support. Finally, $\mathbf{x}$ is updated by applying a hard-thresholding operator which keeps the largest K entries of $\mathbf{x}$ and sets the other entries to 0.

1.1.2 ℓ_1 relaxation

An alternative approach to solve (1.1) is by relaxing it to

$$\begin{aligned} &\min_{\mathbf{x}} \ \|\mathbf{x}\|_1 \\ &\text{s.t. } \mathbf{A}\mathbf{x} = \mathbf{b}, \end{aligned} \tag{1.2}$$

or the equivalent unconstrained problem (namely LASSO [39])

$$\min_{\mathbf{x}} \ \lambda\|\mathbf{x}\|_1 + \frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2. \tag{1.3}$$

Donoho, Candes, Romberg and Tao et al. [16, 9] showed that problem (1.1) and (1.2) are equivalent under certain conditions, i.e., solving (1.2) gives the solution to (1.1). The equivalence between the two problems is importance because (1.2) is an easier problem and can be reformulated as a linear programming problem and then solved by standard linear programming methods such as the simplex method and interior

point method. However, those standard approaches are very expensive for large scale problems. Koh, Kim and Boyd [25] employed a specialized interior-point method that uses a preconditioned conjugate gradient method to approximately solve linear systems in a truncated-Newton framework.

In recent years, a number of more efficient first order methods have been developed. Fixed point continuation (FPC) [22] solves (1.3) by continuously increasing the penalty parameter λ; fast iterative shrinkage-thresholding algorithm (FISTA) introduced by Beck and Teboulle [2] uses Nestrov's acceleration scheme to speed up the soft-thresholding based algorithms; Yang, Zhang and Yin [43] reformulated (2.3) into a variable separable problem, which is solved by the alternating direction method of multipliers (ADMM); Bregman methods, including original Bregman method [45], linear Bregman method [44, 8] and split Bregman method [19] apply Bregman divergence to the ℓ_1 regularizer and obtain fast algorithms.

Block coordinate descent methods are another class of algorithms that have been applied to compressive sensing. For instance, Li and Osher [28] developed a coordinate descent method with convergence guarantee for ℓ_1 minimization. Yun and Toh [47] proposed a block coordinate descent method to solve the general ℓ_1 regularized convex minimization problem with Q-linear convergence rate .

1.2 Machine Learning

Regression and classification are the two core tasks in machine learning. Although the two tasks serve for different purposes, they all try to answer the same question: given a (possibly huge) number of examples together with the known corresponding inferences, can we find a rule that is able to make inferences about future instances? Different models give different rules that explain the given examples and predict the future instances; however, the models formulated as sparse optimization problems seek "simple" rules and outperform the others in the following three senses,

- they provide insight into the most significant features of the data;
- it is easy and cheap to apply them to future instances;
- over-fitting can be avoided to the given examples.

In the following two sections, we will provide some sparse optimization models that serve for the purpose of regression and classification.

1.2.1 Sparse linear regression

Given a data set $\mathbf{A} \in \mathbb{R}^{m\times n}$ and observation $\mathbf{b} \in \mathbb{R}^m$, where m and n represent the number of samples and the number of features respectively, sparse linear regression aims at seeking a sparse vector $\mathbf{x}$ that approximately minimizes $\frac{1}{2}\|\mathbf{A}\mathbf{x}-\mathbf{b}\|_2^2$. It can be modeled as

$$\min_{\mathbf{x}} \ \lambda\|\mathbf{x}\|_1 + \frac{1}{2}\|\mathbf{A}\mathbf{x}-\mathbf{b}\|_2^2,$$

where $\lambda > 0$ is the penalty parameter. Larger λ gives sparser $\mathbf{x}$. Notice that this problem is the same as (2.3), so the algorithms mentioned in section 1.1 can be directly applied. Several other variations of LASSO have also been developed for different regression purposes. For instance, the group LASSO [46] acts like LASSO at the group level (an entire subvector of $\mathbf{x}$ will be zero in the solution) by replacing the regularizer $\|\mathbf{x}\|_1$ with the group sparsity regularizer $\|\mathbf{x}\|_{2,1} = \sum_{i=1}^{s} \|\mathbf{x}_i\|_2$. Another variation is the elastic net regularization [50], which uses $\|\mathbf{x}\|_1 + \frac{1}{2\alpha}\|\mathbf{x}\|_2^2$ instead of $\|\mathbf{x}\|_1$ to not only enjoy a sparse representation but also encourage a group effect (correlated entries of $\mathbf{x}$ tend to be zero or non-zero together) for the solution. Fused LASSO [40] encourages local constancy of the solution by penalizing $\mathbf{x}$ and its successive differences with ℓ_1 norm.

1.2.2 Sparse classification methods

Given a data set $\mathbf{A} \in \mathbb{R}^{m\times n}$ where m and n represent the number of samples and the number of features respectively, and given the labels $\mathbf{b} \in \mathbb{R}^m$ where the value indicate which class the sample belongs to, sparse logistic regression gives a hyperplane classifier $\mathbf{y} = \mathbf{a}^T\mathbf{x} + c$ through solving

$$\min_{\mathbf{x},c} \ \lambda\|\mathbf{x}\|_1 + \frac{1}{m}\sum_{i=1}^{m} \log\left(1 + \exp(-b_i(\mathbf{a}_i^T\mathbf{x} + c))\right). \tag{1.4}$$

Various methods have been applied to solve this problem. Koh, Kim and Boyd [25] proposed an interior point method. Their algorithm takes Newton steps and uses preconditioned conjugated gradient iterations. Sparse multinomial logistic regression

[26], an algorithm for various sparse linear classifiers, can also solve sparse logistic regression. Shi, Yin and Osher [37] proposed a two stage hybrid method where the first stage is based on FPC and the second stage is a customized interior point method.

Sparse support vector machine [4] is another commonly used classification model, which is modeled by

$$\min_{\alpha,c} \ \lambda\|\alpha\|_1 + \frac{1}{m}\sum_{i=1}^{m} \ell(f(\mathbf{a}_i), b_i), \tag{1.5}$$

where $\ell(f(\mathbf{a}_i), b_i) = \max\{0, |f(\mathbf{a}_i) - b_i| - \epsilon\}$, and $f(\mathbf{a}_i) = \sum_j \alpha_j \cdot k(\mathbf{a}_j, \mathbf{a}_i) + c$. Bi et al. [4] solved (1.5) by reformulating it as a linear programming problem.

1.3 Parallel and Distributed Algorithms

Slow increase of the performance of single-processor caused by transmission speeds, limits to miniaturization and economic limitations [2] leads to designing microprocessors in the direction of parallelism. Parallel and distributed computing not only has been used to solve difficult problems in many areas of science and engineering such as communication, computing and sensor network [3, 24], but also attracts people's attention from industry for its ability to process large amount of data in sophisticated ways.

The basic approach for parallelism is by decomposition, which divides a large-scale problem into many smaller ones and solves them in parallel. Various parallel and distributed algorithms have been developed for different numerical computing purposes

[2] https://computing.llnl.gov/tutorials/parallel_comp/

[18], but few parallel methods have been applied to sparse optimization. This section presents some of the existing parallel sparse optimization algorithms.

Lee and Wright [27] implemented SpaRSA on GPU and achieved large speedups over CPU implementations. However, the limited memory of GPU restricts their algorithms from handling problems involving large datasets. Borghi and his collaborators [5] developed a simple compressive sensing algorithm based on prox-linear method that takes advantage of shared memory, parallel and many-core microprocessors. Due to the lack of scalability between memory and CPUs of shared memory system, their implementation does not scale well.

Boyd and his collaborators [6] used ADMM in a distributed scenario to various machine learning problems. Distributed ADMM breaks a large scale problem into some smaller subproblems by introducing copies of variables. Similarly, Mota et al. [30] proposed a distributed ADMM method (D-ADMM) which solved basis pursuit on sensor network under the assumption that the network is connected. However, distributed ADMM does not scale well: given a fixed amount of data, ADMM computation with more nodes does not reduce its running time, because its number of iterations increases with the number of distributed data blocks. The time saved due to a smaller subproblem is offset by the increased number of iterations. This observation motivates us to develop a parallel prox-linear algorithm that does not increase the iterations. Therefore, it will run faster as the data are distributed to more nodes.

Block coordinate descent methods are another class of algorithms that various

parallelism schemes has been introduced to solve large scale sparse optimization problems. Bradley and his collaborators [7] proposed a parallel coordinate descent algorithm for minimizing ℓ_1 regularized losses. They introduced some duplicate variables, transformed the original problem to a differentiable function and applied gradient descent method to the new objective function. Scherrer et al. [36] proposed a parallel block coordinate descent method that clusters features together such that the maximum inner product between features in different blocks is small. Richtárik and Takáč proposed a parallel randomized block coordinate descent method [35] which randomly chose several blocks of coordinates according to certain distribution to update in parallel. However, the above mentioned parallel block coordinate descent methods select the coordinates or blocks in a randomized fashion and disregard the advantage of sparse optimization problems: nonzero components can be efficiently found by greedy methods. Their works together with our observation of the property of sparse optimization motivate us to develop the GRock algorithm.

Some of other recent works [48, 29, 49] analyze parallel stochastic gradient descent method for multicore and distributed settings.

Chapter 2

Problem Statement

We consider the following general form of sparse optimization problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} \ \mathcal{F}(\mathbf{x}) = \lambda \cdot \mathcal{R}(\mathbf{x}) + \mathcal{L}(\mathbf{A}\mathbf{x}, \mathbf{b}), \tag{2.1}$$

where $\mathcal{R}(\mathbf{x})$ is the regularizer, which is usually *block separable* and is used to impose certain structure of the solution. $\mathcal{L}(\mathbf{A}\mathbf{x}, \mathbf{b})$ is a typical smooth data fidelity or loss function. The parameter λ governs the trade-off between fidelity and structure.

Definition 1 (Block separable function)
A function $f(\mathbf{x})$ is block separable if it can be written as $f(\mathbf{x}) = \sum_{i=1}^{N} f_i(\mathbf{x}_i)$, where $\mathbf{x}_i$ is the ith block of $\mathbf{x}$.

For the commonly used regularizers $\mathcal{R}(\mathbf{x})$ such as the ℓ_1 norm, $\ell_{2,1}$ norm, Huber function and elastic net function, we can write $\mathcal{R}(\mathbf{x}) = \sum_{i=1}^{N} r(\mathbf{x}_i)$, i.e., they are block separable. Many of the commonly used loss functions such as the square, logistic and hinge loss function satisfy $\mathcal{L}(\mathbf{A}\mathbf{x}, \mathbf{b}) = \sum_{j=1}^{M} \mathcal{L}(\mathbf{A}_{(j)}\mathbf{x}, \mathbf{b}_j)$ ($\mathbf{A}_{(j)}$ is the jth row block of $\mathbf{A}$). Detailed discussions about the regularizer $\mathcal{R}(\mathbf{x})$ and loss function $\mathcal{L}(\mathbf{A}\mathbf{x}, \mathbf{b})$ are presented in section 2.1 and section 2.2.

2.1 Regularizer

In sparse optimization, various regularizers have been used to control sparsity or structure of the solution, ℓ_1 norm is one of the most commonly used regularizer, which is described as

$$\mathcal{R}(\mathbf{x}) = \|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i|.$$

ℓ_1 norm is convex but not smooth because of its non-differentiability at $\mathbf{x} = 0$.

Elastic net function, firstly introduced in [50], is the following

$$\mathcal{R}(\mathbf{x}) = \|\mathbf{x}\|_1 + \frac{1}{2\alpha}\|\mathbf{x}\|_2^2 = \sum_{i=1}^{n}(|x_i| + \frac{1}{2\alpha}x_i^2).$$

Due to the nonsmoothness of $\|\mathbf{x}\|_1$, the elastic net function is also nonsmooth, but adding the augmented term $\|\mathbf{x}\|_2^2$ is useful for certain sparse optimization problems for it leads to the smoothness of the dual problems [44]. In addition, it encourages a group effect [50] such that correlated entries of $\mathbf{x}$ tend to be zero or non-zero together.

Huber norm [23] is the function that lies between least square and ℓ_1 norm, which is given by $\mathcal{R}(\mathbf{x}) = \sum_{i=1}^{n} r_\mu(x_i)$, where

$$r_\mu(x) = \begin{cases} \dfrac{x^2}{2\mu} & |x| \le \mu \\ |x| - \dfrac{\mu}{2} & |x| \ge \mu. \end{cases}$$

Huber norm is smooth and can be regarded as a smooth approximation of the ℓ_1 norm. It has been broadly used in seismic data analysis [21].

A one dimensional example of the above mentioned norms is given in Figure 2.1.

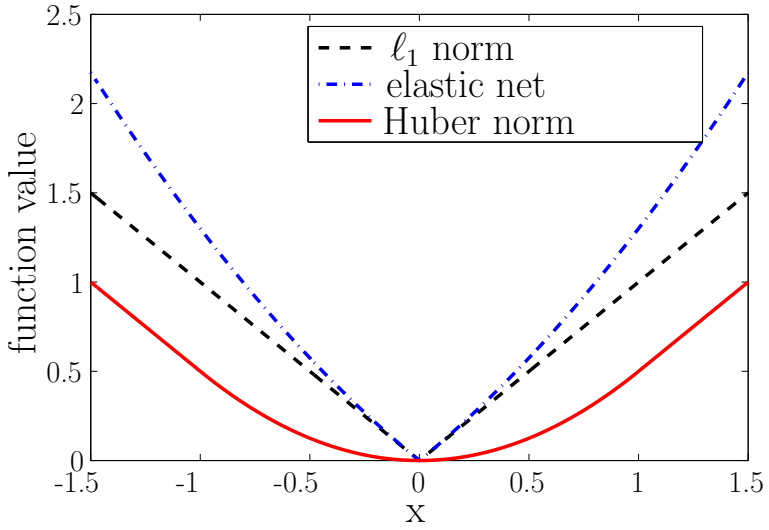


Figure 2.1 : Regularizers. ℓ_1 norm is a piecewise linear function, and it is non-smooth; elastic net function gives an upper bound for ℓ_1 norm, which is also non-smooth; Huber norm lies between ℓ_1 norm and ℓ_2 norm, which can be regarded as a smooth approximation of ℓ_1 norm.

2.2 Loss function

Various loss functions have been used for different purposes. For instance, hinge loss function described by

$$\ell(f(\mathbf{x}), y) = \max\{0, 1 - y \cdot f(\mathbf{x})\},$$

works well for its purpose in SVM as a classifier, while it is not suitable for regression based problems.

Square loss, the most commonly used loss function, is the following

$$\ell(f(\mathbf{x}), y) = (f(\mathbf{x}) - y)^2.$$

It is well suited for regression problems, however, it suffers from one critical flaw: outliers in the data (isolated points that are far from the desired target function) are

punished very heavily by the squaring of the error.

Absolute loss functions is as follows

$$\ell(f(\mathbf{x}), y) = |f(\mathbf{x}) - y|,$$

which is applicable to regression problems, and it can avoid the problem of weighting outliers too harshly.

Logistic loss function is

$$\ell(f(\mathbf{x}), y) = \log(1 + \exp(-y \cdot f(\mathbf{x}))),$$

where $y \in \{-1, 1\}$. It is mainly used for classification purposes.

2.3 Problem Formulation

Assume $\mathcal{R}(\mathbf{x})$ is block separable and $\mathcal{L}(\mathbf{Ax}, \mathbf{b})$ is a summation of loss functions, we can reformulate (2.1) as follows

$$\min_{\mathbf{x}\in\mathbb{R}^n} \lambda \cdot \underbrace{\sum_{i=1}^{N} r(\mathbf{x}_i)}_{\mathcal{R}(\mathbf{x})} + \underbrace{\sum_{j=1}^{m} \ell(\mathbf{a}_j^T \mathbf{x}, b_j)}_{\mathcal{L}(\mathbf{Ax},\mathbf{b})}. \tag{2.2}$$

Some of the examples that has the above formulation include

- the ℓ_1 regularized least square model (also named as LASSO) [39],

$$\min_{\mathbf{x}} \lambda \cdot \|\mathbf{x}\|_1 + \frac{1}{2}\|\mathbf{Ax} - \mathbf{b}\|_2^2, \tag{2.3}$$

  which returns an approximate solution to $\mathbf{Ax} = \mathbf{b}$ that has a small number of nonzero entries. The fidelity and structure of the solution is controlled by parameter λ.

