UNIVERSITY OF CALIFORNIA,
IRVINE

Randomized Fast Solvers for Linear and Nonlinear Problems in Data Science

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Math

by

Huiwen Wu

Dissertation Committee:
Professor Long Chen, Chair
Professor Jack Xin
Chancellor's Professor Hongkai Zhao

2019

DEDICATION

To my family

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my Ph.D advisor, Professor Long Chen, for his wise advice, continuous support and endless patience on my academic research. I have benefited a lot from not only his immense knowledge but also his perspectives as a mathematician.

I would like to express my gratitude to Professor Hal Stern, Professor Tom Trogdon, Professor Nie Qing, Professor Jack Xin and Professor Hongkai Zhao for serving as my committee members and share their insights and expertise on my research. I would like to thank my collaborator Professor Xiaozhe Hu in Tufts University for his enlightening discussions. I would also like to thank all Long's group members and Mathematics Department in University of California, Irvine for all the help and support.

Last but not least, I would express my deep appreciation for my family's unconditional support and encouragement throughout my life.

CURRICULUM VITAE

Huiwen Wu

EDUCATION

Doctor of Philosophy in Math **2019**
University of California, Irvine *Irvine, CA*

Bachelor of Science in Math **2013**
Sichuan University *Chengdu, Sichuan, China*

RESEARCH EXPERIENCE

Graduate Research Assistant **2014–2019**
University of California, Irvine *Irvine, CA*

TEACHING EXPERIENCE

Teaching Assistant **2016–2019**
University of California, Irvine *Irvine, CA*

ABSTRACT OF THE DISSERTATION

Randomized Fast Solvers for Linear and Nonlinear Problems in Data Science

By

Huiwen Wu

Doctor of Philosophy in Math

University of California, Irvine, 2019

Professor Long Chen, Chair

We construct a preconditioner for solving the linear least square problems, which are simplest and most popular arising in data fitting, imaging processing and high dimension data analysis. The existed methods for solving least squares problems either has a large computational cost or depends highly on the condition number of the matrix. Recently, there is a surge of interest in developing randomized algorithms for solving least squares problems for the purpose of efficiency and scalability. We construct a new preconditioner equipped with sampling procedure to reduce computational complexity and apply Gauss Seidel iterations to grab the high frequency component of the solution, which reduces the dependence of performance of the conditioner number. Experimental studies compared with Conjugate Gradient Descent method (CG) are presented on six different simulations including dense Gaussian matrix, 'semi Gaussian' matrix, Sparse random matrix, 'UDV' matrix and Graph Laplacian matrix and a non-negative constrained problem to show the effectiveness of the proposed preconditioner.

A general scheme for solving non-constraint convex and smooth minimization problem $\min_{x\in\mathcal{V}} f(x)$ is developed in this thesis. The scheme does gradient descent on each subspace based on a stable space decomposition of $\mathcal{V}$. With assumptions of Lipschitz continuous of the gradient on $\mathcal{V}$ and its subspaces, convexity or strong convexity on $\mathcal{V}$, we prove linear convergence for strongly convex objective function for both non-uniform sampling and uniform sampling. For non-uniform sampling, the convergence depends on the expected condition number, and for uniform sampling, the convergence depends on the supreme condition number. Moreover, we also show sublinear convergence for convex function. Numerical examples on Nestrov's worst function and linear regression both outperform randomized coordinate method. We conclude that our scheme generalizes the gradient descent methods, randomized (block) coordinate descent methods and full approximation scheme.

Chapter 1

Introduction

During the past two decades, there is a growing enthusiasm for "Big Data". We now come a big data era with great opportunities and challenges. Due to eruption of terabytes data, we confront problems including but not limited to creation, transmitting, processing, and storage capacities.

Due to the randomness of data and problems, randomized algorithms are developed. Compared to deterministic algorithms, randomized versions enjoy the benefits of easier convergence analysis, better numerical performance, and significant complexity reduction.

In this thesis, we focus on two simple but fundamental problems arising in data science. One is solving linear equation:

$$Ax = b, \tag{1.1}$$

where $A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^m, m \gg n$, and $\operatorname{rank}(A) = n$.

Another one is non-constraint smooth and convex minimization problem:

$$\min_{x\in\mathcal{V}} f(x), \tag{1.2}$$

where f is a smooth and convex function and $\mathcal{V}$ is a Hilbert space.

We conclude the development of methods for least squares problems and non-constraint smooth and convex problem and show the intrinsic relation between linear and nonlinear methods in Chapter 3 and Chapter 5. We propose a randomized iterative method based on importance sampling strategy and preconditioning for least squares problem (**??**) and a generic gradient type scheme combined subspace decomposition and randomly gradient descent for (1.2), which are introduced in Chapter 4 and Chapter 6 respectively.

The thesis is organized as follows.

- In Chapter 2, we present the simplest and most common problem in numerical linear algebra – least squares problem. We introduce popular models and associated optimization problems in data science including linear regression and logistic regression. We show that the maximum likelihood settings of linear regression with a linear Gaussian noise is exactly least squares problem. We also derive some fundamental properties of logistic regression. More generally, we provide a non-constraint minimization problem setting with Lipschitz continuous of first order derivative and convex or strongly convex assumptions.

- In Chapter 3, we review existing methods of least squares problem including direct methods, iterative methods and recently developed randomized methods. Direct methods is easy to implement but have limitation that computation complexity is $O(mn^2)$. When the matrix is of huge size, it is not practical and extremely expensive to apply direct methods. Iterative methods have the benefit that in each iteration, the complexity would be $O(n)$ while the drawback is the number of iterations to achieve accuracy ϵ highly depends on condition number of matrix A. As the development of probabilistic learning, several fast least squares solvers have been developed recently utilizing randomization see, for example [15, 37, 4]. These fast least squares solvers try to construct a spectrally equivalent but of smaller size matrix via random transformation and random sampling or mixing. Due to the random transformation, however, these methods destroy the sparsity of A, and thus not suitable for sparse matrices.

- In Chapter 4, we go through our method for least squares problem *non-uniform row sampling fast least square solver* [12]. In our method, we apply row sampling to matrix A in first stage. Then we construct a preconditioner with Gauss-Seidel applied to the sampled matrix A_S. After all, we equip conjugate gradient with preconditioner we have constructed. Our method reduces not only complexity by using iterative scheme but also iteration steps by constructing an efficient preconditioner. By this preconditioner, the sparsity of original problem is preserved. We apply our methods to 5 categories of matrix and improve iteration steps to $1/4$ compared to diagonal preconditioned conjugate gradient in worst case. One application to non-negative constrained problem also shows a huge improvement.

- In Chapter 5, we review gradient type methods for non-constraint smooth and convex minimization problem including gradient descent methods (GD), coordinate descent methods (CD) in cyclic fashion (CCD) and randomized fashion (RCD), stochastic gradient descent methods (SGD) and mini-batch stochastic gradient methods (mini-batch SGD). We present and compare convergence analysis of these methods. We also show the connections between nonlinear methods and linear methods. Apply RCD to linear equation give randomized Gauss-Seidel (RGS) while apply SGD to linear equation give randomized kaczmarz (RK). We show duality between RGS and RK and RCD and SGD.

- In Chapter 6, we introduce our method for non-constraint smooth and convex minimization problem – *randomized fast subspace decent methods (RFASD)*. We combine the idea of RCD and full approximation scheme (FAS) [11]. Suppose we have a stable subspace decomposition $\mathcal{V} = \sum_i \mathcal{V}_i$. Instead of doing gradient descent in each coordinate, doing gradient descent in each subspace achieves better performance as long as the subspace decomposition is stable in appropriate norm so that condition number of original problem is reduced. Numerical examples for Nesterov's worst function [32] compared with RCD show the improvements. Two accelerated versions of RFASD are also provided to further speed up the method.