- the basis pursuit (BP) problem [13]

$$\min_{\mathbf{x}} \ \|\mathbf{x}\|_1 + \ell_{\{\mathbf{x}:\mathbf{A}\mathbf{x}=\mathbf{b}\}}(\mathbf{x}), \tag{2.4}$$

  where $\ell_{\mathbb{S}}(\mathbf{x})$ is the indicate function, i.e., if $\mathbf{x} \in \mathbb{S}$, then $\ell_{\mathbb{S}}(\mathbf{x}) = 0$, otherwise, $\ell_{\mathbb{S}}(\mathbf{x}) = \infty$. Under certain conditions (such as RIP), solving BP gives the sparsest solution of the under determined system $\mathbf{A}\mathbf{x} = \mathbf{b}$;

- the sparse logistic regression [34]

$$\min_{\mathbf{x},c} \ \lambda \cdot \|\mathbf{x}\|_1 + \frac{1}{m}\sum_{i=1}^{m} \log\left(1 + \exp(-b_i(\mathbf{a}_i^T\mathbf{x} + c))\right), \tag{2.5}$$

  which is an effective method for selecting a few features in classification problems;

- the group LASSO problem [46]

$$\min_{\mathbf{x}} \ \lambda \cdot \|\mathbf{x}\|_{1,2} + \frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2, \tag{2.6}$$

  which acts like the LASSO model at the group level: an entire subvector of $\mathbf{x}$ will be zero in the solution.

Some other examples include fused LASSO [40], sparse SVM [4], et al.

Generally speaking, for large-scale problems as listed above, there are two main challenges.

- **Nonsmoothness.** The regularizer $\mathcal{R}(\mathbf{x})$ is usually nonsmooth, which means that classical optimization techniques such as gradient based methods and Newton methods, quasi-Newton methods can not be directly applied to (2.2);

- **Large dataset.** The loss function $\mathcal{L}(\mathbf{Ax}, \mathbf{b})$ involves m data points with dimension n. For large scale problems, m and n are within the scale of $10^5 - 10^7$. This amount of data requires hundreds of gigabytes to terabytes of memory.

The goal of my research is to develop effective and efficient algorithms to overcome the above two challenges. Some general discussion are presented in section 2.3.1 and section 2.3.2.

2.3.1 Non-smoothness

One common way to handle non-smoothness in sparse optimization is the proximal operator [14], which is defined by

$$\mathbf{prox}_{\lambda\mathcal{R}}(\mathbf{t}) = \arg\min_{\mathbf{x}} \lambda \cdot \mathcal{R}(\mathbf{x}) + \frac{1}{2}\|\mathbf{x} - \mathbf{t}\|_2^2. \tag{2.7}$$

For instance, the proximal operator for the ℓ_1 norm is given by

$$\text{shrink}(t, \lambda) = \begin{cases} t - \lambda, & t > \lambda \\ 0, & -\lambda \le t \le \lambda \\ t + \lambda, & t < -\lambda. \end{cases} \tag{2.8}$$

If $\mathcal{R}(\mathbf{x})$ is the *elastic net regularizer*, i.e., $\mathcal{R}(\mathbf{x}) = \|\mathbf{x}\|_1 + \frac{1}{2\alpha}\|\mathbf{x}\|_2^2$, then

$$\mathbf{prox}_{\lambda\mathcal{R}}(\mathbf{x}) = \frac{1}{1 + \lambda\alpha}\,\text{shrink}(\mathbf{x}, \lambda). \tag{2.9}$$

If $\mathcal{R}(\mathbf{x})$ is the Huber norm, then its proximal operator is

$$\mathbf{prox}_{Huber}(t) = \begin{cases} t - \lambda, & t > \lambda + \mu \\ \dfrac{\mu}{\lambda + \mu} \cdot t, & |t| \leq \lambda + \mu \\ t + \lambda, & t < -\lambda - \mu. \end{cases} \tag{2.10}$$

Examples of the different proximal operators are shown in Figure 2.2. Some other proximal operators can be found in [14].

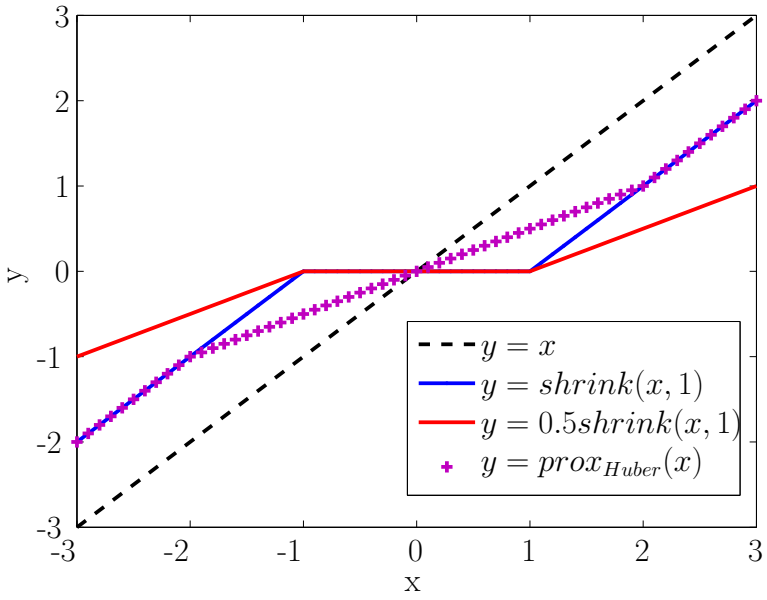


Figure 2.2 : Proximal operators. $\text{shrink}(x, 1)$ is the proximal operator for ℓ_1 norm with $\lambda = 1$; $0.5\,\text{shrink}(x, 1)$ gives the proximal operator for elastic net function with $\lambda = \alpha = 1$; proximal operator for Huber norm with $\lambda = \mu = 1$ is given by $\mathbf{prox}_{Huber}(x)$.

2.3.2 Large Dataset

As discussed previously, the fidelity term $\mathcal{L}(\mathbf{Ax}, \mathbf{b})$ in (2.1) is the summation of m loss functions $\ell(\mathbf{a}_j^T\mathbf{x}, b_j)$, which involve data samples $\mathbf{a}_j \in \mathbb{R}^n$ and the observation $b_j \in \mathbb{R}$.

Let $\mathbf{A}$ be a $m \times n$ matrix and $\mathbf{a}_i^T$ be the ith row of matrix $\mathbf{A}$, i.e., $\mathbf{A} = [\mathbf{a}_1^T; \mathbf{a}_2^T; \cdots; \mathbf{a}_m^T]$. Here m and n are in the scale of $10^5 - 10^7$. Table 2.1 gives the quantification of different scale of problems[1]. This thesis focuses on large size problems. Problems in

Table 2.1 : problem scale quantification

Class	Operations	Dimension	Iter. cost	Memory (bytes)
Small size	All	$10^0 - 10^2$	$n^4 - n^3$	Kilobyte: 10^3
Medium size	$\mathbf{A}^{-1}$	$10^3 - 10^3$	$n^3 - n^2$	Megabyte: 10^6
Large size	$\mathbf{Ax}$	$10^5 - 10^7$	$n^2 - n$	Gigabyte: 10^9
Huge size	$\mathbf{x} + \mathbf{y}$	$10^8 - 10^{12}$	$n - \log n$	Terabyte: 10^{12}

this scale should avoid operations with complexity higher than $O(n^2)$. For instance, matrix vector multiplication which has complexity $O(n^2)$ can be allowed, while matrix and matrix multiplication should be avoided.

In addition, problems in this size cannot be solved by a single processor workstation in general such that we need to make use of the parallel computing to split the computational tasks to different processors in parallel. For instance, matrix $\mathbf{A}$ must be stored in a distributed manner when it is too large to store centrally, or when it is collected in a distributed manner. Hence, solving (2.1) will involve computing $\mathbf{Ax}$ and $\mathbf{A}^T\mathbf{y}$ (multiple times with different $\mathbf{x}$ and $\mathbf{y}$) in a distributed manner. We discuss three scenarios (shown in Figure 2.3):

[1] http://www.maths.ed.ac.uk/prichtar

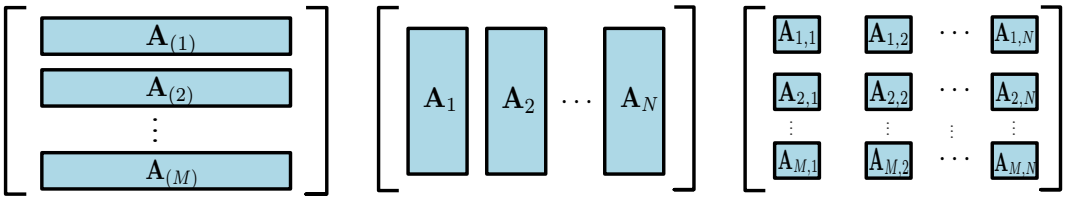


Figure 2.3 : Data distribution scenarios. Left: row block partition; middle: column block partition; right: general block partition

(a) row block distribution: $\mathbf{A}$ is partitioned into row-blocks $\mathbf{A}_{(1)}, \mathbf{A}_{(2)}, \ldots, \mathbf{A}_{(M)}$. Stacking them forms $\mathbf{A}$;

(b) column block distribution: $\mathbf{A} = [\mathbf{A}_1 \ \mathbf{A}_2 \ \cdots \ \mathbf{A}_N]$;

(c) general block distribution: $\mathbf{A}$ is partitioned into MN sub-blocks. The (i,j)th block is $\mathbf{A}_{i,j}$.

Computing

$$\mathbf{A}\mathbf{x} = \begin{bmatrix} \mathbf{A}_{(1)}\mathbf{x} \\ \mathbf{A}_{(2)}\mathbf{x} \\ \cdots \\ \mathbf{A}_{(M)}\mathbf{x} \end{bmatrix} \text{ and } \mathbf{A}^T\mathbf{y} = \begin{bmatrix} \mathbf{A}_1^T\mathbf{y} \\ \mathbf{A}_2^T\mathbf{y} \\ \cdots \\ \mathbf{A}_N^T\mathbf{y} \end{bmatrix},$$

in scenarios (a) and (b), respectively, require $\mathbf{x}$ and $\mathbf{y}$ to be *broadcasted* to (or synchronized across) all the nodes. Then, independent of one another, every node computes $\mathbf{A}_{(i)}\mathbf{x}$ and $\mathbf{A}_j^T\mathbf{y}$.

On the other hand, computing

$$\mathbf{A}^T\mathbf{y} = \sum_{i=1}^{M} \mathbf{A}_{(i)}^T\mathbf{y}_i \text{ and } \mathbf{A}\mathbf{x} = \sum_{i=1}^{N} \mathbf{A}_j\mathbf{x}_j,$$

in scenarios (a) and (b), respectively, take two steps each: assuming every node keeps its corresponding $\mathbf{x}_i$ and $\mathbf{y}_j$, each $\mathbf{A}_{(i)}^T\mathbf{y}_i$ and $\mathbf{A}_j\mathbf{x}_j$ are computed independently in parallel; then the summation is put together through a *reduce* operation across all the nodes.

In summary, in scenario (a), computing $\mathbf{Ax}$ is divided into the parallel jobs of computing $\mathbf{A}_{(i)}\mathbf{x}$, which are preceded by *broadcasting* $\mathbf{x}$ whereas computing $\mathbf{A}^T\mathbf{y}$ is done by parallelly computing $\mathbf{A}_{(i)}^T\mathbf{y}_i$. followed by a *reduce* operation. Scenario (b) has the exact opposite.

In scenario (c), computing either $\mathbf{Ax}$ or $\mathbf{A}^T\mathbf{y}$ requires a mixed use of both broadcasting and reduce operations. For example, it takes three steps to compute

$$\mathbf{Ax} = \begin{bmatrix} \mathbf{A}_{(1)}\mathbf{x} \\ \mathbf{A}_{(2)}\mathbf{x} \\ \cdots \\ \mathbf{A}_{(M)}\mathbf{x} \end{bmatrix} = \sum_{j=1}^{N} \begin{bmatrix} \mathbf{A}_{1,j}\mathbf{x}_j \\ \mathbf{A}_{2,j}\mathbf{x}_j \\ \cdots \\ \mathbf{A}_{M,j}\mathbf{x}_j \end{bmatrix}.$$

In step 1, for every $j = 1, 2, \ldots, N$, $\mathbf{x}_j$ is *broadcasted* to nodes $(1, j), \ldots, (M, j)$ in parallel; in step 2, $\mathbf{A}_{i,j}\mathbf{x}_j$ for all i, j are computed in parallel; finally, for every $i = 1, 2, \ldots, M$ in parallel, a *reduce* operation is applied to nodes $(i, 1), (i, 2), \ldots, (i, N)$ to return $\mathbf{A}_{(i)}\mathbf{x} = \sum_{j=1}^{N} \mathbf{A}_{i,j}\mathbf{x}_j$ to all these nodes for further use. A similar process can compute $\mathbf{A}^T\mathbf{y}$.

In all scenarios, the blocks of $\mathbf{A}$ can have different sizes (although, in scenario (c), the blocks should be aligned; otherwise, extra broadcasting and reduce operations will

be incurred). In fact, assigning larger blocks to faster (or less busy) nodes will improve the overall efficiency. In addition, matrix vector multiplication for all scenarios does not need to send and receive the data matrix **A** across nodes.

Chapter 3

Parallel Algorithms

We take the following three approaches to develop scalable algorithms for sparse optimization, where the first two approaches are novel and the third one is based on Boyd et al.'s work [6], but we extend it to the dual problem.

- **Parallel prox-linear algorithms.** Prox-linear algorithms linearize $L(\mathbf{x})$ at $\mathbf{x}^k$ and then proximal operator is applied. The separable structure of the terms in the objective enables us to make the existing algorithms scalable by parallelize some of the operations. Details are provided in section 3.1.
- **Parallel greedy coordinate descent method (GRock).** GRock is developed based on greedy block-coordinate update. At each iteration, it selects a few blocks of variables and then a few variables in each selected block, both by greedy means, and update the latter in parallel. We include the proof for the sub-linear convergence rate of GRock. See section 3.2 for details.
- **Parallel alternating direction method of multipliers.** In general, there are two ways to apply parallel ADMM to problem (2.1): 1. parallelize the original ADMM by parallelizing operations; 2. introduce new variables and reformulate the original problem into many smaller problems which can be solved in parallel. Detailed discussions are presented in section 3.3.

3.1 Parallel Prox-linear Algorithms

When $\mathcal{R}(\mathbf{x})$ is block separable and $\mathcal{L}(\mathbf{Ax}, \mathbf{b})$ is the summation of loss functions, it is straightforward to develop distributed implementations of prox-linear algorithms such as ISTA, FPC and FISTA. In short, prox-linear algorithms are the iterations of gradient descent and proximal operations. The former requires $\nabla_x \mathcal{L}(\mathbf{Ax}^k, \mathbf{b}) = \mathbf{A}^T \nabla \mathcal{L}(\mathbf{Ax}^k, \mathbf{b})$, which can be obtained based on the distributed computing of $\mathbf{Ax}$ and $\mathbf{A}^T\mathbf{y}$ together with the separability of $\mathcal{L}(\mathbf{Ax}, \mathbf{b})$. The latter is often straightforward and can take advantage of the separability of $\mathcal{R}(\mathbf{x})$.

Specifically, the basic prox-linear iteration applied to (2.1) is

$$\mathbf{x}^{k+1} \leftarrow \arg\min_{\mathbf{x}} \lambda \cdot \mathcal{R}(\mathbf{x}) + \langle \mathbf{x} - \mathbf{x}^k, \mathbf{A}^T \nabla \mathcal{L}(\mathbf{Ax}^k, \mathbf{b}) \rangle + \frac{1}{2\delta_k} \|\mathbf{x} - \mathbf{x}^k\|_2^2, \tag{3.1}$$

where $\mathcal{L}(\mathbf{Ax}, \mathbf{b})$ is linearized at $\mathbf{x}^k$. The last term keeps $\mathbf{x}^{k+1}$ close to $\mathbf{x}^k$ where the distance is controled by stepsize δ_k. Combining the second term and the last term, we can obtain the solution of (3.1) as follows

$$\mathbf{x}^{k+1} = \mathbf{prox}_{\lambda\mathcal{R}} \left(\mathbf{x}^k - \delta_k \mathbf{A}^T \nabla \mathcal{L}(\mathbf{Ax}^k, \mathbf{b}) \right), \tag{3.2}$$

where $\mathbf{prox}_{\lambda\mathcal{R}}(\mathbf{x})$ is the proximal operator defined in (2.7).

To parallel the prox-linear methods, we consider the three data partition schemes as discussed in section 2.3.2 respectively.

In scenario (a), every node i keeps $\mathbf{A}_{(i)}$, $\mathbf{b}_i$ and current $\mathbf{x}^k$. It computes $\mathbf{A}_{(i)}\mathbf{x}^k$ and then $\nabla \mathcal{L}_i(\mathbf{A}_{(i)}\mathbf{x}^k, \mathbf{b}_i)$. Once such computation is finished on all nodes, a *reduce* operation computes $\mathbf{A}^T \nabla \mathcal{L}(\mathbf{Ax}^k, \mathbf{b}) = \sum_{i=1}^{M} \mathbf{A}_{(i)}^T \nabla \mathcal{L}_i(\mathbf{A}_{(i)}\mathbf{x}^k; \mathbf{b}_i)$ and returns the sum to

every node. Every node then independently finishes (3.2), and all obtain the identical $\mathbf{x}^{k+1}$.

In scenario (b), every node j keeps $\mathbf{A}_j$, $\mathbf{b}$ and the jth block of $\mathbf{x}^k$. Every node computes $\mathbf{A}_j\mathbf{x}_j^k$ and awaits the *reduce* operation to return $\mathbf{A}\mathbf{x}^k = \sum_{j=1}^{N} \mathbf{A}_j\mathbf{x}_j^k$. Then, every node j independently computes $\mathbf{A}_j^T\nabla\mathcal{L}(\mathbf{A}\mathbf{x}^k, \mathbf{b})$ and further, based on the separability of $\mathcal{R}$, obtains $\mathbf{x}_j^{k+1} = \mathbf{prox}_{\lambda\mathcal{R}_j}\left(\mathbf{x}_j^k - \delta_k\mathbf{A}_j^T\nabla\mathcal{L}(\mathbf{A}\mathbf{x}^k, \mathbf{b})\right)$.

In scenario (c), every node (i, j) keeps $\mathbf{A}_{i,j}$, $\mathbf{b}_i$ and the jth block of $\mathbf{x}^k$. Then the related matrix vector computation can be efficient implemented by the parallel schemes described in section 2.3.2. Notice that scenarios (a) and (b) require the separability of $\mathcal{L}(\mathbf{A}\mathbf{x}, \mathbf{b})$ and $\mathcal{R}(\mathbf{x})$ respectively, but scenario (c) needs both functions to be separable.

Step size δ_k plays a key role in the performance of prox-linear methods and its estimation is not trivial. If δ_k is fixed, it is often set inversely proportional to the Lipschitz constant of $\nabla_x\mathcal{L}(\mathbf{A}\mathbf{x}, \mathbf{b})$, which can be estimated by a distributed implementation of the power method based on operations $\mathbf{A}\mathbf{x}$ and $\mathbf{A}^T\mathbf{y}$. Some algorithms determine δ_k by the Barzilai-Borwein method followed by back-tracking line search, both of which require extra computation but nonetheless boil down to inner products involving $\nabla\mathcal{L}(\mathbf{A}\mathbf{x}^k, \mathbf{b})$.

In section 3.1.1, we give an example of applying parallel prox-linear algorithm to LASSO according to scenario (b); section 3.1.2 gives the parallel algorithm for solving sparse logistic regression according to scenario (a).

3.1.1 Example: distributed LASSO

The LASSO model is

$$\min_{\mathbf{x}} \lambda\|\mathbf{x}\|_1 + \frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2.$$

Linearize the second term at $\mathbf{x}^k$ gives

$$\mathbf{x}^{k+1} = \arg\min_{\mathbf{x}} \ \lambda\|\mathbf{x}\|_1 + \frac{1}{2\delta_k}\|\mathbf{x} - (\mathbf{x}_k - \delta_k\mathbf{A}^T(\mathbf{A}\mathbf{x}^k - \mathbf{b}))\|_2^2$$

whose solution is given by

$$\mathbf{x}^{k+1} = \mathbf{prox}_{\lambda\|\cdot\|_1}\left(\mathbf{x}^k - \delta_k\mathbf{A}^T(\mathbf{A}\mathbf{x}^k - \mathbf{b})\right).$$

The ℓ_1 proximal operator $\mathbf{prox}_{\lambda\|\cdot\|_1}$ is known as shrinkage or soft-thresholding, which is component-wise separable and can be computed in closed form as $\text{shrink}(x_j, \lambda) = \text{sign}(x_j)\max(|x_j| - \lambda, 0)$.

Algorithm 1 presents our distributed implementation for scenario (b), which is based on FISTA [2], a Nesterov-type acceleration of (3.2), and the SPMD (single program, multiple data) technique.

When Algorithm 1 runs on distributed nodes, steps 3, 5, 6 and 7 are executed independently on every node, and the *MPI_Allreduce* operation in step 4 forms the sum and sends it to all the nodes. Note that we require $\delta_k < \dfrac{1}{\|\mathbf{A}\|_2^2}$ to ensure convergence. For fixed stepsize, we need to evaluate $\|\mathbf{A}\|_2^2$, which equals to the maximum eigenvalue of $\mathbf{A}^T\mathbf{A}$. Notice that power method can be applied even though matrix $\mathbf{A}$ is stored in a distributed manner, because the power iteration only involves repeated matrix and

Algorithm 1 P-FISTA: parallel LASSO

node j keeps $\mathbf{A}_j$, $\mathbf{b}$, initializes $\mathbf{x}_j^0 = \mathbf{x}_j^1 = 0$;

for $k = 1, 2, \ldots, K$ **do**

  $\bar{\mathbf{x}}_j \leftarrow \mathbf{x}_j^k + \frac{k-2}{k+1}\left(\mathbf{x}_j^k - \mathbf{x}_j^{k-1}\right)$;

  $\mathbf{w} \leftarrow \sum_{j=1}^{N} \mathbf{A}_j \bar{\mathbf{x}}_j$ by MPI_Allreduce;

  $\mathbf{y} \leftarrow \nabla \mathcal{L}(\mathbf{w}; \mathbf{b})$;

  $\mathbf{g}_j \leftarrow \mathbf{A}_j^T \mathbf{y}$;

  $\mathbf{x}_j^{k+1} \leftarrow \mathbf{prox}_{\lambda\|\cdot\|_1}(\bar{\mathbf{x}}_j - \delta_k \mathbf{g}_j)$;

end for

vector multiplication. Given an initial vector $\mathbf{x}^0$, the eigenvector corresponding to the largest eigenvalue of $\mathbf{A}^T\mathbf{A}$ is estimated by the following iteration

$$\mathbf{x}^{k+1} = \frac{\mathbf{A}^T\mathbf{A}\mathbf{x}^k}{\|\mathbf{A}^T\mathbf{A}\mathbf{x}^k\|_2}. \tag{3.3}$$

In every iteration, $\mathbf{x}^k$ is multiplied by matrix $\mathbf{A}^T\mathbf{A}$ and normalized. Assume $\mathbf{A}$ is partitioned according to senario (b), then

$$\begin{bmatrix} \mathbf{x}_1^{k+1} \\ \mathbf{x}_2^{k+1} \\ \vdots \\ \mathbf{x}_B^{k+1} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_1^T\mathbf{A}\mathbf{x}^k \\ \mathbf{A}_2^T\mathbf{A}\mathbf{x}^k \\ \vdots \\ \mathbf{A}_B^T\mathbf{A}\mathbf{x}^k \end{bmatrix} / \|\mathbf{A}^T\mathbf{A}\mathbf{x}^k\|_2. \tag{3.4}$$

Notice that most of the computation efforts can be done locally, the only communication needed is to calculate $\mathbf{A}\mathbf{x}^k = \sum_{i=1}^{B} \mathbf{A}_i \mathbf{x}_i^k$. When the power iteration reaches the stopping

criterion, the eigenvalue can by computed by $\lambda_{\max} = \frac{\|\mathbf{A}^T\mathbf{A}\mathbf{x}^k\|_2^2}{\|\mathbf{A}\mathbf{x}^k\|_2^2}$.

3.1.2 Example: distributed sparse logistic regression

Sparse logistic regression solves

$$\min_{\mathbf{w},c} \ \lambda\|\mathbf{w}\|_1 + \frac{1}{m}\sum_{i=1}^{m}\log\left(1+\exp(-b_i(\mathbf{w}^T\mathbf{a}_i + c))\right). \tag{3.5}$$

For given data $\mathbf{a}_i$ and b_i, form matrix

$$[\mathbf{C}\ \mathbf{b}] = \begin{bmatrix} b_1\mathbf{a}_1^T & b_1 \\ b_2\mathbf{a}_2^T & b_2 \\ \cdots & \\ b_m\mathbf{a}_m^T & b_m \end{bmatrix} = \begin{bmatrix} \mathbf{C}_{(1)} & \mathbf{b}_{(1)} \\ \mathbf{C}_{(2)} & \mathbf{b}_{(2)} \\ \cdots & \\ \mathbf{C}_{(M)} & \mathbf{b}_{(M)} \end{bmatrix},$$

where each $\mathbf{C}_{(i)}$ and $\mathbf{b}_{(i)}$ are blocks of rows of $\mathbf{C}$ and $\mathbf{b}$, respectively, and $\mathcal{L}(\mathbf{t}) = \frac{1}{m}\sum_{i}^{m}\log(1+\exp(-t_i))$. Then, (3.5) can be rewritten as

$$\min_{\mathbf{w},c} \ \lambda\|\mathbf{w}\|_1 + \mathcal{L}(\mathbf{C}\mathbf{w} + c\cdot\mathbf{b}), \tag{3.6}$$

whose prox-linear iteration is

$$\mathbf{w}^{k+1} = \mathbf{prox}_{\lambda\|\cdot\|_1}(\mathbf{w}^k - \delta_k\mathbf{C}^T\nabla\mathcal{L}(\mathbf{C}\mathbf{w}^k + \mathbf{b}c^k))$$

$$c^{k+1} = c^k - \delta_k\mathbf{b}^T\nabla\mathcal{L}(\mathbf{C}\mathbf{w}^k + \mathbf{b}c^k).$$

Algorithm 2 describes our implementation for scenario (a). Steps 3–5, 8, and 9 run independently on every node, and steps 6 and 7 call MPI_Allreduce to form the sums and send them to all nodes.

Algorithm 2 PSLR: parallel sparse logistic regression

node i keeps $\mathbf{C}_{(i)}$, $\mathbf{b}_{(i)}$, sets $\mathbf{w}^0 = \mathbf{w}^1 = 0$, $c^0 = c^1 = 0$;

for $k = 1, 2, \ldots, K$ **do**

$\bar{\mathbf{w}} \leftarrow \mathbf{w}^k + \frac{k-2}{k+1}\left(\mathbf{w}^k - \mathbf{w}^{k-1}\right)$;

$\bar{c} \leftarrow c^k + \frac{k-2}{k+1}\left(c^k - c^{k-1}\right)$;

$\mathbf{y}_{(i)} \leftarrow \nabla \mathcal{L}_i(\mathbf{C}_{(i)}\bar{\mathbf{w}} + \mathbf{b}_{(i)}\bar{c})$;

$\mathbf{g}_{\mathbf{w}} \leftarrow \sum_{i=1}^{M} \mathbf{C}_{(i)}^T \mathbf{y}_{(i)}$ by MPI_Allreduce;

$g_c \leftarrow \sum_{i=1}^{M} \mathbf{b}_{(i)}^T \mathbf{y}_{(i)}$ by MPI_Allreduce;

$\mathbf{w}^{k+1} \leftarrow \mathbf{prox}_{\lambda\|\cdot\|_1}(\bar{\mathbf{w}} - \delta_k \mathbf{g}_{\mathbf{w}})$;

$c^{k+1} \leftarrow \bar{c} - \delta_k g_c$.

end for

3.1.3 Convergence analysis

The convergence of the sequence given by (3.2) was proved in [15]. It is based on the assumption that $\mathcal{R}(\mathbf{x})$ and $\mathcal{L}(\mathbf{Ax}, \mathbf{b})$ are convex, and the gradient of $\mathcal{L}(\mathbf{Ax}, \mathbf{b})$ is Lipschitz continuous. In general, linear convergence rate of (3.2) is not available, but it can be achieved by adding some additional assumptions [1]. Nestrov [32], Beck and Teboulle [2] established the convergence rate of $O(\frac{1}{k^2})$ with respect to the objective function.

3.2 Parallel Greedy Coordinate Descent Methods

The coordinate descent method (CD) is one of the first schemes applied to nonlinear optimization. At each iteration of this method, the objective $\mathcal{F}(\mathbf{x})$ is minimized with respect to a chosen coordinate (or block of coordinates) while the others are fixed. Its main advantage is the simplicity of each update.

It is important to decide which coordinate(s) to update at each iteration. The most common rule cycles through all of the coordinates in a Gauss-Seidel fashion; we call it *Cyclic CD*. In *Random CD*, the coordinate(s) are selected at random, and its analysis is based on the expectation of objective value [33]. *Greedy CD*, on the other hand, chooses the coordinate(s) with the best merit value(s), for example, a coordinate of the gradient vector with the maximal absolute value. There are also rules that mix the above three, which we call *Mixed CD*. For example, Scherrer et al. [36] partitioned the coordinates into B blocks and randomly selected P blocks. Within each block, a coordinate is selected for update in a greedy manner.

3.2.1 Algorithm

We argue that sparse optimization favors Greedy CD than other rules. In fact, under some orthogonality conditions (such as the RIP and incoherence conditions), certain greedy rules are guaranteed to select the coordinates corresponding to those nonzero entries in the final solution. Below we present GRock, a greedy coordinate-block descent method for solving (2.1) which is friendly with parallel computing. To this

end, we assume $\mathcal{R}(\mathbf{x})$ is separable, $\mathcal{L}(\mathbf{Ax}, \mathbf{b})$ is convex and $\mathbf{A}$ has columns with unit 2-norm (to simplify our analysis). We assume

$$\mathcal{L}(\mathbf{A}(\mathbf{x}+\mathbf{d}), \mathbf{b}) \leq \mathcal{L}(\mathbf{Ax}, \mathbf{b}) + \mathbf{g}^T\mathbf{d} + \frac{\beta}{2}\mathbf{d}^T\mathbf{A}^T\mathbf{Ad}, \tag{3.7}$$

where $\beta > 0$ and $\mathbf{g} = \mathbf{A}^T\nabla\mathcal{L}(\mathbf{Ax}, \mathbf{b})$ is the gradient of $\mathcal{L}(\mathbf{Ax}, \mathbf{b})$. Notice that $\beta = 1$ for the square loss function and $\beta = \frac{1}{4}$ for the logistic loss. We define the potential decrease of each coordinate i by

$$d_i = \arg\min_d \ \lambda \cdot r(x_i + d) + g_i d + \frac{\beta}{2}d^2, \tag{3.8}$$

which is given in closed form as $d_i = \mathbf{prox}_{\frac{\lambda}{\beta}r}(x_i - \frac{1}{\beta}g_i) - x_i$, where g_i is the ith entry of $\mathbf{g}$.

As we will later consider scenario (b) and use multiple computing nodes, we divide the coordinates into N blocks. For each block j, let

$$m_j = \max\{|d| : d \text{ is an element of } \mathbf{d}_j\}, \tag{3.9}$$

and s_j be such that $m_j = d_{s_j}$, i.e., coordinate s_j achieves the maximum. At each iteration, out of the N blocks, the P blocks with largest m_j are selected, and their maximizing entries s_j are updated. In short, the best coordinate of each of the best P blocks are updated.