- We conclude the thesis in Chapter 7. We develop a randomized row sampling preconditioner for least squares problem by importance sampling and Gauss-Seidel iteration. We equip CG with row sampling preconditioner and compare with diagonal preconditioned CG. To solve non-constraint convex and smooth minimization problem, we construct a generic scheme RFASD. We conclude existing methods GD, RCD, RBCD and FAS as examples of RFASD.

Chapter 2

Linear and Nonlinear Problems in Data Science

In this chapter, we introduce one of fundamental problems in numerical linear algebra – least squares problem. Then we come to two basic models in Data Science – linear regression and logistic regression, which have linear and non-linear gradient of objective functions respectively. After all, we give the basic setting of general minimization problem with convex and smooth objective function.

2.1 Least Squares Problem

Least squares method is one of the simplest and most commonly applied techniques of data fitting. It can be applied in statistics to construct linear regression model and unbiased linear estimator [34], in imaging processing for image deblurring [6], and in high-dimensional data analysis like canonical polyadic tensor decomposition [18] etc. Least squares problems arise when applying least squares method and fall into two categories: linear or ordinary least squares or nonlinear least squares. Now we focus on linear least squares problem.

Consider the overdetermined system

$$Ax = b,$$

where $A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^m, m \geq n$, and assume $\mathrm{rank}(A) = n$. As $m > n$, solutions to (1.1) are in general not unique. The least squares solution to the overdetermined system (1.1) is

$$x_{\text{opt}} = \arg\min_x \|Ax - b\|_2^2, \tag{2.1}$$

where $\|\cdot\|_2$ is the l^2 norm of a vector.

Find the least squares solution x_{opt} is mathematically equivalent to solving the normal equation

$$A^\intercal A\, x_{\text{opt}} = A^\intercal b. \tag{2.2}$$

Since A is full rank, $A^\intercal A$ is non-singular and thus $x_{\text{opt}} = (A^\intercal A)^{-1}A^\intercal b$. However, traditional methods to compute $A^\intercal A$ is costing, i.e., $O(n^3)$ for QR or SVD decomposition.

2.2 Linear Regression

Linear Regression is one simple but important model in probabilistic learning. In this section, we discuss the math problem after applying linear regression model and show the connection of linear regression and one of the most fundamental problem in numerical linear algebra – least squares problem. When the assumption for while noise is normal distribution, in order to achieve maximum likelihood, the linear system we confront in linear regression model is exactly least squares problem. Various methods for least squares problems have been developed for least squares problem due to its fundamental role in numerical linear algebra, which will be introduced in Chapter 2. These methods also contribute to solving linear regression problems.

In this section, we discuss the applications of least squares problem in data science. One important and popular application is linear regression. We now show the maximum likelihood settings of linear regression with a linear Gaussian noise is exactly a least squares problem. To illustrate the equivalent relation, we follow notes of Lindsten et al. *Probabilistic modeling - linear regression* & *Gaussian processes* [23].

Suppose we have a list of data $\{y_i, x_{i1}, x_{i2}, \cdots, x_{id}\}_{i=1}^N$ as N input-output data pairs where $x_{ik}, y_i \in \mathbb{R}$ with $k = 1, \cdots, d$.

A linear regression model assumes the linear relationship between dependent variable y and d-vector input data x with a disturbance term ϵ. For a single i, the model takes the form.

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_d x_{id} + \epsilon_i = x_i^\intercal \beta + \epsilon_i,\ i = 1, \cdots, N, \tag{2.3}$$

where $\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \cdots \\ \beta_d \end{bmatrix} \in \mathbb{R}^{d+1}$ and $x_i = \begin{bmatrix} 1 \\ x_{i1} \\ \cdots \\ x_{id} \end{bmatrix} \in \mathbb{R}^{d+1}$.

Combine (2.3) of N data, the model can be defined in a matrix form. The model is described as

$$y = X\beta + \epsilon, \tag{2.4}$$

where

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_N \end{bmatrix}, X = \begin{bmatrix} x_1^\intercal \\ x_2^\intercal \\ \cdots \\ x_N^\intercal \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1d} \\ 1 & x_{21} & x_{22} & \cdots & x_{2d} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & x_{N1} & x_{N2} & \cdots & x_{Nd} \end{bmatrix}, \epsilon = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \cdots \\ \epsilon_N \end{bmatrix}.$$

- Each $\epsilon_i, i = 1, \cdots, N$ is independent identical Gaussian random variables with mean 0 and variance σ^2, i.e. $\epsilon \sim \mathcal{N}(0, \sigma^2)$.
- X is the matrix with each row $x_i^\intercal$ as concatenation of 1 and input data with d features $\begin{bmatrix} x_{i1}, x_{i2}, \cdots x_{id} \end{bmatrix}^\intercal$.
- $\beta \in \mathbb{R}^{d+1}$ is unknown deterministic parameter in maximum likelihood settings.
- $y \in \mathbb{R}^N$ is a vector with each component y_i as labels of corresponding input data x_i.
- In training process, y is known and we need to figure out β, while in testing process, we use β estimated in training and make prediction of y.

In order to simplify mathematical analysis, we assume conditionally independent for y_i.

- (Conditional Independence) y_i are conditionally independent given β, i.e.

$$p(y|\beta) = p(y_1, y_2, \cdots, y_N|\beta) = \prod_{i=1}^{N} p(y_i|\beta). \tag{2.5}$$

We observe randomness in label y due to (2.4), ϵ is Gaussian and X, β are deterministic. Thus $y - X\beta$ inherits Gaussian property.

$$\epsilon \sim \mathcal{N}(0, \sigma^2) \Rightarrow y - X\beta \sim \mathcal{N}(0, \sigma^2) \Rightarrow y \sim \mathcal{N}(X\beta, \sigma^2).$$

Then we have

$$p(y|\beta) = \mathcal{N}(y|X\beta, \sigma^2), \tag{2.6}$$

where

- $p(y|\beta)$ denotes the conditional probability of y given parameter β and
- $\mathcal{N}(y|X\beta, \sigma^2)$ is short for y is Gaussian with mean $X\beta$ and variance σ^2.

By assumption of conditional independence, we have

$$p(y|\beta) = p(y_1, y_2, \cdots, y_N|\beta) = \prod_{i=1}^{N} p(y_i|\beta) = \prod_{i=1}^{N} \mathcal{N}(y_i|\beta^\intercal x_i, \sigma^2). \tag{2.7}$$

Define the likelihood function $L(\beta)$,

$$L(\beta) = \prod_{i=1}^{N} \mathcal{N}(y_i|\beta^\intercal x_i, \sigma^2). \tag{2.8}$$

In order to maximize conditional probability $p(y|\beta)$, it leads to maximize $L(\beta)$, which is the same as maximize log likelihood function

$$\begin{aligned}
\ell(\beta) = \log(L(\beta)) &= \log\left(\prod_{i=1}^{N} \mathcal{N}(y_i|\beta^\intercal x_i, \sigma^2)\right) \\
&= \sum_{i=1}^{N} \log\left(\mathcal{N}(y_i|\beta^\intercal x_i, \sigma^2)\right) \\
&= \sum_{i=1}^{N} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp -\frac{(y_i - \beta^\intercal x_i)^2}{2\sigma^2}\right) \\
&= N \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \sum_{i=1}^{N} \frac{(y_i - \beta^\intercal x_i)^2}{2\sigma^2}
\end{aligned}$$

The maximum likelihood solution is

$$\hat{\beta}_{\mathrm{ML}} = \arg\min_{\beta} \ell(\beta) = \arg\min_{\beta} \sum_{i=1}^{N} (y_i - \beta^{\intercal} x_i)^2, \tag{2.9}$$

which is exactly least squares solution of (2.1).