Let $\mathcal{P}$ be the set that contains the P selected entries, then the algorithm GRock is summarized in Algorithm 3.

Algorithm 3 GRock for scenario (b)

1: Initialize $\mathbf{x} = 0 \in \mathbb{R}^n$

2: **while** not converged **do**

3: $\mathbf{d}_j \leftarrow$ (3.8) for each block j;

4: $m_j, s_j \leftarrow$ (3.9) for each block j;

5: $\mathcal{P} \leftarrow$ the indices of the P blocks with largest m_j;

6: $x_{s_j} \leftarrow x_{s_j} + d_{s_j}$, for each $j \in \mathcal{P}$.

7: **end while**

3.2.2 GRock demonstration

In this section, we give two examples to show the strength and weakness of GRock.

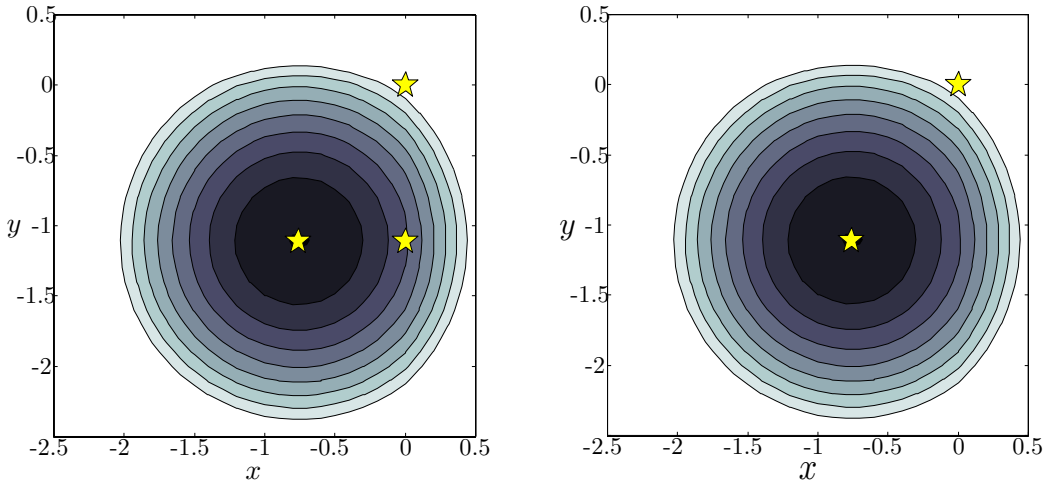
The first example is

$$\min \frac{1}{10}\|\mathbf{x}\|_1 + \frac{1}{2}\|\mathbf{x} - \mathbf{b}\|_2^2, \tag{3.10}$$

where $\mathbf{x} \in \mathbb{R}^2$, and $\mathbf{b}$ is a two dimensional vector. In Figure 3.1(a), GRock with $P = 1$ requires 2 iterations to converge to the optimal solution; but Figure 3.1(b) demonstrates that GRock with $P = 2$ converges to the optimal solution with only one step.

The second example is a three dimensional LASSO problem, where

$$\mathbf{A} = \begin{bmatrix} 0.9234 & -0.2004 & 1.8446 \\ 0.3273 & 0.7827 & 0.2291 \\ 0.2006 & -0.5892 & 3.2037 \end{bmatrix},$$

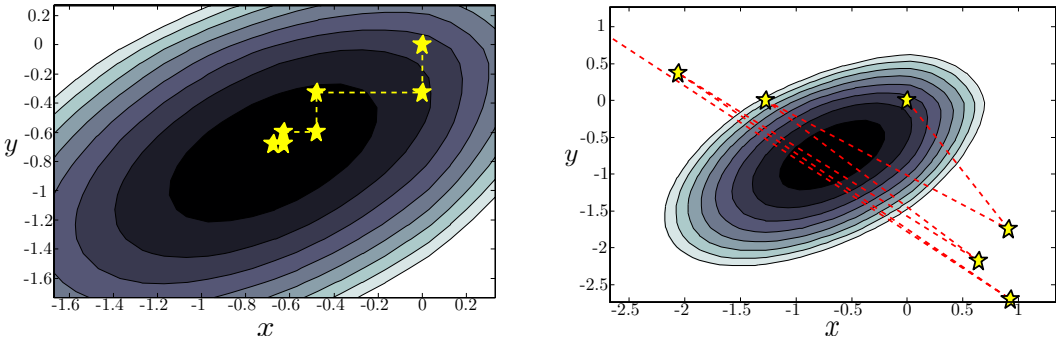


(a) $P = 1$ (b) $P = 2$

Figure 3.1 : Example I. GRock with $P = 1$ needs two steps to converges to the optimal solution; GRock with $P = 2$ converges to the optimal solution with only one step.

$\lambda = \frac{1}{10}$ and $\mathbf{b}$ is a random vector. To visualize the iterative process, we project the contour of the objective function to a two dimensional space. Figure 3.2(b) demonstrates $P = 3$, in which case the objective increases and the points diverge over the iterations. Due to the slanted shape of the contour, if three components are updated together, the updates must work together in order to decrease the objective, whereas each update (3.8) ignores the others and does not have the big picture.

In conclusion, on one hand, if the contour of the objective function has the circle shape, updating larger number of coordinates leads to faster convergence; on the other hand, if the contour of the objective function has a slant shape, we should choose fewer coordinates to update. Based on this intuition, convergence analysis is provided in the next section.



(a) $P = 1$ (b) $P = 3$

Figure 3.2 : Example II. GRock with $P = 1$ converges to the optimal solution; GRock with $P = 3$ diverges.

3.2.3 Convergence analysis

To ensure convergence, similar to the analysis in [36], we define a block spectral radius as follows,

Definition 2 (Block Spectral Radius)

The block spectral radius is defined by

$$\rho_P = \max_{\mathbf{M}\in\mathcal{M}} \rho(\mathbf{M}), \tag{3.11}$$

where $\mathcal{M}$ is the set of all $P \times P$ submatrices with exactly one column selected from each of the P blocks of $\mathbf{A}$.

Notice that ρ_P is monotonically increasing with respect to P, thus $1 \leq \rho_P \leq \rho_n$ where ρ_n is the largest eigenvalue of $\mathbf{A}^T\mathbf{A}$. The intuition for ρ_P is that if the columns of $\mathbf{A}$ from different blocks are nearly orthogonal to one another then $\mathbf{M} \in \mathcal{M}$ will be close to identity matrix and hence ρ_P is small. In fact, Lemma 1 shows that $\rho_P < 2$

guarantees monotonically decreasing of the objective function.

Lemma 1

Assume that $\mathcal{R}(\mathbf{x})$ *is convex,* $\mathcal{L}(\mathbf{Ax}, \mathbf{b})$ *satisfies assumption (3.7), and* $\mathbf{x}^k$ *be the sequence generated by Algorithm 3. If* $\rho_P < 2$*, then*

$$\mathcal{F}(\mathbf{x}^{k+1}) - \mathcal{F}(\mathbf{x}^k) \leq \frac{\rho_P - 2}{2}\beta\|\mathbf{x}^{k+1} - \mathbf{x}^k\|_2^2.$$

Proof. For simplicity we use $\mathcal{L}(\mathbf{x})$ to represent $\mathcal{L}(\mathbf{Ax}, \mathbf{b})$, then assumption (3.7) is equivalent to

$$\mathcal{L}(\mathbf{x} + \mathbf{d}) \leq \mathcal{L}(\mathbf{x}) + \nabla\mathcal{L}(\mathbf{x})^T\mathbf{d} + \frac{\beta}{2}\mathbf{d}^T\mathbf{A}^T\mathbf{Ad}. \tag{3.12}$$

Let $\mathbf{g} = \nabla\mathcal{L}(\mathbf{x}^k)$ and $\bar{\mathbf{d}} = \mathbf{x}^{k+1} - \mathbf{x}^k$. We have

$$\begin{aligned}
&\mathcal{F}(\mathbf{x}^{k+1}) - \mathcal{F}(\mathbf{x}^k) \\
&= \mathcal{L}(\mathbf{x}^{k+1}) + \lambda\mathcal{R}(\mathbf{x}^{k+1}) - \mathcal{L}(\mathbf{x}^k) - \lambda\mathcal{R}(\mathbf{x}^k) \\
&\leq \mathbf{g}^T\bar{\mathbf{d}} + \frac{\beta}{2}\bar{\mathbf{d}}^T\mathbf{A}^T\mathbf{A}\bar{\mathbf{d}} + \lambda(\mathcal{R}(\mathbf{x}^{k+1}) - \mathcal{R}(\mathbf{x}^k)) \\
&\leq \mathbf{g}^T\bar{\mathbf{d}} + \frac{\beta}{2}\bar{\mathbf{d}}^T\mathbf{A}^T\mathbf{A}\bar{\mathbf{d}} + \lambda\partial\mathcal{R}(\mathbf{x}^{k+1})^T\bar{\mathbf{d}}.
\end{aligned}$$

Define J as the index set containing the P selected indexes $\{j_1, j_2, ..., j_P\}$ from P blocks, then the optimality condition gives

$$\beta\bar{\mathbf{d}}_J + \mathbf{g}_J + \lambda\partial R_J(\mathbf{x}^{k+1}) = 0.$$

Correspondingly, define the $P \times P$ submatrix $\mathbf{M}$ with entries $\mathbf{M}_{s,t} = \mathbf{A}_{j_s}^T \mathbf{A}_{j_t}$, then

$$\begin{aligned}
&\mathcal{F}(\mathbf{x}^{k+1}) - \mathcal{F}(\mathbf{x}^k) \\
\leq\ & \mathbf{g}_J^T \bar{\mathbf{d}}_J + \frac{\beta}{2}(\bar{\mathbf{d}}_J)^T \mathbf{M} \bar{\mathbf{d}}_J + \lambda \partial \mathcal{R}_J(\mathbf{x}^{k+1})^T \bar{\mathbf{d}}_J \\
\leq\ & \mathbf{g}_J^T \bar{\mathbf{d}}_J + \frac{\beta \rho_P}{2}(\bar{\mathbf{d}}_J)^T \bar{\mathbf{d}}_J + (-\mathbf{g}_J - \beta \bar{\mathbf{d}}_J)^T \bar{\mathbf{d}}_J \\
=\ & \frac{\rho_P - 2}{2} \beta (\bar{\mathbf{d}}_J)^T \bar{\mathbf{d}}_J \\
=\ & \frac{\rho_P - 2}{2} \beta \|\bar{\mathbf{d}}\|_2^2.
\end{aligned}$$

■

Notice that as long as any two columns from two different blocks are linearly independent, we automatically have $\rho_2 < 2$. Before proving the main convergence result, we give Lemma 2 as follows.

Lemma 2

Assume the gradient of $\mathcal{L}(\mathbf{x})$ is Lipschitz continuous with Lipschitz constant L, $\mathbf{x}^*$ is the optimal solution and $\|\mathbf{x}^k - \mathbf{x}^*\|_2 \leq C$, $\forall k$, then the iterates of Algorithm 3 satisfy

$$\mathcal{F}(\mathbf{x}^{k+1}) - \mathcal{F}(\mathbf{x}^*) \leq C \left(2L + \beta \sqrt{\frac{N}{P}} \right) \|\mathbf{x}^{k+1} - \mathbf{x}^k\|_2.$$

Proof. Let $\bar{\mathbf{d}} = \mathbf{x}^{k+1} - \mathbf{x}^k$ and $\mathbf{d}^k$ be the vector of potentials for all the coordinates defined in (3.8). Since $\mathcal{L}(\mathbf{x})$ is a differentiable convex function, on one hand, we have

$$\begin{aligned}
\mathcal{L}(\mathbf{x}^{k+1}) - \mathcal{L}(\mathbf{x}^k) &\leq \langle \mathbf{g}^{k+1}, \bar{\mathbf{d}} \rangle \\
&= \langle \mathbf{g}^{k+1} - \mathbf{g}^k, \bar{\mathbf{d}} \rangle + \langle \mathbf{g}^k, \bar{\mathbf{d}} \rangle \\
&\leq L \|\bar{\mathbf{d}}\|_2^2 + \langle \mathbf{g}^k, \mathbf{x}^{k+1} - \mathbf{x}^k \rangle,
\end{aligned} \tag{3.13}$$

where $\mathbf{g}^k$ and $\mathbf{g}^{k+1}$ are the gradients of $\mathcal{L}(\mathbf{x})$ at $\mathbf{x}^k$ and $\mathbf{x}^{k+1}$ respectively. The second inequality is derived by the Lipschitz continuity of $\nabla\mathcal{L}(\mathbf{x})$. On the other hand, $\mathcal{L}(\mathbf{x})$ being convex also gives the following inequality

$$\mathcal{L}(\mathbf{x}^k) - \mathcal{L}(\mathbf{x}^*) \ \leq \ \langle \mathbf{g}^k, \mathbf{x}^k - \mathbf{x}^* \rangle. \tag{3.14}$$

Combining (3.13) and (3.14) gives

$$\mathcal{L}(\mathbf{x}^{k+1}) - \mathcal{L}(\mathbf{x}^*) \ \leq \ L\|\bar{\mathbf{d}}\|_2^2 + \langle \mathbf{g}^k, \mathbf{x}^{k+1} - \mathbf{x}^* \rangle. \tag{3.15}$$

Based on the convexity property of $\mathcal{R}(\mathbf{x})$, we have

$$\begin{aligned}
& \mathcal{F}(\mathbf{x}^{k+1}) - \mathcal{F}(\mathbf{x}^*) \\
= \ & \mathcal{L}(\mathbf{x}^{k+1}) - \mathcal{L}(\mathbf{x}^*) + \lambda(\mathcal{R}(\mathbf{x}^{k+1}) - \mathcal{R}(\mathbf{x}^*)) \\
\leq \ & L\|\bar{\mathbf{d}}\|_2^2 + (\lambda\partial\mathcal{R}(\mathbf{x}^{k+1}) + \mathbf{g}^k)^T(\mathbf{x}^{k+1} - \mathbf{x}^*) \\
= \ & L\|\bar{\mathbf{d}}\|_2^2 - \beta(\mathbf{d}^k)^T(\mathbf{x}^{k+1} - \mathbf{x}^*) \\
\leq \ & L\|\bar{\mathbf{d}}\|_2^2 + \beta C\|\mathbf{d}^k\|_2 \\
\leq \ & 2CL\|\bar{\mathbf{d}}\|_2 + \beta C\sqrt{\frac{N}{P}}\|\bar{\mathbf{d}}\|_2 \\
= \ & C\left(2L + \beta\sqrt{\frac{N}{P}}\right)\|\bar{\mathbf{d}}\|_2
\end{aligned}$$

■

Based on Lemma 1 and Lemma 2, we have the following convergence result.

Theorem 1 (GRock Convergence)

Let $\mathbf{x}^*$ *be an optimal solution of (2.1), and* $\mathbf{x}^k$ *be the sequence generated by Algorithm 3. Assume* $\|\mathbf{x}^k - \mathbf{x}^*\|_2 \leq C$, $\mathcal{F}(\mathbf{x})$ *satisfies the assumptions in Lemma 1 and the gradient of* $\mathcal{L}(\mathbf{Ax}, \mathbf{b})$ *is Lipschitz continuous with Lipschitz constant* L. *Then*

$$\mathcal{F}(\mathbf{x}^k) - \mathcal{F}(\mathbf{x}^*) \;\leq\; \frac{2C^2\left(2L + \beta\sqrt{\frac{N}{P}}\right)^2}{(2-\rho_P)\beta} \cdot \frac{1}{k}.$$

Proof. Combine the result from Lemma 1 and Lemma 2, we have

$$\mathcal{F}(\mathbf{x}^{k+1}) - \mathcal{F}(\mathbf{x}^k) \;\leq\; \frac{(\rho_P - 2)\beta}{2C^2(2L + \beta\sqrt{\frac{N}{P}})^2}\left(\mathcal{F}(\mathbf{x}^k) - \mathcal{F}(\mathbf{x}^*)\right)^2. \tag{3.16}$$

Let $a_k = \mathcal{F}(\mathbf{x}^k) - \mathcal{F}(\mathbf{x}^*)$ and $c = \dfrac{(2-\rho_P)\beta}{2C^2\left(2L + \beta\sqrt{\frac{N}{P}}\right)^2}$, then (3.17) can be simplified as

$$a_{k+1} - a_k \leq -ca_k^2, \tag{3.17}$$

from which we can derive $0 \leq a_k \leq \dfrac{1}{c}$. In addition, we have

$$\begin{aligned}
\frac{1}{a_{k+1}} &\geq \frac{1}{a_k(1 - c \cdot a_k)} \\
&= \frac{1}{a_k} + \frac{c}{1 - c \cdot a_k} \\
&\geq \frac{1}{a_k} + c \\
&\geq \frac{1}{a_1} + k \cdot c \geq (k+1) \cdot c,
\end{aligned}$$

from which we can conclude that

$$\mathcal{F}(\mathbf{x}^k) - \mathcal{F}(\mathbf{x}^*) \;\leq\; \frac{2C^2\left(2L + \beta\sqrt{\frac{N}{P}}\right)^2}{(2-\rho_P)\beta} \cdot \frac{1}{k}.$$

■

Though it is difficult to evaluate ρ_P and obtain a proper P such that $\rho_P < 2$, in practice, we can always start with $P = 1$ and iteratively increase P according to the objective function. Specifically, the chosen P in each iteration should ensure strict monotonically decreasing of objective function. The following theorem connects block spectral radius to the RIP constant.

Theorem 2

If matrix $\mathbf{A}$ *satisfies the RIP of order* P *with* $\delta_P \leq 1/3$*, then* $\rho_P < 2$*.*

Proof. For any matrix $\mathbf{M} \in \mathcal{M}$, we have $\mathbf{M} = \mathbf{A}_s^T \mathbf{A}_s$ with s being a set of P elements from P different blocks. Here $\mathbf{A}_s$ is the submatrix whose columns are indexed by s. For any vector $\mathbf{x}$ whose components are zero for the complement s^C of s. We have

$$\|\mathbf{A}_s \mathbf{x}_s\|_2^2 = \|\mathbf{A}\mathbf{x}\|_2^2 \leq (1+\delta_P)\|\mathbf{x}\|_2^2 = (1+\delta_P)\|\mathbf{x}\|_2^2 = (1+\delta_P)\|\mathbf{x}_s\|_2^2. \tag{3.18}$$

Since $\mathbf{A}$ is not normalized, we can normalized $\mathbf{A}$ by column with $\mathbf{A} = \tilde{\mathbf{A}}\mathbf{N}$, where $\mathbf{N}$ is a diagonal matrix with $\mathbf{N}_{i,i} = \|\mathbf{A}_i\|_2$. Therefore we have

$$\|\tilde{\mathbf{A}}\mathbf{N}_s \mathbf{x}_s\|_2^2 \leq (1+\delta_P)\|\mathbf{x}_s\|_2^2. \tag{3.19}$$

From RIP condition, we have the diagonal elements belong to $(1-\delta_P, 1+\delta_P)$. We have

$$\|\mathbf{x}_s\|_2^2 \leq \frac{1}{1-\delta_P}\|\mathbf{N}_s \mathbf{x}_s\|^2. \tag{3.20}$$

Thus we have

$$\|\tilde{\mathbf{A}}\mathbf{N}_s\mathbf{x}_s\|_2^2 \leq \frac{1+\delta_P}{1-\delta_P}\|\mathbf{N}_s\mathbf{x}_s\|_2^2. \tag{3.21}$$

If $\delta_P < 1/3$, we have $\frac{1+\delta_P}{1-\delta_P} < 2$, and the proof of the convergence follows. ■

3.3 Parallel Alternating Direction Method of Multipliers

The alternating direction method of multipliers (ADMM) was originally introduced by Spingarn and his collaborators [38] for solving separable convex optimization problem. It solves the following convex optimization problem with linear constraints

$$\begin{aligned} &\min_{\mathbf{x},\mathbf{y}} \ f(\mathbf{x}) + g(\mathbf{y}) \\ &\text{s.t. } \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} = \mathbf{c}, \end{aligned} \tag{3.22}$$

where $\mathbf{A} \in \mathbb{R}^{m\times n_1}$, $\mathbf{B} \in \mathbb{R}^{m\times n_2}$, $f(\mathbf{x})$ and $g(\mathbf{y})$ are convex functions. Though our problem (2.1) does not have the above form, it can be equivalently reformulated to (3.22) through introducing splitting variables.

The ADMM is based on the augmented Lagrangian function, which is defined by

$$L(\mathbf{x},\mathbf{y},\mathbf{z}) = f(\mathbf{x}) + g(\mathbf{y}) + \mathbf{z}^T(\mathbf{A}\mathbf{x}+\mathbf{B}\mathbf{y}-\mathbf{c}) + \frac{\beta}{2}\|\mathbf{A}\mathbf{x}+\mathbf{B}\mathbf{y}-\mathbf{c}\|_2^2, \tag{3.23}$$

where $\mathbf{z}$ is the Lagrangian multiplier, and $\beta > 0$ is the penalty parameter. The ADMM takes advantage of the separability structure of (3.22) and minimizes with respect to $\mathbf{x}$ and $\mathbf{y}$ alternatively in each iteration, after which the Lagrangian multiplier $\mathbf{z}$ is updated. Scaling $\mathbf{z}$ by a factor of $\frac{1}{\beta}$, we summarize the ADMM in Algorithm 4.

Algorithm 4 Alternating Direction Method of Multipliers (ADMM)

1: Initialize $\mathbf{x}^0$, $\mathbf{y}^0$, $\mathbf{z}^0$, β;

2: **for** $k = 1, 2, \ldots, K$ **do**

3: $\quad \mathbf{x}^{k+1} \leftarrow \min_{\mathbf{x}} f(\mathbf{x}) + g(\mathbf{y}^k) + \frac{\beta}{2}\|\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y}^k - \mathbf{b} - \mathbf{z}^k\|_2^2$;

4: $\quad \mathbf{y}^{k+1} \leftarrow \min_{\mathbf{y}} f(\mathbf{x}^{k+1}) + g(\mathbf{y}) + \frac{\beta}{2}\|\mathbf{A}\mathbf{x}^{k+1} + \mathbf{B}\mathbf{y} - \mathbf{b} - \mathbf{z}^k\|_2^2$;

5: $\quad \mathbf{z}^{k+1} \leftarrow \mathbf{z}^k - (\mathbf{A}\mathbf{x}^{k+1} + \mathbf{B}\mathbf{y}^{k+1} - \mathbf{b})$.

6: **end for**

In general there are two ways to parallelize ADMM including parallelizing the operations of the original algorithm and reformulating the original problem into a parallelizable form. In section 3.3.1, we give the details of parallelizing the original ADMM; In section 3.3.2 and section 3.3.3, we reformulate the original problem into primal and dual parallelizable forms respectively.

3.3.1 Parallel the original ADMM

To apply ADMM to (2.1), we need to reformulate it to the following standard form

$$\begin{aligned} &\min_{\mathbf{x},\mathbf{y}} \ \lambda\mathcal{R}(\mathbf{x}) + \mathcal{L}(\mathbf{y}, \mathbf{b}) \\ &\text{s.t. } \mathbf{A}\mathbf{x} - \mathbf{y} = \mathbf{0}. \end{aligned} \tag{3.24}$$

Notice that the prox-linear method can update $\mathbf{x}^{k+1}$ and be parallelized according to the discussions in section 3.1. To update $\mathbf{y}^{k+1}$, we can apply steepest descent method,

$$\mathbf{y}^{k+1} \leftarrow \mathbf{y}^k - \alpha_k \left(\nabla L(\mathbf{y}^k, \mathbf{b}) + \beta(\mathbf{y}^k + \mathbf{b}^k + \mathbf{z}^k - \mathbf{A}\mathbf{x}^{k+1})\right)$$

which only involves matrix-vector multiplication and other cheaper operations, hence it can be parallelized. Updating $\mathbf{z}^{k+1}$ in parallel is trivial and we refer readers to section 2.3.2 for details.

3.3.2 Apply ADMM to the primal problem

In this section, we apply ADMM to (2.1) in a distributed fashion. Assume $\mathcal{L}(\mathbf{Ax}, \mathbf{b})$ is the summation of loss functions, then we can reformulate (2.1) as follows

$$\begin{aligned} \min_{\mathbf{x},\mathbf{x}_{(i)}} \quad & \sum_{i=1}^{M} \mathcal{L}_i(\mathbf{A}_{(i)}\mathbf{x}_{(i)}, \mathbf{b}_i) \quad +\lambda\mathcal{R}(\mathbf{x}) \\ \text{s.t.} \quad & \begin{bmatrix} I & & \\ & \ddots & \\ & & I \end{bmatrix} \begin{bmatrix} \mathbf{x}_{(1)} \\ \vdots \\ \mathbf{x}_{(M)} \end{bmatrix} - \begin{bmatrix} I \\ \vdots \\ I \end{bmatrix} \mathbf{x} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}, \end{aligned} \tag{3.25}$$

where $\mathbf{x}_{(i)} \in \mathbb{R}^n$ are copies of $\mathbf{x}$. Notice (3.25) is in the separable form, so ADMM can be applied and lead to the following parallel primal ADMM algorithm.

Since $\mathbf{x}_{(1)}, \mathbf{x}_{(2)}, \cdots, \mathbf{x}_{(M)}$ are independent to each other, we can update them in parallel as indicated by step 3. So compared to the original ADMM which requires to solve a relative large scale problem to update $\mathbf{x}^{k+1}$, reformulated problem (3.25) breaks step 3 of Algorithm 4 down to many smaller problems. Note that steps 4 & 5 are due to

$$\frac{\beta}{2}\left\| \begin{bmatrix} I & & \\ & \ddots & \\ & & I \end{bmatrix} \begin{bmatrix} \mathbf{x}_{(1)} \\ \vdots \\ \mathbf{x}_{(M)} \end{bmatrix} - \begin{bmatrix} I \\ \vdots \\ I \end{bmatrix} \mathbf{x} - \begin{bmatrix} \mathbf{z}_1 \\ \vdots \\ \mathbf{z}_M \end{bmatrix} \right\|_2^2 = \frac{\beta M}{2} \left\| \mathbf{x} - \frac{1}{M}\sum_i (\mathbf{x}_{(i)} - \mathbf{z}_i) \right\|_2^2 + C,$$

Algorithm 5 Parallel Primal Alternating Direction Method of Multipliers (PP-ADMM)

1: Initialize $\mathbf{x}_{(i)}^0$, $\mathbf{z}_{(i)}^0$, β;

2: **for** $k = 1, 2, \ldots, K$ **do**

3: $\mathbf{x}_{(i)}^{k+1} \leftarrow \min_{\mathbf{x}_{(i)}} \mathcal{L}_i(\mathbf{A}_i\mathbf{x}_{(i)}, \mathbf{b}_i) + \frac{\beta}{2}\|\mathbf{x}_{(i)} - \mathbf{x}^k - \mathbf{z}_i^k\|_2^2$, for every i in parallel;

4: allreduce $\frac{1}{M}\sum_i(\mathbf{x}_{(i)}^{k+1} - \mathbf{z}_i^k)$ for all nodes;

5: $\mathbf{x}^{k+1} \leftarrow \mathcal{R}(\mathbf{x}) + \frac{\beta M}{2}\left\|\mathbf{x} - \frac{1}{M}\sum_i(\mathbf{x}_{(i)}^{k+1} - \mathbf{z}_i^k)\right\|_2^2$ in parallel;

6: $\mathbf{z}_i^{k+1} \leftarrow \mathbf{z}_i^k - (\mathbf{x}_{(i)}^{k+1} - \mathbf{x}^{k+1})$ for every i in parallel.

7: **end for**

where C is a constant that is independent of $\mathbf{x}$. In step 5, calculating $\mathbf{x}^{k+1}$ in parallel means each process does the same computation. Though duplicated computation can be avoided by computing $\mathbf{x}^{k+1}$ with one process and broadcasting it to the others, communication overhead slows down the algorithm, so it is preferable to duplicate the computation of step 5 in each process.

3.3.3 Apply ADMM to the dual problem

By introducing a spliting variable $\mathbf{y}$, we can reformulate (2.1) to

$$\begin{aligned} \min_{\mathbf{x},\mathbf{y}} \quad & \lambda \cdot \mathcal{R}(\mathbf{x}) + \mathcal{L}(\mathbf{y}, \mathbf{b}) \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} - \mathbf{y} = 0, \end{aligned} \tag{3.26}$$

from which we obtain the dual problem

$$\begin{aligned}
&\min_{\mathbf{x},\mathbf{y}} \ \max_{\mathbf{z}} \{\lambda\mathcal{R}(\mathbf{x}) + \mathcal{L}(\mathbf{y},\mathbf{b}) + \mathbf{z}^T(\mathbf{A}\mathbf{x}-\mathbf{y})\} \\
\Leftrightarrow &\max_{\mathbf{z}} \ \min_{\mathbf{x},\mathbf{y}} \ \Big\{\lambda\mathcal{R}(\mathbf{x}) + \mathcal{L}(\mathbf{y},\mathbf{b}) + \mathbf{z}^T(\mathbf{A}\mathbf{x}-\mathbf{y})\Big\} \\
\Leftrightarrow &\max_{\mathbf{z}} \ \Big\{\min_{\mathbf{x}}\{\lambda\mathcal{R}(\mathbf{x}) + \mathbf{z}^T\mathbf{A}\mathbf{x}\} + \min_{\mathbf{y}}\{\mathcal{L}(\mathbf{y},\mathbf{b}) - \mathbf{z}^T\mathbf{y}\}\Big\} \\
\Leftrightarrow &\max_{\mathbf{z}} \ \mathcal{G}(\mathbf{A}^T\mathbf{z}) + \mathcal{H}(\mathbf{z}),
\end{aligned} \tag{3.27}$$

where

$$\mathcal{G}(\mathbf{z}) := \min_{\mathbf{x}}\{\lambda\mathcal{R}(\mathbf{x}) + \mathbf{z}^T\mathbf{x}\}$$

and

$$\mathcal{H}(\mathbf{z}) := \min_{\mathbf{y}}\{\mathcal{L}(\mathbf{y},\mathbf{b}) - \mathbf{z}^T\mathbf{y}\}.$$

In fact, $\mathcal{G}(\mathbf{z}) = -(\lambda\mathcal{R})^*(-\mathbf{z})$ and $\mathcal{H}(\mathbf{z}) = \mathcal{L}^*(\mathbf{z},\mathbf{b})$[1]. Assume $\mathcal{R}(\mathbf{x}) = \sum_{j=1}^{N}\mathcal{R}_j(\mathbf{x}_j)$ (block separable), then $\mathcal{G}$ is also separable, i.e.,

$$\mathcal{G}(\mathbf{A}^T\mathbf{z}) = \sum_j \mathcal{G}_j(\mathbf{A}_j^T\mathbf{z}).$$

Based on the this assumption, we can reformulate the dual problem to the following separable form through introducing splitting variables $\mathbf{y}_j$ and duplicate variables $\mathbf{z}_{(j)}$,

$$\begin{aligned}
&\min_{\mathbf{y}_j,\mathbf{z}_j,\mathbf{z}} \ -\frac{1}{N}\sum_j \mathcal{H}(\mathbf{z}_{(j)}) - \sum_j \mathcal{G}_j(\mathbf{y}_j) \\
&\text{s.t.} \quad \begin{bmatrix} \mathbf{A}_1^T & & \\ & \ddots & \\ & & \mathbf{A}_N^T \end{bmatrix} \begin{bmatrix} \mathbf{z}_{(1)} \\ \vdots \\ \mathbf{z}_{(N)} \end{bmatrix} - \begin{bmatrix} \mathbf{I} & & \\ & \ddots & \\ & & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_N \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}
\end{aligned} \tag{3.28}$$

[1] $f^*(\mathbf{x})$ denotes the conjugate function of $f(\mathbf{x})$.

$$\begin{bmatrix} \mathbf{I} & & \\ & \ddots & \\ & & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{z}_{(1)} \\ \vdots \\ \mathbf{z}_{(N)} \end{bmatrix} - \begin{bmatrix} \mathbf{I} \\ \vdots \\ \mathbf{I} \end{bmatrix} \mathbf{z} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}, \tag{3.29}$$

from which the parallel dual alternating direction method of multipliers is derived in Algorithm 6.