2.3 Logistic Regression

The next model in Data Science we introduce is Logistic Regression (LR) model. We discuss motivation, derivation of objective function from perspective of maximum likelihood and properties of objective function in this section. We follow Shalizi's lecture notes in statistics [40].

2.3.1 Introduction

Regression problem has continuous output variable while classification problem has discrete output variable. For some problems, instead of simply guessing "yes" or "no", we prefer a probability of each instance belongs to some class. Then logistic regression model is needed.

Logistic regression model is perfect for binary classification problem. We want to know the probability p for data belongs to class A, where $1 - p$ is the probability belongs to another class B. In order to estimate p, we starts from simplest model – linear model.

- $p = x^{\intercal}\beta,\ x, \beta \in \mathbb{R}^{d+1}$, where x is the input data concatenated with 1 and β are parameters. The model has issues that Firstly, p must between 0 and 1 but linear functions are bounded. Secondly, in many situations, we empirically see "diminishing returns" – changing p by the same amount requires a bigger change in x when p is already large or small than p is close to $\frac{1}{2}$.

- $\log(p) = x^\intercal\beta,\ x, \beta \in \mathbb{R}^{d+1}$, where x is the input data concatenated with 1 and β are parameters. The problem is that logarithms are unbounded in one direction when $p \in (0, 1)$.
- $\log\left(\frac{p}{1-p}\right) = x^\intercal\beta,\ x, \beta \in \mathbb{R}^{d+1}$, where $\log\left(\frac{p}{1-p}\right)$ is logistic tranformation of p.

Now we have the model

$$\log\left(\frac{p(x)}{1-p(x)}\right) = x^\intercal\beta.$$

Solve for $p(x)$, we have

$$p(x|\beta) = \frac{1}{1+\exp(-x^\intercal\beta)}.$$

Definition 2.3.1. *Sigmoid function*

$$\sigma(t) = \frac{1}{1+\exp(-t)} \qquad \forall t \in \mathbb{R}. \tag{2.10}$$

The plot of sigmoid function is in Figure 2.1.

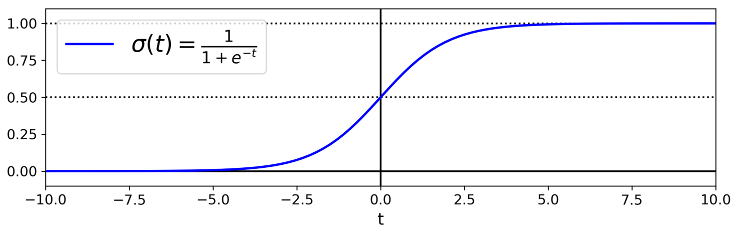


Figure 2.1: Sigmoid Function

Source: Figure from [3]

In order to minimize the mis-classification rate, we should predict $y = 1$ where $p \geq 0.5$ and $y = 0$ when $p < 0.5$, i.e. to guess 1 when $x^\intercal\beta$ is nonnegative and 0 otherwise. The decision boundary is the line (hyperline) where $x^\intercal\beta = 0$. Thus logistic regression is a linear classifier.

2.3.2 Likelihood Function for Logistic Regression

Because logistic regression predicts probabilities, rather than just classes, we can fit it using likelihood. For each training data-point, we have a vector of features x_i and the observed label y_i. The probability of that class is either p if $y_i = 1$ or $1 - p$ if $y_i = 0$. The likelihood is then

$$L(\beta) = \prod_{i=1}^{N} p(x_i|\beta)^{y_i}(1 - p(x_i|\beta))^{1-y_i}. \tag{2.11}$$

Take log on both sides, we have the log likelihood function.

$$l(\beta) = \sum_{i=1}^{N} \left(y_i \log(p(x_i|\beta)) + (1 - y_i)\log(1 - p(x_i|\beta))\right). \tag{2.12}$$

In order to maximize log likelihood function, we need to solve the minimization problem of negative log likelihood function. The mean of negative log likelihood function gives cost function of Logistic Regression.

Definition 2.3.2. *(Cost function of Logistic Regression)*

$$c(\beta) = -\frac{1}{N}\sum_{i=1}^{N} \left(y_i \log(p(x_i|\beta)) + (1 - y_i)\log(1 - p(x_i|\beta))\right). \tag{2.13}$$

Cost function of a single training instance has the form

$$c(\beta) = \begin{cases} -\log(p(x|\beta)) & \text{if} \quad y = 1; \\ -\log(1 - p(x|\beta)) & \text{if} \quad y = 0. \end{cases} \tag{2.14}$$

It is easily to see the cost function of a single training instance is convex. The summation of convex function is also convex.

From Figure 2.2 we can see, if the observed label is 1, p goes to 1, cost function reaches minimum

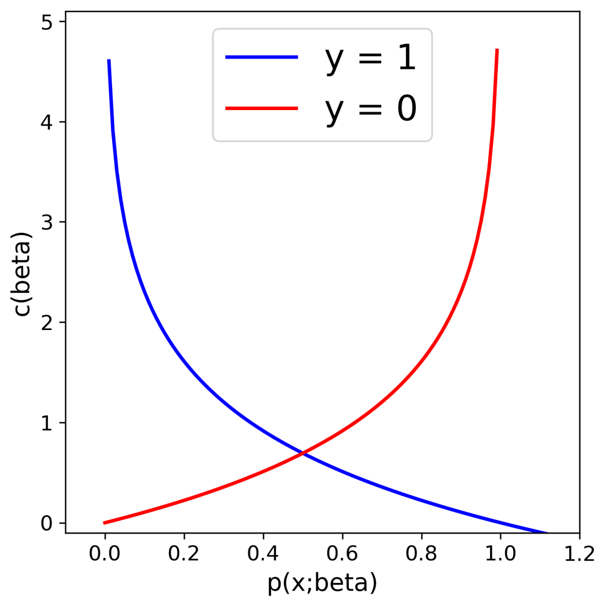


Figure 2.2: LR cost function of single instance

while p goes to 0 cost function goes to ∞. If observed label is 0, p goes to 0 cost function reaches minimum while p goes to 1 cost function goes to ∞. The aim of minimzation of cost function is to push p to 1 if $y = 1$ and push p to 0 if $y = 0$.

Take derivative of log likelihood function we have

$$\begin{aligned}\frac{\partial l}{\partial \beta_j} &= -\sum_{i=1}^{N} \frac{1}{1+\exp(x^{\intercal}\beta)} \exp{(x^{\intercal}\beta)} x_{ij} + \sum_{i=1}^{n} y_i x_{ij} \\ &= \sum_{i=1}^{N} (y_i - p(x_i|\beta)) x_{ij}.\end{aligned}$$

The optimal point is where each component of gradient equal 0. This is a transcendental equation. Thus there is no closed form solution.

2.3.3 More on Logistic Regression Objective Function

Now we provide a simplified version of logistic regression objective function (2.13) presented as an example in [20]. Then we come to derive some properties of Logistic Regression cost function based on simplified formula (2.15).

By (2.13), we can write out the cost function

$$\begin{aligned}
c(\beta) &= -\frac{1}{N}\sum_{i=1}^{N}(y_i \log(p(x_i|\beta)) + (1-y_i)\log(1-p(x_i|\beta))) \\
&= -\frac{1}{N}\sum_{i=1}^{N}(y_i \log(\sigma(x_i^\intercal \beta)) + (1-y_i)\log(1-\sigma(x_i^\intercal \beta))) \\
&= -\frac{1}{N}\sum_{i=1}^{N}(-\log(1+\exp(-x_i^\intercal \beta)) + (1-y_i)(-x_i^\intercal \beta))
\end{aligned}$$

Substitute parameter β by x, data matrix X with A, where $a_i^\intercal$ is row vector of A corresponding to i-th data. Denote $b_i = 1 - y_i$.

We have a simplified version of LR objective function. The probability we estimate is $p_i(x) = \sigma(-a_i^\intercal x)$, where σ is sigmoid function defined in (2.10).

$$f(x) = \frac{1}{N}\sum_{i=1}^{N}[\log(1+\exp(-a_i^\intercal x)) - b_i a_i^\intercal x] \tag{2.15}$$

Theorem 2.3.3. *Cost function of Logistic Regression* (2.15) *is convex.*

Proof. We prove the cost function of Logistic Regression (2.15) is convex by calculating the Hessian matrix of (2.15) and show it is symmetric positive definite (SPD). Following is the computation.

The second term $-b_i a_i^\intercal x$ in (2.15) is linear. When we compute second order derivative, it results in 0. Thus we consider second order derivative of the first term in (2.15).

Let $a = \begin{bmatrix} a_1, \cdots, a_{d+1} \end{bmatrix}^\intercal$.

$$\begin{aligned}
\frac{\partial}{\partial x_k}(\log(1+\exp(-a^\intercal x))) &= \frac{\partial}{\partial x_k}(\log(1+\exp(-\sum_j x_j a_j))) \\
&= \frac{\exp(-\sum_j x_j a_j)}{1+\exp(-\sum_j x_j a_j)} a_k \\
&= -\left(1 - \frac{1}{1+\exp(-\sum_j x_j a_j)}\right) a_k \\
\frac{\partial^2}{\partial x_k^2}(\log(1+\exp(-a^\intercal x))) &= \frac{\exp(-\sum_j x_j a_j)}{(1+\exp(-\sum_j x_j a_j))^2} a_k^2 \\
&= \left(\frac{1}{1+\exp(-\sum_j x_j a_j)} - \frac{1}{(1+\exp(-\sum_j x_j a_j))^2}\right) a_k^2 \\
\frac{\partial^2}{\partial x_k \partial x_l}(\log(1+\exp(-a^\intercal x))) &= \left(\frac{1}{1+\exp(-\sum_j x_j a_j)} - \frac{1}{(1+\exp(-\sum_j x_j a_j))^2}\right) a_k a_l
\end{aligned}$$

Thus the Hessian matrix of Logistic Regression cost function is

$$H_{kl} = \frac{1}{n}\sum_{i=1}^{n} \frac{\exp(-{a_i}^\intercal x)}{(1+\exp(-{a_i}^\intercal x))^2} a_{ik} a_{il}.$$

In matrix form,

$$\begin{aligned} H &= \frac{1}{n}\sum_{i=1}^{n} \frac{\exp(-a_i^\intercal x)}{(1+\exp(-a_i^\intercal x))^2} a_i a_i^\intercal \\ &= \frac{1}{n}\sum_{i=1}^{n} \left(\frac{\exp(-a_i^\intercal x)}{(1+\exp(-a_i^\intercal x))^2} a_i a_i^\intercal \right) \end{aligned}$$

H is symmetric semi-positive definite since (SSPD) it is the linear combination of rank 1 matrices $a_i a_i^\intercal$ with positive coefficients. As long as the set of row vector $\{a_i\}_{i=1}^{N}$ is linear independent. H is symmetric positive definite (SPD). Since the Hessian matrix of function f is SPD, f is convex. $\square$

To conclude, we have a brief summary of Linear Regression and Logistic Regression Model in Table 2.1.

Table 2.1: Comparisons of Linear Regression and Logistic Regression

	Linear Regression	Logistic Regression
Problems	Regression	Binary Classification
Input	Data with continuous target values	Data with discrete labels
Output	continuous	categorical with probability
Model	$y = X\beta + \epsilon$, $\epsilon \sim N(0, \sigma^2)$	$p(X;\beta) = \frac{1}{1+\exp(-X\beta)}$
Objective Function	$f(\beta) = \frac{1}{N}\|y - X\beta\|^2$	$f(\beta) = \frac{1}{N}\sum_{i=1}^{N}[\log(1+\exp(x_i^\intercal\beta)) - y_i x_i^\intercal \beta]$
Properties	Linear Convex Smooth	Nonlinear Convex Smooth
Popular Solver	Direct Solver	SGD

2.4 Non-constraint Convex and Smooth Minimization Problem

Linear regression and Logistic regression are two specific minimization problem with convex and smooth objective function. Now we come to general non-constraint minimization problem setting.

Consider non-constraint minimization problem

$$\min_{x \in \mathcal{V}} f(x), \tag{2.16}$$

where f is a smooth and convex function and its derivative is Lipschitz continuous with constant L and $\mathcal{V}$ is a Hilbert space. In practice, $\mathcal{V} = \mathbb{R}^N$ but might be assigned with an inner products other than the standard l^2 inner product. Solving minimization problem (1.2) is a central task with wide applications in fields of scientific computing, machine learning and data science etc.

We consider the minimization problem (1.2) with following setting. The Hilbert space $\mathcal{V}$ is a vector space equipped with an inner product $(\cdot, \cdot)_{\mathcal{V}}$. Although our discussion might be valid in general Hilbert spaces, we restrict ourself to the finite dimensional space and without of loss generality we take $\mathcal{V} = \mathbb{R}^N$.

The standard l^2 dot product is

$$(x, y) = x \cdot y := \sum_{i=1}^{N} x_i y_i. \tag{2.17}$$

But $\mathcal{V}$ could be assigned an inner product $(\cdot, \cdot)_{\mathcal{V}}$ other than l^2 and the norm induced is

$$\|u\|_{\mathcal{V}} = \sqrt{(u, u)_{\mathcal{V}}}, \quad \forall u \in \mathcal{V}. \tag{2.18}$$

In particular, given a symmetric positive definite(SPD) matrix A, a new inner product can be defined with respect to A.

$$(u, v)_A = (Au, v), \ \forall\, u, v \in \mathcal{V}. \tag{2.19}$$

A new norm with respect to A is induced by $A-$inner product.

$$\|u\|_A = \sqrt{(u, u)_A}, \ \forall\, u \in \mathcal{V}. \tag{2.20}$$

Let $\mathcal{V}' := \mathcal{L}(\mathcal{V}, \mathbb{R})$ be the linear space of all linear continuous mappings $\mathcal{V} \to \mathbb{R}$, which is called the dual space of $\mathcal{V}$. Define dual norm on $\mathcal{V}'$: $h \in \mathcal{V}' \mapsto \|h\| \in \mathbb{R}^+$, by the formula

$$\|h\|_{\mathcal{V}'} = \sup_{\|x\|_{\mathcal{V}} \leq 1} |h(x)|. \tag{2.21}$$

The duality pair $\langle \cdot, \cdot \rangle$ is defined as

$$\langle h, x \rangle := h(x), \; \forall\, x \in \mathcal{V}, \; h \in \mathcal{V}'. \tag{2.22}$$

Given by 'F. Riesz representation theorem' [24], for any $h \in \mathcal{V}'$ there exists one and only one element $x_h \in \mathcal{V}$ such that the representation formula

$$h(x) = (x, x_h), \; \forall\, x \in \mathcal{V} \tag{2.23}$$

holds. And

$$\|h\|_{\mathcal{V}'} = \|x_h\|_{\mathcal{V}} \tag{2.24}$$

holds.