Algorithm 6 Parallel Dual Alternating Direction Method of Multipliers (PD-ADMM)

1: Initialize $\mathbf{y}_j^0$, $\mathbf{z}^0$, $\mathbf{z}_{(j)}^0$, and dual variables $\mathbf{x}_j^0$, $\mathbf{p}_i^0$.

2: **for** $k = 1, 2, \ldots, K$ **do**

3: $\mathbf{y}_i^{k+1} \leftarrow \arg\min_{\mathbf{y}_i} -\mathcal{G}_i(\mathbf{y}_i) + \frac{\alpha}{2}\|\mathbf{A}_i\mathbf{z}_{(i)}^k - \mathbf{y}_i - \mathbf{x}_i^k\|_2^2$, for every i in parallel;

4: $\mathbf{z}_{(i)}^{k+1} \leftarrow \arg\min_{\mathbf{z}_{(i)}} -\frac{1}{N}\mathcal{H}(\mathbf{z}_{(i)}) + \frac{\alpha}{2}\|\mathbf{A}_i\mathbf{z}_{(i)} - \mathbf{y}_i^{k+1} - \mathbf{x}_i^k\|_2^2 + \frac{\beta}{2}\|\mathbf{z}_{(i)} - \mathbf{z}^k - \mathbf{p}_i^k\|_2^2$, for every i in parallel;

5: allreduce $\mathbf{z}^{k+1} \leftarrow \frac{1}{N}\sum_i^N (\mathbf{z}_{(i)}^{k+1} - \mathbf{p}_i^k)$ for all nodes;

6: $\mathbf{x}_{(i)}^{k+1} \leftarrow \mathbf{x}_{(i)}^k - (\mathbf{A}_i^T\mathbf{z}_{(i)}^{k+1} - \mathbf{y}_i^{k+1})$ for every i in parallel;

7: $\mathbf{p}_i^{k+1} \leftarrow \mathbf{p}_i^k - (\mathbf{z}_{(i)}^{k+1} - \mathbf{z}^{k+1})$ for every i in parallel.

8: **end for**

Compared to PP-ADMM, PD-ADMM has two groups of Lagrangian multipliers, where $\mathbf{x}_1, \cdots, \mathbf{x}_N$ correspond to the first group of constraints and $\mathbf{p}_1, \cdots \mathbf{p}_N$ relate to the second group of constraints. In the next section, we give an example of applying PD-ADMM to LASSO.

3.3.4 An example: LASSO

In order to transform the original problem LASSO (2.3) into the separable form, we introduce a duplicate variable $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}$, and derive the separable formation

$$\begin{aligned} &\min_{\mathbf{x}\in\mathbb{R}^n, \mathbf{r}\in\mathbb{R}^m} \ \lambda\|\mathbf{x}\|_1 + \frac{1}{2}\|\mathbf{r}\|_2^2 \\ &s.t. \ \mathbf{A}\mathbf{x} + \mathbf{r} = \mathbf{b}, \end{aligned} \tag{3.30}$$

from which the dual problem is derived as

$$\begin{aligned} &\max_{\mathbf{y}\in\mathbb{R}^m} \min_{\mathbf{x}\in\mathbb{R}^n, \mathbf{r}\in\mathbb{R}^m} \left\{ \|\mathbf{x}\|_1 + \frac{1}{2}\|\mathbf{r}\|_2^2 - \mathbf{y}^T(\mathbf{A}\mathbf{x} + \mathbf{r} - \mathbf{b}) \right\} \\ \Longleftrightarrow &\max_{\mathbf{y}\in\mathbb{R}^m} \left\{ \mathbf{b}^T\mathbf{y} - \frac{1}{2}\|\mathbf{y}\|^2 : \frac{1}{\lambda}\mathbf{A}^T\mathbf{y} \in \mathbb{B}_1^\infty \right\}, \end{aligned} \tag{3.31}$$

where $\mathbb{B}_1^\infty \triangleq \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x}\|_\infty \leq 1\}$. To parallelize it according to scenario (b), we introduce several copies of variables and transform (3.31) to the form of (3.28) as shown below

$$\begin{aligned} &\min_{\mathbf{y}_l, \mathbf{y}, \mathbf{z}} \ \sum_{l=1}^{N} \left(\frac{1}{N}\mathbf{b}^T\mathbf{y}_l + \frac{1}{2N}\|\mathbf{y}_l\|_2^2 + \ell\{\|\mathbf{z}_l\|_\infty \leq 1\} \right) \\ &\text{s.t. } \mathbf{A}_l^T\mathbf{y}_l + \lambda\mathbf{z}_l = 0 \quad \forall\ l = 1, \cdots, N \\ &\mathbf{y}_l - \mathbf{y} = 0 \quad \forall\ l = 1, \cdots, N. \end{aligned} \tag{3.32}$$

By applying ADMM to (3.32), we have the ADMM iterations as described below

$$\begin{aligned} \mathbf{z}_l^{k+1} &= \arg\min_{\mathbf{z}_l} \ \mathcal{L}(\mathbf{z}_l, \mathbf{y}^k, \mathbf{y}_l^k, \mathbf{x}^k, \mathbf{p}^k) \\ \mathbf{y}^{k+1} &= \arg\min_{\mathbf{y}} \ \mathcal{L}(\mathbf{z}_l^{k+1}, \mathbf{y}, \mathbf{y}_l^k, \mathbf{x}^k, \mathbf{p}^k) \\ \mathbf{y}_l^{k+1} &= \arg\min_{\mathbf{y}_l} \ \mathcal{L}(\mathbf{z}_l^{k+1}, \mathbf{y}^{k+1}, \mathbf{y}_l, \mathbf{x}^k, \mathbf{p}^k) \\ \mathbf{x}_l^{k+1} &= \mathbf{x}_l^k - \gamma(\mathbf{A}_l^T\mathbf{y}_l^{k+1} + \lambda\mathbf{z}_l^{k+1}) \\ \mathbf{p}_l^{k+1} &= \mathbf{p}_l^k - \gamma(\mathbf{y}_l^{k+1} - \mathbf{y}^{k+1}), \end{aligned} \tag{3.33}$$

where $\mathcal{L}(\mathbf{z}_l, \mathbf{y}, \mathbf{y}_l, \mathbf{x}, \mathbf{p})$ is the augmented Lagrangian function which is given by

$$\mathcal{L}(\mathbf{z}_l, \mathbf{y}, \mathbf{y}_l, \mathbf{x}, \mathbf{p}) = \sum_{l=1}^{N}(\frac{1}{N}\mathbf{b}^T\mathbf{y}_l + \frac{1}{2N}\|\mathbf{y}_l\|_2^2 + \ell\{\|\mathbf{z}_l\|_\infty \leq 1\} + \frac{\alpha}{2}\|\mathbf{y}_l - \mathbf{y} - \mathbf{p}_l\|_2^2 + \frac{\beta}{2}\|\mathbf{A}_l^T\mathbf{y}_l + \lambda\mathbf{z}_l - \mathbf{x}_l\|_2^2),$$

where $\mathbf{x}_l$, $\mathbf{p}_l$ are multipliers and α, β are the penalty parameters. For given $(\mathbf{z}_l^k, \mathbf{y}^k, \mathbf{y}_l^k, \mathbf{x}^k, \mathbf{p}^k)$, we obtain the next iteration $(\mathbf{z}_l^{k+1}, \mathbf{y}^{k+1}, \mathbf{y}_l^{k+1}, \mathbf{x}^{k+1}, \mathbf{p}^{k+1})$ by applying alternating minimization to augmented Lagrangian function as indicated by (3.33). Firstly, it is easy to show that if $\mathbf{y}^k, \mathbf{y}_l^k, \mathbf{x}^k, \mathbf{p}^k$ are fixed, $\mathbf{z}_l$ $(l = 1, 2, ..., B)$ are updated by

$$\mathbf{z}_l^{k+1} = \text{Proj}_{\mathbb{B}_1^\infty}\left(\frac{1}{\lambda}(\mathbf{x}_l^k - \mathbf{A}_l^T\mathbf{y}_l^k)\right).$$

Secondly, for fixed $\mathbf{z}_l^{k+1}$, $\mathbf{y}_l^k, \mathbf{x}^k, \mathbf{p}^k$, $\mathbf{y}$ is updated by

$$\mathbf{y}^{k+1} = \frac{1}{B}\sum_{l=1}^{N}(\mathbf{y}_l^k - \mathbf{p}_l^k).$$

Thirdly, to update $\mathbf{y}_l$, we keep $\mathbf{z}_l = \mathbf{z}_l^{k+1}$, $\mathbf{y} = \mathbf{y}^{k+1}, \mathbf{x}^k, \mathbf{p}^k$ fixed and solve

$$\min_{\mathbf{y}_l} \mathbf{b}^T\mathbf{y}_l + \frac{1}{2B}\|\mathbf{y}_l\|_2^2 + \frac{\alpha}{2}\|\mathbf{y}_l - \mathbf{y}^{k+1} - \mathbf{p}_l^k\|_2^2 + \frac{\beta}{2}\|\mathbf{A}_l^T\mathbf{y}_l + \lambda\mathbf{z}_l^{k+1} - \mathbf{x}_l^k\|_2^2),$$

whose solution is given by

$$\mathbf{y}_l^{k+1} = \left((\alpha + \frac{1}{B})\mathbf{I} + \beta\mathbf{A}_l\mathbf{A}_l^T\right)^{-1}\left(\alpha(\mathbf{y}^{k+1} + \mathbf{p}_l) + \beta\mathbf{A}_l(\mathbf{x}_l^k - \mathbf{z}_l^{k+1}) - \frac{\mathbf{b}}{N}\right).$$

Combine the previous update iterations, the PD-ADMM algorithm for LASSO is summarized as shown in Algorithm 7.

3.3.5 Convergence analysis

Many convergence results for ADMM have been discussed in the literature. We briefly summarize the basic and general convergence results from [6]. Assume $f(\mathbf{x})$ and $g(\mathbf{y})$

Algorithm 7 PD-ADMM for LASSO

1: Initialize $\mathbf{x}, \mathbf{y}, \mathbf{y}_l, \mathbf{z}, \mathbf{p}$

2: **while** not converged **do**

3: $\mathbf{z}_l^{k+1} \leftarrow \text{Proj}_{\mathbb{B}_1^\infty}(\frac{1}{\lambda}(\mathbf{x}_l^k - \mathbf{A}_l^T \mathbf{y}_l^k))$;

4: $\mathbf{y}^{k+1} \leftarrow \frac{1}{N}\sum_{l=1}^{B}(\mathbf{y}_l^k - \mathbf{p}_l^k)$;

5: $\mathbf{y}_l^{k+1} \leftarrow ((\alpha + \frac{1}{N})\mathbf{I} + \beta \mathbf{A}_l \mathbf{A}_l^T)^{-1}(\alpha(\mathbf{y}^{k+1} + \mathbf{p}_l) + \beta \mathbf{A}_l(\mathbf{x}_l^k - \mathbf{z}_l^{k+1}) - \frac{\mathbf{b}}{N})$;

6: $\mathbf{x}_l^{k+1} \leftarrow \mathbf{x}_l^k - \gamma(\mathbf{A}_l^T \mathbf{y}_l^{k+1} + \lambda \mathbf{z}_l^{k+1})$;

7: $\mathbf{p}_l^{k+1} \leftarrow \mathbf{p}_l^k - \gamma(\mathbf{y}_l^{k+1} - \mathbf{y}^{k+1})$.

8: **end while**

are closed proper and convex functions and assume that the Lagrangian function has a saddle point, then the ADMM iterates satisfy the following three properties: firstly, the iterates approach feasibility; secondly, the objective function converges to the optimal value; thirdly, the dual variable converges to the optimal solution.

Chapter 4

Numerical Experiments

In this chapter, we apply our proposed parallel prox-linear algorithm and GRock algorithm to LASSO and sparse logistic regression, and we compare them with the state-of-the-art algorithm ADMM[6]. The fast convergence speed of GRock is demonstrated and the elasticity of the three algorithms are evaluated.

4.1 LASSO

In this section, we conduct various experiments of LASSO to demonstrate the fast convergence speed as well as the elasticity of GRock.

4.1.1 Greedy rocks for sparse optimization

In general optimization problems, coordinate descent methods update only a few coordinates in each iteration and thus take more iterations than gradient or prox-linear methods. However, in sparse optimization, it is the opposite with Greedy CD since its coordinate selections often occur on those corresponding to nonzero entries in the solution. As such selection will keep most entries in $\mathbf{x}$ as zero throughout the iterations, the problem dimension is effectively reduced. On the contrary, prox-linear iterations typically have dense intermediate solutions at the beginning, and they can

take fairly many iterations before the intermediate solutions become finally sparse.

To illustrate this, we compare GRock to the other three selection schemes such as Cyclic CD, Greedy CD [28] and Mixed CD [36] and apply them to LASSO. The convergence results of the different coordinate descent methods are shown in Figure 4.1. For Mixed CD and GRock, we partition the matrix $\mathbf{A} \in \mathbb{R}^{512\times1024}$ by column into 64 blocks, and we select 8 blocks to update in each iteration. Figure 4.1 shows that greedy selection is effective for LASSO. In particular, Figure 4.1 highlights the fastest convergence of our GRock algorithm for this example.

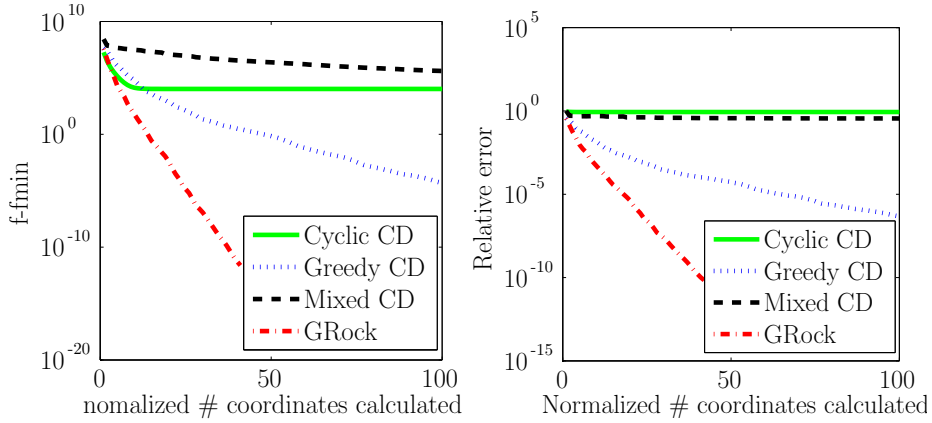


Figure 4.1 : A comparison of different coordinate descent methods. The left figure illustrates the convergence with respect to objective function; the right figure demonstrate the convergence with respect to the relative error.

Moreover, we conduct a group of experiments to highlight the advantages of GRock. In the first group of experiments, the matrix $\mathbf{A} \in \mathbb{R}^{128\times256}$ whose entries are drawn from the standard normal distribution; we test the performance of the four coordinate descent methods on different sparsity level (sparsity ranges from 2 to 40) and different dynamic range of the true solution $\mathbf{x}$. Specifically, We test the data on the following

five ranges: $10^1, 10^2, 10^3, 10^4, 10^5$, set the maximum number of iterations to 500 and set the tolerance of relative error to 10^{-7}. Similar to the previous experiment, $\mathbf{b}$ is generated by the method in [2] such that the given sparse vector $\mathbf{x}$ is the exact solution of the LASSO.