The objective function $f(x) : \mathcal{V} \mapsto \mathbb{R}$ may satisfy part of the following assumptions:

- (LC) The first order derivative of f is Lipschitz continuous with Lipschitz constant L, i.e.,

$$\|\nabla f(x) - \nabla f(y)\|_{\mathcal{V}'} \leq L\|x - y\|_{\mathcal{V}}, \quad \forall\, x,\, y \in \mathcal{V}.$$

- (C) f is convex, i.e.,

$$f(x) \geq f(y) + \langle \nabla f(y), (x - y) \rangle, \quad \forall\, x,\, y \in \mathcal{V}.$$

- (SC) f is strongly convex with strong convexity constant $\mu > 0$, i.e.,

$$\langle \nabla f(x) - \nabla f(y), x - y \rangle \geq \mu \|x - y\|_{\mathcal{V}}^2, \quad \forall\, x,\, y \in \mathcal{V}.$$

Chapter 3

Existing Methods for Least Squares Problems

Consider the linear equation (1.1) in Chapter 1. Suppose A is full rank. The solution of (1.1) exists but may be not unique. Then we come to find the solution of normal equation (2.2), which is also the solution of least squares problem (2.1). There are various methods for solving the least squares problems (2.1). Two main approaches can be applied including direct methods and iterative methods. For direct methods, a decomposition of $A^\intercal A$ will be implemented like Gauss Elimination, QR, SVD or Cholesky factorization.

3.1 Direct Methods

3.1.1 Gauss Elimination

In order to solve (1.1) by Gauss Elimination, we start from augmented matrix equation of linear system.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} & b_m. \end{bmatrix}$$

Perform elementary row operations to transform the augmented matrix into upper triangular form. The elementary row operations include row switching, row multiplication and row addition. Since linear vector space is closed under linear combination, the three fundamental row operations would not change the solution of linear system. The complexity for solving $A^\intercal A$ by Gauss Elimination is $O(n^3)$ with matrix multiplication $O(mn^2)$.

3.1.2 Cholesky Decomposition

Cholesky decomposition is a decomposition of a symmetric, positive-definite (SPD) matrix into the product of a lower triangular matrix and its transpose, i.e.

$$A = LL^\intercal, \tag{3.1}$$

where L is a lower triangular matrix with real positive diagonal entries and $L^\intercal$ is transpose of L.

Another form which is closely related to Cholesky decomposition is LDL-decomposition.

$$A = LDL^{\intercal}, \tag{3.2}$$

where L is lower-triangular whose diagonal elements equal 1. Cholesky decomposition and LDL-decomposition are mainly used for solve linear equations $Ax = b$, where A is SPD. Thus we can apply them to normal equation (2.2). If we have Cholesky decomposition $A^{\intercal}A = LL^{\intercal}$, (2.2) can be solved by

1. Solve $Ly = Ab$ with forward substitution;
2. Solve $L^{\intercal}x = y$ with backward substitution.

The algorithm to achieve Cholesky decomposition is a modified version of Gauss Elimination. Let $i := 1$ and $A^{(1)} := A$. At step i, matrix $A^{(i)}$ has the form

$$A^{(i)} = \begin{bmatrix} I_{i-1} & 0 & 0 \\ 0 & a_{ii} & b_i^{\intercal} \\ 0 & b_i & B^{(i)} \end{bmatrix}$$

The lower triangular matrix can be defined as

$$L_i := \begin{bmatrix} I_{i-1} & 0 & 0 \\ 0 & \sqrt{a_{ii}} & 0 \\ 0 & \frac{1}{\sqrt{a_{ii}}}b_i & I_{n-i} \end{bmatrix}$$

Then the update form holds

$$A^{(i)} = L_i A^{(i+1)} L^{\intercal}.$$

The algorithm will terminate when $i = n$, matrix multiplication from L_1 to L_n gives the lower triangular matrix L, i.e.

$$L = L_1 L_2 \cdots L_n.$$

Complexity of Cholesky decomposition is $O(\frac{n^3}{3})$.

Apply Cholesky decomposition to solve normal equation (2.2), the complexity is $O(mn^2)$ for matrix multiplication $A^\intercal A$, $O(n^3)$ for Cholesky decomposition, and $O(n^2)$ for forward and backward substitution. Thus the total complexity is $O(mn^2 + n^3 + n^2)$.

3.1.3 QR Decomposition

A QR decomposition of matrix A is of the form

$$A = QR,$$

where Q is an orthonormal matrix of size $m \times m$ and R is an upper triangular matrix of size $m \times n$. In the least squares case when $m \geq n$, the bottom $(m-n)$ rows of an $m \times n$ upper triangular matrix consist entirely of zeros. We have a "thin" version of QR decomposition.

$$A = QR = Q \begin{bmatrix} R_1 \\ 0 \end{bmatrix} = \begin{bmatrix} Q_1, Q_2 \end{bmatrix} \begin{bmatrix} R_1 \\ 0 \end{bmatrix} = Q_1 R_1,$$

where R_1 is an $n \times n$ upper triangular matrix, 0 is an $(m-n) \times n$ zero matrix, Q_1 is $m \times n$, Q_2 is $m \times (m-n)$, and the columns of Q_1 and Q_2 form an orthogonal set. The complexity for QR decomposition of matrix A is $O(mn^2)$. After we get QR decomposition of matrix A, the linear system (1.1) can be solved via

$$y = R^{-1}b;$$
$$x = Q^{-1}y.$$

Since Q is orhtonormal, $Q^{-1} = Q^{\intercal}$. And the inverse of upper triangular matrix R can be solved in linear complexity $O(n)$. Thus the total time complexity of solving linear system would be $O(mn^2 + mn + n)$ which dominated by QR decomposition.

3.1.4 SVD Decomposition

In order to achieve SVD decomposition, we are actually seeking information of singular values and singular vectors of matrix A.

Compute

$$A = U\Sigma V^T,$$

where

- U is a $m \times m$ orthogonal matrix.
- $\Sigma = \begin{bmatrix} \sigma_1 & \cdots & 0 \\ 0 & \cdots & 0 \\ 0 & \cdots & \sigma_m \end{bmatrix}$ is a diagonal $m \times m$ matrix with non-negative real numbers σ_i on the diagonal.
- V is a $m \times n$ orthogonal matrix.

The diagonal entries are σ_i of Σ are known as the singular values of A. Complexity of SVD decomposition is $O(mn^2)$. After SVD decomposition, least squares problem can be solved by

$$\begin{aligned} y &= U^\intercal b; \\ z &= \Sigma^{-1} y; \\ x &= Vz. \end{aligned}$$

Since U and V are orthonormal matrices, computing inverse equal multiplying by transpose. Σ is a diagonal matrix. Inverse of Σ is the diagonal matrix with diagonal elements $\frac{1}{\sigma_i},\ i = 1, \cdots, n$. Then equation (1.1) can be solved in $O(mn)$. The total complexity is $O(mn^2 + mn)$ with SVD decomposition dominating complexity.

3.2 Iterative Methods

3.2.1 Residual Correction Method

For symmetric positive definite (SPD) matrix A, we have a residual-correction method for solving (1.1). If A is not symmetric, we can apply residual correction method to normal equation (2.2) as long as $A^\intercal A$ is SPD. For the type of residual-correction method, we follow closely on Xu's papers [49, 50] and Chen's notes [10], where discussion of implementation and convergence analysis can be checked. The method is generated by three steps:

1. form the residual $r = b - Ax^k$;
2. compute correction $e = Br$ where B is an approximation of A^{-1};
3. update with correction $x^{k+1} = x^k + e$.

Decompose SPD A into sum of three matrices.

$$A = L + D + U,$$

where D is diagonal, L is lower triangular, U is upper triangular and $L = U^{\intercal}$. There are various choice of B. Here we lists some of them.