Figure 4.2 shows the convergence results for the four methods with $\lambda = 0.1$, where x label and y label represent the dynamic range of $\mathbf{x}^*$ and the different level of sparsity respectively. The value corresponding to the number of iterations for them to meet the stopping criterion. As indicated by the figure, GRock works the best among the

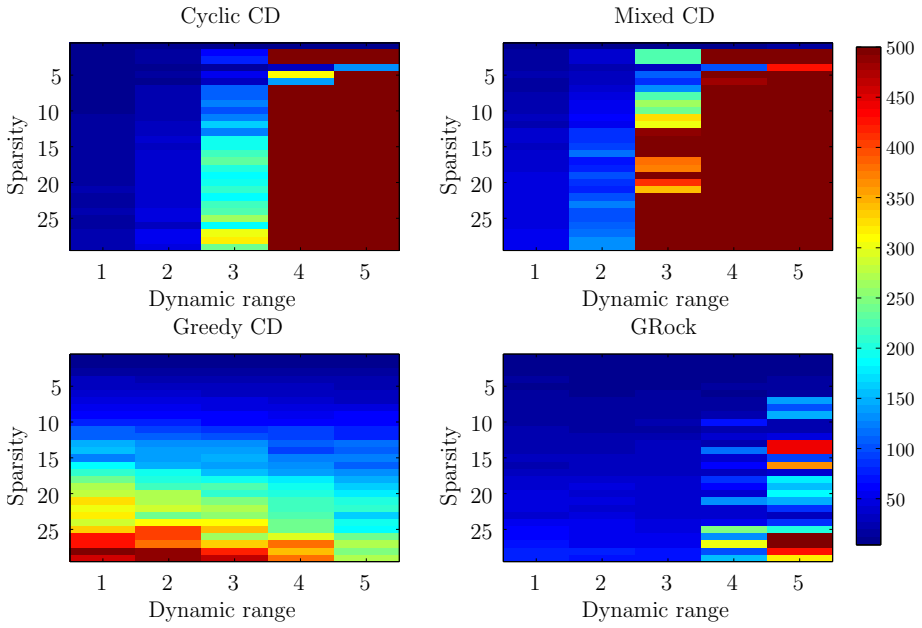


Figure 4.2 : A comparison of different coordinate descent methods with different sparsity level and dynamic range of $\mathbf{x}$; the ith column corresponding to the range of 10^i; the color represents number of iterations.

four methods. In particular, GRock and greedy CD work better than the other two methods for high dynamic range signals. For small dynamic range and high sparsity

level signal, GRock and cyclic CD outperforms the others.

Figure 4.3 shows the convergence results for the four methods with $\lambda = 1$. The convergence results are better than the ones in Figure 4.2, because larger λ makes it easier to identify the nonzero components, and as a consequence faster convergence can be achieved.

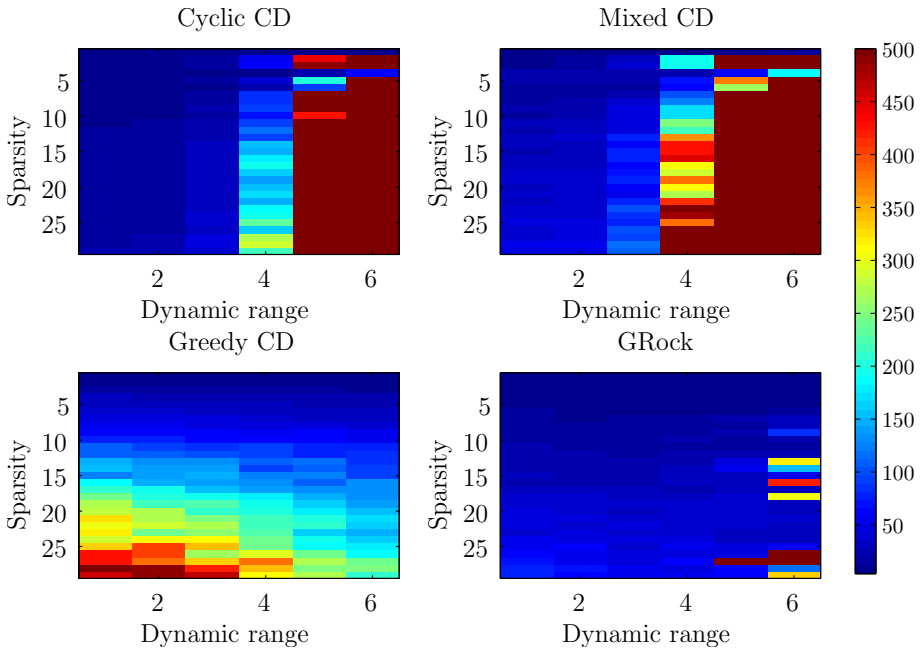


Figure 4.3 : A comparison of different coordinate descent methods with different sparsity level and dynamic range of $\mathbf{x}$; the ith column corresponding to the range of 10^i; the color represents number of iterations.

In addition, we test the GRock algorithm on the *mug*025_12_12.*mat* dataset which is created by the authors in [7]. The matrix $\mathbf{A}$ of this dataset has dimension of $[6205, 24820]$ with about 0.4% nonzero entries. Convergence results are shown in Figure 4.4. The "optimal" solution $\mathbf{x}^*$ is calculated by FISTA with 400 iterations. The algorithm is terminated after 100 iterations or $\frac{\|\mathbf{x}^k - \mathbf{x}^*\|}{\|\mathbf{x}^*\|} \leq 10^{-11}$.

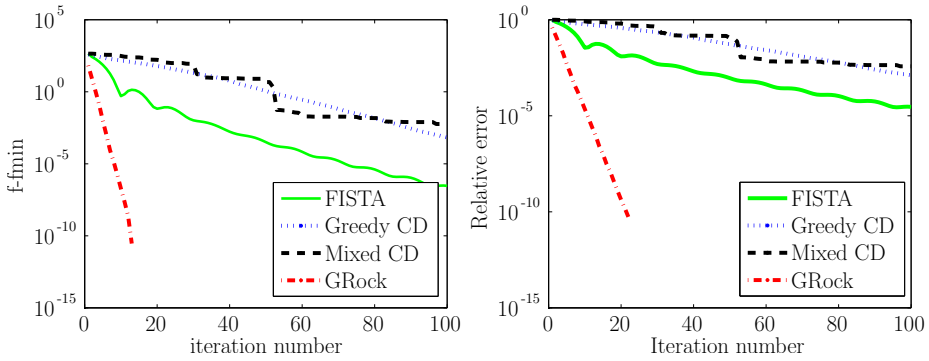


Figure 4.4 : A comparison of different coordinate descent methods and FISTA.

4.1.2 Synthetic small datasets

Our parallel performance tests use both a cluster at Rice University and Amazon EC2. The cluster at Rice consists of 170 Appro Greenblade E5530 nodes each with two quad-core 2.4GHz Xeon (Nahalem) CPUs. Each node has 12GB of memory shared by all cores on the node. The number of processes used is equal to that of the cores.

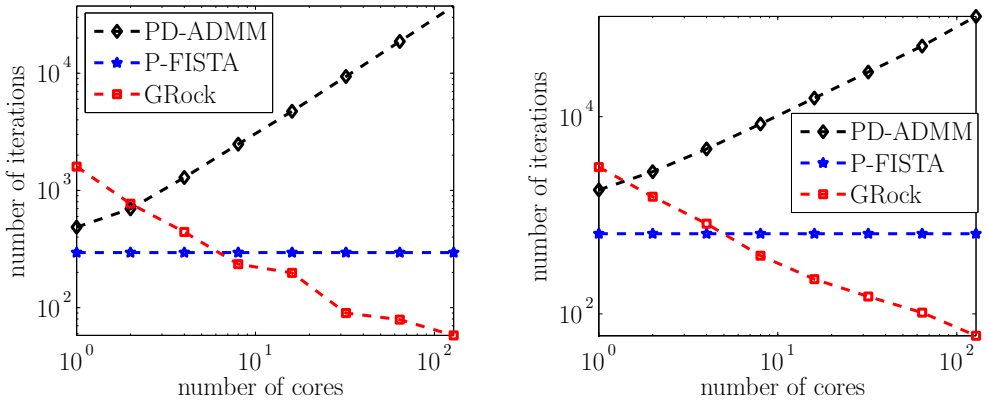
We compare three distributed algorithms on LASSO, including parallel ADMM applied to the Lagrange dual of LASSO (PD-ADMM), P-FISTA, and GRock. Two instances of LASSO in the test are described in Table 4.1.

Table 4.1 : datasets tested on Rice cluster

	$\mathbf{A}$ type	$\mathbf{A}$ size	λ	sparsity of $\mathbf{x}^*$
dataset I	Gaussian	1024×2048	0.1	100
dataset II	Gaussian	2048×4096	0.01	200

Observation vector $\mathbf{b}$ is obtained by the method proposed in [2] such that the given sparse vector $\mathbf{x}^*$ is the optimal solution to the LASSO problem. The stopping criterion is set to $\frac{\|\mathbf{x}^k - \mathbf{x}^*\|}{\|\mathbf{x}^*\|} \leq 10^{-11}$. In P-FISTA, stepsize δ_k is set to $\frac{1}{\|\mathbf{A}\|_2^2}$. In GRock, we set $P = N$, the number of data blocks. The test tries $N = 1, 2, 4, 8, 16, 32, 64, 128$.

Fig. 4.5(a) and 4.5(b) show total number of iterations vs number of cores. As expected, the numbers of iterations of P-FISTA remain constant. PD-ADMM has its number of iterations linearly increasing with the number of processes, which is caused by the variable duplications in the ADMM formulation (see section 8 in [6]). On the other hand, GRock takes fewer iterations as increasing $N = P$ means more coordinates being updated at each iteration.

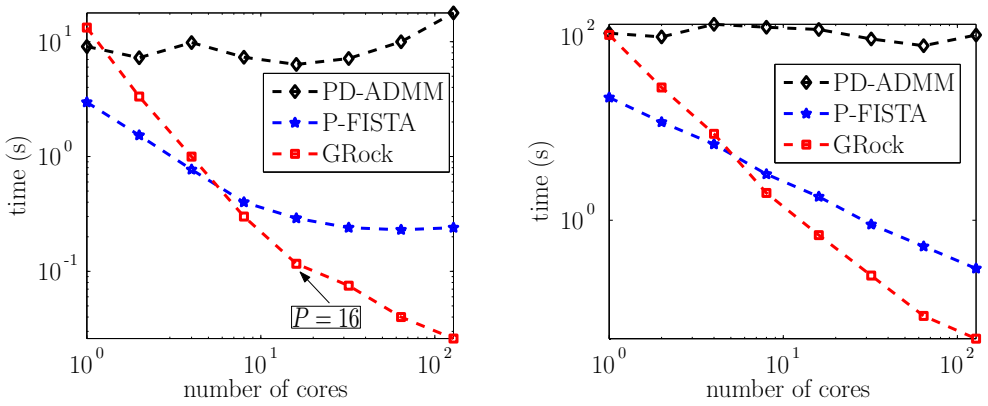


(a) Dataset I (b) Dataset II

Figure 4.5 : Cores vs iterations

Fig. 4.6(a) and 4.6(b) show total time vs number of cores. PD-ADMM has its total computation time staying roughly constant due to a combined effect of more iterations and cheaper per-iteration cost. P-FISTA has its total time reducing linearly

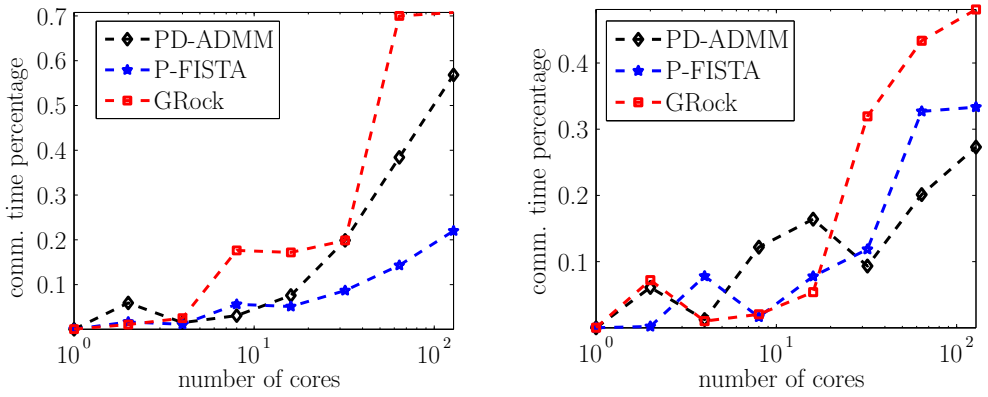
until the communication time begins to signify. GRock reduces its total time faster



(a) Dataset I (b) Dataset II

Figure 4.6 : Cores vs time

than P-FISTA, as a result of both cheaper per-iteration cost and fewer iterations. The results match the percentages of the communication part of total time, both of which are measured on a specific core, given in Fig. 4.7(a) and 4.7(b).



(a) Dataset I (b) Dataset II

Figure 4.7 : Cores vs communication overhead percentage

The percentage is overall linearly correlated to the number of cores. Note that the

matrix-factorization time during the initialization of PD-ADMM is counted in the total time.

Note that we do not have to choose $P = N$ for GRock as in this experiment. For some problems, we may need to choose $P < N$ to ensure the convergence of GRock.

4.1.3 Synthetic large datasets

In this LASSO test, $\mathbf{A}$ is generated by randn$(100000, 200000)$ with 20 billion nonzero entries and having a size of 170GB. The solution $\mathbf{x}^*$ has 4000 nonzero entries, each sampled from $\mathcal{N}(0, 1)$ independently. We set the maximum number of iterations to 2500 and the stopping criterion as $\frac{\|\mathbf{x}^k - \mathbf{x}^*\|}{\|\mathbf{x}^*\|} \leq 10^{-5}$. On Amazon EC2, we requested 20 high-memory quadruple extra-large instances giving us a total of 160 cores and 1.2TB of memory in total. Table 4.2 compares the performance of PD-ADMM, P-FISTA and GRock. As indicated by Table 4.2, it takes 105 minutes for PD-ADMM to perform matrix matrix multiplication and matrix factorization, which are the type of operations that we should avoid for large scale problems. In terms of convergence speed, GRock takes only 104 iterations to meet the stopping criterion, however, the other two methods cannot get to the accuracy of 10^{-5} within 2500 iterations. Note that the performance of PD-ADMM depends on a penalty parameter. We pick it as the best out of only a few trials as we cannot afford more trials.

Table 4.2 : large dataset time results

	PD-ADMM	P-FISTA	GRock
estimate stepsize (min.)	n/a	1.6	n/a
matrix factorization (min.)	51	n/a	n/a
iteration time (min.)	105	40	1.7
number of iterations	2500	2500	104
communication time	30.7	9.5	0.5
stopping relative error	1E-1	1E-3	1E-5
total time (min)	**156**	41.6	**1.7**
cost	**$85**	**$22.6**	**$0.93**

4.2 Sparse Logistic Regression

This section presents various experiments of sparse logistic regression to demonstrate the fast convergence speed as well as the elasticity of GRock.

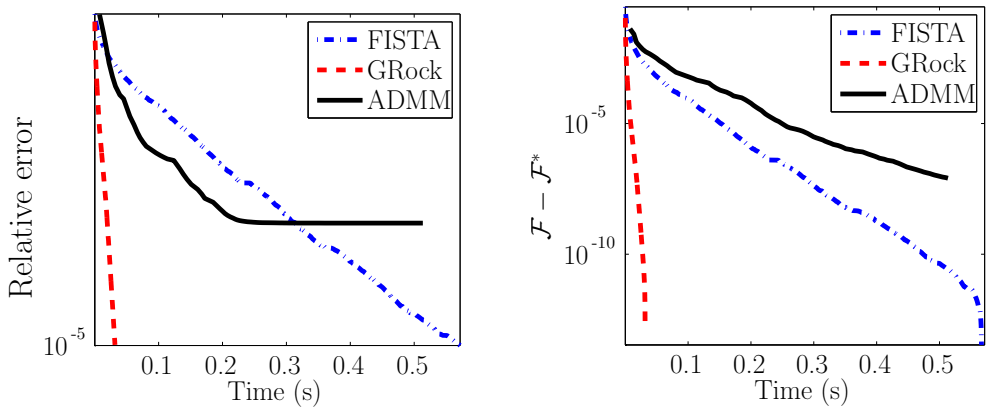
4.2.1 Synthetic small datasets

In this experiment, datasets are generated as shown in Table 4.3. Label vector $\mathbf{b}$ is generated by $\text{sign}(\mathbf{A}\mathbf{w} + v)$, where $\mathbf{w}$ is a given sparse vector, and v is a random scalar. The "optimal" solution $\mathbf{x}^*$ is calculated by CVX [20]. The step size (δ) of FISTA is set to 1. In GRock, we set $P = 16$. The augmented Lagrangian parameter of ADMM is set to 1.

Table 4.3 : Logistic regression datasets

	A	size(**A**)	λ
Dataset I	Gaussian	100×50	0.01
Dataset II	Gaussian	50×100	0.01

Figure 4.8(a) and Figure 4.8(b) show the convergence results for dataset I. As expected, GRock converges faster than the other two algorithms. In terms of relative error of $\mathbf{x}$, ADMM converges faster than FISTA at the beginning, but it can only achieve accuracy of 10^{-4}. In fact, it has been suggested by [6] that ADMM can be very slow to converge to high accuracy.

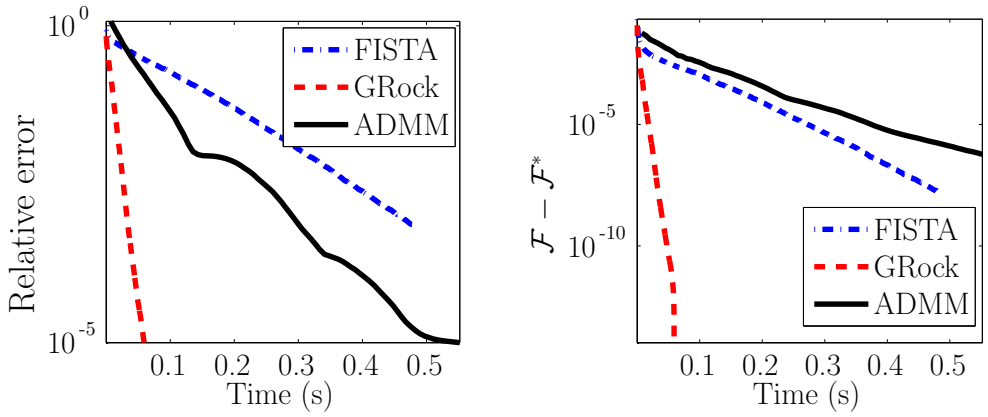


(a) Time vs relative error of $\mathbf{x}$ (b) Time vs accuracy of $\mathcal{F}(\mathbf{x})$

Figure 4.8 : Dataset I

Figure 4.9(a) and Figure 4.9(b) are the convergence results for dataset II. As indicated by the figures, GRock converges faster than the other two algorithms. In

terms of relative error of $\mathbf{x}$, ADMM converges faster than FISTA, but it is slightly slower than FISTA in terms of the decreasing of objective function.



(a) Time vs relative error of $\mathbf{x}$ (b) Time vs accuracy of $\mathcal{F}(\mathbf{x})$

Figure 4.9 : Dataset II

4.2.2 Real datasets

To demonstrate the elasticity of our algorithms, we test our algorithms on the adult dataset from the UCI machine learning repository[1]. Specifically, the dataset has 48842 samples and 123 features. We use 16280 samples as our training dataset and set $\lambda = 0.001$.[2]. We test our codes on 1, 8, 16, 64, and 128 cores on Rice STIC cluster.

Table 4.4 shows the time results for our proposed algorithms. P-FISTA has its total time reducing linearly until the communication time begins to signify (at # of

[1]http://archive.ics.uci.edu/ml/datasets/Adult

[2]Notice that λ could be optimized with cross validation. Since we focus on demonstrating the speed of GRock, we simply set $\lambda = 0.001$.

cores = 128). GRock reduces its total time faster than P-FISTA, as a result of both cheaper per-iteration cost and fewer iterations.

Table 4.4 : Parallel time results for the UCI adult dataset

# of cores	P-FISTA(s)	GRock(s)
1	8.74	1.74
8	2.24	0.48
16	0.6	0.13
64	0.24	0.05
128	0.24	0.07

Chapter 5

Conclusion and Future Work

5.1 Conclusion

This thesis proposes two approaches including distributed implementation of prox-linear algorithms and a parallel greedy coordinate-block descent method (GRock) for solving large-scale sparse optimization problems. Our approaches are motivated by the two typical structures of sparse optimization problems, namely, separable objective functions and rough orthogonality in the data.

The convergence of our parallel prox-linear algorithms are guaranteed by the existing prox-linear method theory. We established the sublinear convergence rate based on block spectral radius for the GRock algorithm and explained why greedy works well for sparse optimization problems.

We carried out extensive numerical experiments with LASSO and sparse logistic regression. Experimental results show the fast convergence speed of GRock. They also indicate that both proposed approaches are more scalable than the popular distributed ADMM method, which is nonetheless more general and has applications beyond sparse optimization.

5.2 Future Work

In this thesis, we applied various parallel and distributed techniques to sparse optimization problems. However, there are still many open questions that are worth further investigation.

5.2.1 Convergence rate for GRock

In section 3.2.3, we established the $O(\frac{1}{k})$ convergence rate for the GRock algorithm. However, empirical evidences lead us to believe that better convergence rate can be achieved by adding some more assumptions. For instance, our numerical results in section 4.1.1 show that the convergence rate of the GRock algorithm is linear. Therefore, we are interested in establishing a better convergence rate for GRock.

5.2.2 Asynchronous algorithms

Our two parallel and distributed algorithms are synchronous, which require a significant amount of coordination or scheduling between nodes. We expect better performance of our algorithms by overlapping communication with computation, but asynchronous communication might lead to divergence. So we would like to conduct some parallel experiments to test the performance of asynchronous algorithms and analysis their convergence.

5.2.3 Regularization with linear transform

Our parallel schemes are based on the assumption that the regularization function $\mathcal{R}(\mathbf{x})$ is block separable, however, some regularizers are not separable. For instance, $\mathcal{R}(\mathbf{x}) = \|\Phi\mathbf{x}\|_1$ where Φ is a linear transform is not block separable in general. However, notice that primal-dual algorithm [12] can be applied to solve this type of problem. Specifically, to solve

$$\min_{\mathbf{x}} \ \lambda \cdot \mathcal{R}(\Phi\mathbf{x}) + \mathcal{L}(\mathbf{A}\mathbf{x}, \mathbf{b}), \tag{5.1}$$

we can reformulate the above problem to the following equivalent form

$$\min_{\mathbf{x}} \ \lambda \cdot \mathcal{F}((\Phi; \mathbf{A})\mathbf{x}) + \mathcal{O}(\mathbf{x}), \tag{5.2}$$

where $\mathcal{F}((\Phi; \mathbf{A})\mathbf{x}) = \lambda \cdot \mathcal{R}(\Phi\mathbf{x}) + \mathcal{L}(\mathbf{A}\mathbf{x}, \mathbf{b})$, and $\mathcal{O}(\mathbf{x}) = 0$. Then, by applying the primal-dual algorithm, we can obtain the following iterations.

$$\begin{cases} \mathbf{y}_1^{k+1} & \leftarrow \mathbf{prox}_{\mathcal{F}}(\mathbf{y}_1^k + \sigma\Phi\bar{\mathbf{x}}^k) \\ \mathbf{y}_2^{k+1} & \leftarrow \mathbf{prox}_{\mathcal{F}}(\mathbf{y}_2^k + \sigma\mathbf{A}\mathbf{x}^k) \\ \mathbf{x}^{k+1} & \leftarrow \mathbf{x}^k - \tau(\Phi^T\mathbf{y}_1^{k+1} + \mathbf{A}^T\mathbf{y}_2^{k+1}) \\ \bar{\mathbf{x}}^{k+1} & \leftarrow \mathbf{x}^{k+1} + \theta(\mathbf{x}^{k+1} - \mathbf{x}^k) \end{cases} \tag{5.3}$$

which only involves matrix and vector multiplication, and can be parallelized by scenarios (a), (b) and (c) as mentioned in section 2.3. The numerical performance of applying (5.3) to various large scale application problems worth further investigation.

Bibliography

[1] Heinz H Bauschke and Jonathan M Borwein. On projection algorithms for solving convex feasibility problems. *SIAM review*, 38(3):367–426, 1996.

[2] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

[3] Dimitri P Bertsekas and John N Tsitsiklis. Parallel and distributed computation. 1989.

[4] Jinbo Bi, Kristin Bennett, Mark Embrechts, Curt Breneman, and Minghu Song. Dimensionality reduction via sparse support vector machines. *The Journal of Machine Learning Research*, 3:1229–1243, 2003.

[5] Alexandre Borghi, Jérôme Darbon, Sylvain Peyronnet, Tony F Chan, and Stanley Osher. A simple compressive sensing algorithm for parallel many-core architectures. *CAM Report*, pages 08–64, 2008.

[6] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction

method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[7] Joseph K Bradley, Aapo Kyrola, Danny Bickson, and Carlos Guestrin. Parallel coordinate descent for ℓ_1 regularized loss minimization. *arXiv preprint arXiv:1105.5379*, 2011.

[8] Jian-Feng Cai, Stanley Osher, and Zuowei Shen. Linearized bregman iterations for compressed sensing. *Math. Comp*, 78(1515-1536):55–59, 2009.

[9] Emmanuel J Candès, Justin Romberg, and Terence Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *Information Theory, IEEE Transactions on*, 52(2):489–509, 2006.

[10] Emmanuel J Candes and Terence Tao. Decoding by linear programming. *Information Theory, IEEE Transactions on*, 51(12):4203–4215, 2005.

[11] Emmanuel J Candès and Michael B Wakin. An introduction to compressive sampling. *Signal Processing Magazine, IEEE*, 25(2):21–30, 2008.

[12] Antonin Chambolle and Thomas Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision*, 40(1):120–145, 2011.

[13] Scott Shaobing Chen, David L Donoho, and Michael A Saunders. Atomic decomposition by basis pursuit. *SIAM journal on scientific computing*, 20(1):33–

61, 1998.

[14] Patrick L Combettes and Jean-Christophe Pesquet. Proximal splitting methods in signal processing. In *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*, pages 185–212. Springer, 2011.

[15] Patrick L Combettes and Valérie R Wajs. Signal recovery by proximal forward-backward splitting. *Multiscale Modeling & Simulation*, 4(4):1168–1200, 2005.

[16] David Leigh Donoho. Compressed sensing. *Information Theory, IEEE Transactions on*, 52(4):1289–1306, 2006.

[17] David Leigh Donoho, Iddo Drori, Yaakov Tsaig, and Jean-Luc Starck. *Sparse solution of underdetermined linear equations by stagewise orthogonal matching pursuit*. Department of Statistics, Stanford University, 2006.

[18] T Len Freeman and Chris Phillips. *Parallel numerical algorithms*. Prentice-Hall, Inc., 1992.

[19] Tom Goldstein and Stanley Osher. The split bregman method for ℓ_1-regularized problems. *SIAM Journal on Imaging Sciences*, 2(2):323–343, 2009.

[20] Michael Grant and Stephen Boyd. CVX: Matlab software for disciplined convex programming, version 2.0 beta. http://cvxr.com/cvx, September 2012.

[21] Antoine Guitton and William W Symes. Robust inversion of seismic data using the huber norm. *Geophysics*, 68(4):1310–1319, 2003.

[22] Elaine T Hale, Wotao Yin, and Yin Zhang. Fixed-point continuation for ℓ_1 minimization: Methodology and convergence. *SIAM Journal on Optimization*, 19(3):1107–1130, 2008.

[23] Peter J Huber. Robust regression: asymptotics, conjectures and monte carlo. *The Annals of Statistics*, 1(5):799–821, 1973.

[24] Sundararaja S Iyengar and Richard R Brooks. *Distributed sensor networks*. Chapman & Hall/CRC Boca Raton, 2005.

[25] Kwangmoo Koh, Seung-Jean Kim, and Stephen Boyd. An interior-point method for large-scale ℓ_1-regularized logistic regression. *Journal of Machine learning research*, 8(8):1519–1555, 2007.

[26] Balaji Krishnapuram, Lawrence Carin, Mario AT Figueiredo, and Alexander J Hartemink. Sparse multinomial logistic regression: Fast algorithms and generalization bounds. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(6):957–968, 2005.

[27] Sangkyun Lee and Stephen J Wright. Implementing algorithms for signal and image reconstruction on graphical processing units. *Computer Sciences Department, University of Wisconsin-Madison, Tech. Rep*, 2008.

[28] Yingying Li and Stanley Osher. Coordinate descent optimization for ℓ_1 minimization with application to compressed sensing; a greedy algorithm. *Inverse Problems and Imaging*, 3(3):487–503, 2009.

[29] Gideon Mann, Ryan McDonald, Mehryar Mohri, Nathan Silberman, and Dan Walker. Efficient large-scale distributed training of conditional maximum entropy models. *Advances in Neural Information Processing Systems*, 22:1231–1239, 2009.

[30] João FC Mota, João MF Xavier, Pedro MQ Aguiar, and Markus Puschel. Distributed basis pursuit. *Signal Processing, IEEE Transactions on*, 60(4):1942–1956, 2012.

[31] Deanna Needell and Joel A Tropp. Cosamp: Iterative signal recovery from incomplete and inaccurate samples. *Applied and Computational Harmonic Analysis*, 26(3):301–321, 2009.

[32] Yurii Nesterov. A method of solving a convex programming problem with convergence rate $o(\frac{1}{k^2})$. *Soviet Mathematics Doklady*, 27(2):372–376, 1983.

[33] Yurii Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *Core discussion papers*, 2:2010, 2010.

[34] Andrew Y Ng. Feature selection, ℓ_1 vs. ℓ_2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78. ACM, 2004.

[35] Peter Richtárik and Martin Takáč. Parallel coordinate descent methods for big data optimization. *arXiv preprint arXiv:1212.0873*, 2012.

[36] Chad Scherrer, Ambuj Tewari, Mahantesh Halappanavar, and David Haglin.

Feature clustering for accelerating parallel coordinate descent. In *NIPS*, pages 28–36, 2012.

[37] Jianing Shi, Wotao Yin, Stanley Osher, and Paul Sajda. A fast hybrid algorithm for large-scale ℓ_1-regularized logistic regression. *The Journal of Machine Learning Research*, 11:713–741, 2010.

[38] Jonathan E Spingarn. Applications of the method of partial inverses to convex programming: decomposition. *Mathematical Programming*, 32(2):199–223, 1985.

[39] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[40] Robert Tibshirani, Michael Saunders, Saharon Rosset, Ji Zhu, and Keith Knight. Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(1):91–108, 2005.

[41] Joel A Tropp. Greed is good: Algorithmic results for sparse approximation. *Information Theory, IEEE Transactions on*, 50(10):2231–2242, 2004.

[42] Joel A Tropp and Anna C Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *Information Theory, IEEE Transactions on*, 53(12):4655–4666, 2007.

[43] Junfeng Yang and Yin Zhang. Alternating direction algorithms for ℓ_1 problems in compressive sensing. *SIAM journal on scientific computing*, 33(1):250–278, 2011.

[44] Wotao Yin. Analysis and generalizations of the linearized bregman method. *SIAM Journal on Imaging Sciences*, 3(4):856–877, 2010.

[45] Wotao Yin, Stanley Osher, Donald Goldfarb, and Jerome Darbon. Bregman iterative algorithms for ℓ_1 minimization with applications to compressed sensing. *SIAM Journal on Imaging Sciences*, 1(1):143–168, 2008.

[46] Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.

[47] Sangwoon Yun and Kim-Chuan Toh. A coordinate gradient descent method for ℓ_1-regularized convex minimization. *Computational Optimization and Applications*, 48(2):273–307, 2011.

[48] Martin Zinkevich, John Langford, and Alex J Smola. Slow learners are fast. In *Advances in Neural Information Processing Systems*, pages 2331–2339, 2009.

[49] Martin Zinkevich, Markus Weimer, Alex Smola, and Lihong Li. Parallelized stochastic gradient descent. *Advances in Neural Information Processing Systems*, 23(23):1–9, 2010.

[50] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.