- Richardson $B_R = \alpha I$;
- Jacobi $B_J = D^{-1}$;
- Forward Gauss-Seidel $B_{\text{FGS}} = (D + L)^{-1}$;
- Backward Gauss-Seidel $B_{\text{BGS}} = (D + U)^{-1}$;
- Symmetric Gauss-Seidel $B_{\text{SGS}} = (D + U)^{-1} D (D + L)^{-1}$;
- Successive Over Relaxation (SOR) $B_{\text{SOR}} = \alpha(D + \alpha L)^{-1}$.

The update for each component of x^k is

$$[x^{k+1}]_i = \frac{1}{a_{ii}}(b_i - \sum_{j=1}^{i-1} a_{ij}[x^k]_j - \sum_{j=i+1}^{n} a_{ij}[x^k]_j). \tag{3.3}$$

3.2.2 Conjugate Gradient Method

Conjugate Gradient (CG) method is an iterative method to solve symmetric and positive definite (SPD) systems. It is applicable to sparse systems that are too large to be solved by a direct solver. In general for Krylov subspace methods, only matrix-vector product instead of matrix-matrix product is required and thus save the storage and reduce the complexity provided the method convergences fast. In the overdetermined case, if CG is applied to the normal equation (2.2), only the matrix-vector multiplication Au and $A^{\intercal}v$ is needed which cost $O(nnz(A))$ operations. Here

$nnz(\cdot)$ is the number of nonzero entries of a matrix. In order to achieve the accuracy ϵ, CG needs $O(|\log(\epsilon)|\sqrt{\kappa(A^{\intercal}A)}) = O(|\log(\epsilon)|\kappa(A))$ steps, where the condition number of matrix A is defined by

$$\kappa(A) = \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)} \tag{3.4}$$

with $\sigma_{\max}(A), \sigma_{\min}(A)$ being the maximal and minimal singular values of A respectively. Therefore the complexity for CG applied to (2.2) for dense matrices is $O(mn\kappa(A)|\log(\epsilon)|)$ while for space matrices is $O(nnz(A)\kappa(A)|\log(\epsilon)|)$. Tailored implementations of CG to normal equations (2.2) include CGLS [7] and LSQR [36].

CGLS method is mathematically equivalent to applying CG to normal equations (2.2) without actually forming the product matrix $A^{\intercal}A$ [7].

LSQR is another implementation of a conjugate-gradient type method for solving sparse linear equations and sparse least-squares problems, which is based on the Golub-Kahan bidiagonalization process. It is algebraically equivalent to applying CGLS, but is likely to obtain more accurate solutions in fewer iterations especially if A is ill-conditioned at the expense of more storage and more complexity per iteration [36].

3.2.3 Kaczmarz Method

For consistent systems, Kaczmarz method can be applied. Recall that a linear system is called consistent if there is at least one solution, i.e. $b \in \mathrm{range(A)}$ in (1.1). Kaczmarz method is to project approximation x_k onto the hyperplane $a_i x = b_i$ where a_i is the i-th row of matrix A.

Kaczmarz method was first discovered by the Polish mathematician Stefan Kaczmarz, and was discovered in the field of image reconstruction from projections by Richard Gordon, Robert Bender and Cabor Herman in 1970, where it is called the Algebraic Reconstruction Technique(ART).

The computational advantage of Kaczmarz methods relative to other methods depends on the system being sparse. It has been demonstrated to be superior, in some biomedical imaging applications, to other methods such as filtered back projection method.

It has many applications ranging from computed tomography (CT) to signal processing. It can be obtained also by applying to the hyperplanes, described by the linear system, the method of successive projections onto convex sets (POCS) [2].

Here is an intuitive idea about how to understand Kaczmarz Method. To solve a linear system $Ax = b$, we can view to find a linear combination of columns of A such that

$$x_1a^1 + x_2a^2 + \cdots + x_na^n = b,$$

where $a^i, 1 \leq i \leq n$ are the columns vectors of matrix A, i.e.

$$A = \begin{bmatrix} a^1, a^2, \cdots a^n \end{bmatrix}.$$

Then we try to understand $Ax = b$ in another way. Suppose $a_1, \cdots, a_m$ are the row vectors of matrix A

$$A = \begin{bmatrix} a_1^{\intercal} \\ a_2^{\intercal} \\ \cdots \\ a_m^{\intercal} \end{bmatrix}.$$

and $x \in \mathbb{R}^m$ is the solution we want to find. Then $Ax = b$ can be written as

$$\begin{cases} a_1^\intercal x = b_1 \\ a_2^\intercal x = b_2 \\ \dots \\ a_m^\intercal x = b_m. \end{cases} \tag{3.5}$$

Since each equation $a_i^\intercal x = b_i, 1 \leq i \leq m$ represents a hyperplane, the system (3.5) shows that solution x is a point in the intersection of all the hyperplanes if the system is consistent.

3.2.4 Orthogonal Projection Operator

We define the orthogonal projection of x onto the hyperplane given by $c^\intercal x = d$,

$$\mathcal{P}_{c,d}(\tilde{x}) = x - \frac{c}{\|c\|^2}(c^\intercal \tilde{x} - d).$$

This operator does an orthogonal projection of the current estimate vector $\tilde{x}$ onto the hyperplane $c^\intercal x = d$.

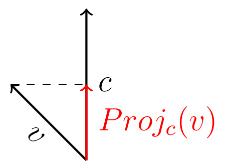
Suppose x is any point in $\mathbb{R}^n$. x_0 is any point on the plane, i.e., $(c, x_0) = c^\intercal x_0 = d$.

Step 1. Do the inner product of c and v. Let $v = \tilde{x} - x_0$.

$$c^\intercal \tilde{x} - d = c^\intercal \tilde{x} - c^\intercal x_0 = c^\intercal (\tilde{x} - x_0) = (c, v).$$

Step 2. Project v onto the vector c.

$$\mathcal{P}_c(v) = \frac{c}{\|c\|^2}(c, v) = \frac{c}{\|c\|^2}(c^\intercal \tilde{x} - d)$$



Step 3. Orthogonally project point $\tilde{x}$ onto the plane $c^\intercal x = d$.

$$\mathcal{P}_{c,d}(\tilde{x}) = \tilde{x} - \mathcal{P}_c(v) = \tilde{x} - \frac{c}{\|c\|^2}(c, v) = \tilde{x} - \frac{c}{\|c\|^2}(c^\intercal \tilde{x} - d).$$

Oswald and Zhou [35] obtained the convergence rate of cyclic Kaczmarz method

$$\|x_{\text{opt}} - x_k\|_2^2 \leq \left[1 - \frac{1}{(\log(n)+1)\kappa(A^\intercal D^{-1} A)}\right]^k \|x_{\text{opt}} - x_0\|_2^2,$$

where x_{opt} is the least squares solution, x_k is the kth iterate, x_0 is the initial guess and D is an $m \times m$ diagonal matrix which induces a row scaling and k denotes the number of sweeps. In order to achieve accuracy ϵ, the total complexity of Kaczmarz method is $O(mn\log(n)\kappa(A^\intercal D^{-1}A)|\log(\epsilon)|)$.

The advantage of iterative methods is that they utilize the sparsity since in each iteration only matrix-vector multiplications are calculated. For example, to achieve accuracy ϵ the complexity of CG is $O(nnz(A)\kappa(A)|\log(\epsilon)|)$ when A is sparse. For Kaczmarz method, the complexity also reduces since the cost of each projection is less than $O(n)$ depending on the sparsity of A. As the iteration steps of both CG and Kaczmarz for consistent systems depend crucially on the condition number $\kappa(A)$, they are slow if A is ill-conditioned, i.e. $\kappa(A) \gg 1$. Preconditioner can be used to improve the condition number and in turn accelerate the convergence. One way to construct effective preconditioners is to use random sampling and random transformation, which will be discussed in later section.

3.3 Randomized Methods

There are several approaches to accelerate traditional least squares solvers via randomization. The main idea is either use random sampling or random projection to reduce the size of the original matrix, e.g. the randomized Kaczmarz method [41] and randomized fast solvers in [4, 16], or construct preconditioners by random sampling to reduce the condition number which enable to apply PCG [37] or LSQR [4] of the preconditioned system.

3.3.1 Randomized Kaczmarz Methods

Randomized Kaczmarz method does the orthogonal projection randomly. There are different ways to implement random choice. One way is to choose each row with equal probability. Another one is to choose with probability proportional to the norm of each row. Let p_i denote the probability of $i-th$ row being chosen, then

$$p_i = \frac{\|a_i\|_2^2}{\|A\|_F^2}, 1 \leq i \leq m.$$

The way to implement random sampling is to use a cumulative vector F. Define F as

$$F = \begin{bmatrix} p_1 \\ p_1 + p_2 \\ \cdots \\ p_1 + \cdots + p_m \end{bmatrix} = \begin{bmatrix} p_1 \\ p_1 + p_2 \\ \cdots \\ 1 \end{bmatrix}.$$

Randomly generate a number $t \in [0, 1]$. Compare t with each component of F, find the first index i such $F(i) \geq t$. The corresponding a_i is the row to be chosen.

The convergence rate of the cyclic Kaczmarz method highly depends on the ordering of rows of A. In order to achieve a faster convergence which is independent of ordering of rows, choosing rows at random is a good strategy. For consistent systems, i.e. $b \in \text{Range}(A)$, Strohmer and Vershynin [41] proposed a randomized Kaczmarz (RK) method which selects the hyperplane to do projection via the probability proportional to $\|a_i\|_2^2$, and proved its exponential convergence in expectation, i.e.

$$\mathbb{E}(\|x_k - x_{\text{opt}}\|_2^2) \leq (1 - \kappa_F(A)^{-2})^k \|x_0 - x_{\text{opt}}\|_2^2,$$

where x_k is the k-th iteration, x_0 is the initial guess, x_{opt} is the least squares solution and $\kappa_F(A) = \|A\|_F^2 \|(A^\intercal A)^{-1}\|_2^2$ is a scaled condition number. To achieve the accuracy ϵ, the expected iteration steps $O(\kappa_F^2(A)|\log(\epsilon)|)$ and the total expected complexity is $O(n\kappa_F^2(A)|\log(\epsilon)|)$.

For consistent systems, the randomized Kaczmarz method converges with expected exponential rate independent of the number of equations in the system. Indeed, the solver does not need to know the whole system but only a $O(n \log n)$ rows as the system is assumed to be consistent [41]. Thus it outperforms some traditional methods like CG on general extremely overdetermined system. The main limitation of RK is its inability of handling inconsistent systems. For instance, to solve $Ax = b$, where $b = y + w$, with $y = b_{R(A)}$ is the projection of b onto range of A and $w = b_{R(A)^\perp}$, the randomized Kaczmarz method is effective when least squares estimation is effective, i.e. the least squares error $\|w\|_2$ is negligible [51]. Extension of randomized Kaczmarz methods to inconsistent systems can be found in [29, 51, 47].

3.3.2 Fast Least Squares Solvers by Random Transformations

Drineas, Mahoney, Muthukrishnan and Sarlos [15] developed two randomized algorithms for least square problems. Instead of solving the original least squares problem $\|Ax - b\|_2^2$, they solve an approximate least squares problem $\|XAx - Xb\|_2^2$, where $X = SHD$ for randomized sampling or $X = THD$ for randomized projection. The operator HD is the randomized Hadamard transformation which aims to spread out the elements of A and S is the uniform sampling matrix and T is the randomized projection matrix aiming to reduce the size of the original problem. The complexity of Hadamard transformation is $O(m\log(m))$ and the complexity of traditional methods to the approximated least squares problem is $O(rn^2)$ with the sample size $r = O(n/\epsilon)$ is chosen so that XA is full rank but $r \ll m$. The complexity reduce to $O(mn\log(n/\epsilon) + n^3/\epsilon) \ll O(mn^2)$ provided ϵ is not too small [16].

Rokhlin and Tygert [37] proposed a fast randomized algorithm for overdetermined linear least squares regression. They constructed subsampled randomized Fourier transform (SRFT) matrix T of size $r \times m$ and then apply pivoted QR decomposition to $TA = QR\Pi$, with a $r \times n$ orthonormal matrix Q, an upper-triangular $n \times n$ matrix R and an $n \times n$ permutation matrix Π. Then apply PCG with the right preconditioning matrix $P = R\Pi$, i.e. to minimize the approximated system $\|AP^{-1}y - b\|$.

According to theory developed in [37], $\kappa(AP^{-1}) = \kappa(TU)$ where the columns of U are left singular vectors of A. The condition number of TU can be controlled by the number of rows of T, i.e. r. In practice, $\kappa(TU) \leq 3$ when $r = 4n$. In this method, it converges fast since $\kappa(AP^{-1})$ is much smaller than $\kappa(A)$ but needs one QR decomposition which is $O(n^2r)$. The total theoretical complexity is $O((\log(r) + \kappa(AP^{-1})|\log(\epsilon)|mn) + O(n^2r)$.

Avron, Maymounkov and Toledo [4] developed an algorithm called BLENDENPIK, which super-charges LAPACK's dense least-squares solver. They introduce the concept of coherence number $\mu(A)$, which is the maximum of squared norm of rows of Q, where the columns of Q are a set of orthonormal bases of range of A. They find that the uniform sampling will work if the coherence number of the matrix is small and apply the row mixing to reduce $\mu(A)$ if it is large [4]. The crucial observation is that a unitary transformation preserves the condition number but changes the coherence number $\mu(A)$. After prepossessing with row mixing, the coherence number of the matrix $\mu(A)$ is reduced and uniform sampling can be applied to get a sampled matrix A_s with only $O(n\log(n))$ rows. Then they decomposed the sampled matrix $A_s = QR$ and used LSQR method to solve the original system $Ax = b$ with preconditioner R^{-1} [4]. The complexity of row mixing is $O(mn\log(m))$ and of QR decomposition of sampled matrix is $O(n^3)$. LSQR applied to the linear system $Ax = b$ with preconditioner R^{-1} costs $O(mn\kappa(AR^{-1})|\log(\epsilon)|)$. To conclude, the total complexity of BLENDENPIK method is $O(mn\log(m) + n^3 + mn\kappa(AR^{-1})|\log(\epsilon)|)$.

3.4 Our Contribution

The common feature of these fast least squares solvers is that they all try to use random sampling or random transformation to get a spectrally equivalent matrices but with considerably small size. Then an efficient preconditioner can be constructed via these sampled matrices. However, the preprocesses of all these methods transform sparse matrices into dense matrices, which cannot take the advantage of sparsity if the original matrix A is sparse.

To conclude, the traditional methods like QR and SVD decomposition need $O(mn^2)$ which is prohibitive when m, n is large. Iterative methods such as CG and Kaczmarz can reduce the complexity if the matrix is well conditioned and failed for the ill conditioned cases. Preconditioner based on randomized row sampling algorithms have been developed but destroy the sparsity. Our contribution is the combination of two aspects: constructing a preconditioner which can keep the sparsity and improving the poor conditioning for highly overdetermined matrix.

Chapter 4

Importance Row Sampling Fast Least Squares Solver

4.1 Row Sampling

4.1.1 Preliminaries and Notation

Before walking into the details of our algorithms, we introduce some notation and concepts we may confront. Suppose A is a matrix of size $m \times n$ with $m \geq n$. Denote $a_1, a_2, \cdots, a_m$ to be the row vectors of A and $a^1, a^2, \cdots, a^n$ the column vectors of A. For any vector v, $\|v\| = (\sum_i v_i^2)^{\frac{1}{2}}$ is the l_2 norm of v and is called norm of v for short. For any matrix A, the spectral norm $\|A\| = \max_{x \neq 0} \|Ax\|/\|x\|$ is the induced matrix norm by vector l_2 norm and the Frobenius norm $\|A\|_F = (\sum_{i,j} a_{ij}^2)^{\frac{1}{2}}$.

Definition 4.1.1 (Condition Number)**.** *The condition number $\kappa(A)$ of matrix A is defined as*

$$\kappa(A) = \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)}$$

with $\sigma_{\max}(A), \sigma_{\min}(A)$ are the maximal and minimal singular values of A respectively.

A matrix is said to be singular if the condition number is infinite. In our setting, the matrix A is of full rank. Thus the smallest singular value of A is nonzero and the condition number $\kappa(A) < \infty$.

Recall that $\|A\|_F = (\sum \sigma_i^2)^{\frac{1}{2}}$, where σ_i's are all the singular values of matrix A.

Another concept need to mention is the coherence number introduced in [4].

Definition 4.1.2 (Coherence Number [4])**.** *Let A be an $m \times n$ full rank matrix, and let U be an $m \times n$ matrix whose columns form an orthonormal basis of the column space of A. The coherence of A is defined as*

$$\mu(A) = \max_{1 \le i \le m} \|u_i\|_2^2,$$

where u_i is the ith row of matrix U.

Obviously, the coherence number of a matrix is always between n/m and 1. Matrices with small coherence numbers are called incoherent [4], for example, the Gaussian matrix X which every element is an independent number generated by standard normal distribution. One example of semicoherent matrices is the one with large coherent number but only half rows have a large norm in the orthogonal factor, for example,

$$Y_{m\times n} = \begin{bmatrix} X_{(m-n/2)\times n/2} & 0 \\ 0 & I_{n/2} \end{bmatrix} \tag{4.1}$$

where $I_{n/2\times n/2}$ is a square identity matrix. Coherent matrices are of large coherent number, for instance,

$$Z_{m\times n} = \begin{bmatrix} I_n \\ 0 \end{bmatrix}, \tag{4.2}$$

where $I_{n\times n}$ is the square identity matrix of size n.

4.1.2 Row Sampling

We present a row sampling algorithm. Given a matrix $A_{m\times n}$ and a probability mass function $\{p_k, k = 1, 2, \cdots, m\}$, which will be called sampling density, randomly choose s rows of A via the given sampling density; see Algorithm 1.

```
Input: $A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^m$, a probability mass function $\{p_k, k = 1, 2, \cdots, m\}$ and a
  sample size $s$.
Output: Sampled matrix $A_s \in \mathbb{R}^{s\times n}$
  for $t = 1 : s$ do
    Pick $i_t \in \{1, 2, \cdots, m\}$ with probability $\Pr\{i_t = k\} = p_k$ in identical and independent
    distributed (i.i.d.) trials.
  end for
Let $S \in \mathbb{R}^{s\times m}$ with $S_{t,i_t} = 1/(sp_{i_t})^{1/2}$, then $A_s = SA$ is a sampling of $A$.
```

Algorithm 1: The row sampling algorithm introduced in [16].

Among various sampling densities, we chose the one proportional to the squared norm of each row:

$$p_k = \frac{\|a_k\|_2^2}{\|A\|_F^2}, k = 1, 2, \cdots, m. \tag{4.3}$$

The naive uniform sampling $p_k = 1/m, k = 1, 2 \cdots, m$ fails when the coherence number of the matrix is large. For example, for the coherent matrix Z defined in (4.2), we have to sample all s rows from the first n rows otherwise we will get a rank deficient matrix, whose probability is $s!/m^s$.

4.1.3 Approximation property of the non-uniform sampling

If we write the normal matrix as the summation of rank 1 matrices

$$A^\intercal A = \sum_{i=1}^{m} a_i^\intercal a_i.$$

Then the approximation obtained by the random sampling is given by

$$A_s^\intercal A_s = \frac{1}{s}\sum_{t=1}^{s}\frac{1}{p_{i_t}}a_{i_t}^\intercal a_{i_t}.$$

It is straightforward to verify that $A_s^\intercal A_s$ is an unbiased estimator for $A^\intercal A$, i.e.

$$\mathbb{E}[A_s^\intercal A_s] = A^\intercal A, \tag{4.4}$$

for any choice of sampling density. The choice (4.3) will minimize the variance in Frobenius norms; see Lemma 1 of Chapter 2 in [27].

More importantly such row sampling density keeps the spectral norm in a small variance with high probability. To show this, we need the following concentration result.

Theorem 4.1.3 (Matrix Bernstein (Theorem 6.1 in [45]))**.** *Let $\{X_k\}$ be a sequence of independent random, self- adjoint matrices with dimension d. Assume*

$$\mathbb{E}[X_k] = 0 \quad \textit{and} \quad \lambda_{\max}(X_k) \le R \quad \textit{almost surely.}$$

Let

$$\sigma^2 = \|\sum_k \mathrm{Var}(X_k)\| = \|\sum_k \mathbb{E}(X_k^2)\|.$$

Then for all $t \ge 0$

$$\Pr\left(\lambda_{max}(\sum_k X_k) \ge t\right) \le d\exp\left(-\frac{t^2/2}{\sigma^2 + Rt/3}\right). \tag{4.5}$$

From the Matrix Bernstein inequality, we can derive the following corollary.

Corollary 4.1.4 (Sum of Rank-1 Matrices)**.** *Let $y_1, y_2, \cdots, y_s$ be i.i.d. random column vectors in $\mathbb{R}^n$ with*

$$\|y_k\| \le M \quad \textit{and} \quad \|\mathbb{E}[y_k y_k^\intercal]\| \le \alpha^2,$$

for $k = 1, 2, \ldots, s$. Then for any $\epsilon \in [0, 1]$

$$\Pr\left(\|\frac{1}{s}\sum_{k=1}^{s} y_k y_k^\intercal - \mathbb{E}[y_1 y_1^\intercal]\| \geq \epsilon \right) \leq 2n \exp\left(-\frac{3s\epsilon^2}{(6\alpha^2 + 2\epsilon)(M^2 + \alpha^2)} \right).$$

Proof. Let $Y_k = y_k y_k^\intercal, A = \mathbb{E}[Y_k]$, and $X_k = (Y_k - A)/s$ for $k = 1, 2, \ldots, s$. Then $\mathbb{E}[X_k] = 0$. We bound the spectral norm of X_k as

$$\lambda_{\max}(X_k) = \|X_k\| \leq \frac{1}{s}(\|Y_k\| + \|A\|) \leq \frac{M^2 + \alpha^2}{s},$$

where we use the fact $\|Y_k\| = \|y_k y_k^\intercal\| = \|y_k\|^2 \leq M^2$. We then compute the variance

$$\begin{aligned}
\|\mathbb{E}[X_k^2]\| &= \|\operatorname{Var}(X_k)\| \\
&= \frac{1}{s^2}\|\operatorname{Var}(Y_k)\| \\
&= \frac{1}{s^2}\|\mathbb{E}[Y_k^2] - A^2\| \\
&\leq \frac{1}{s^2}(\|\mathbb{E}\left[\|y_k\|^2 Y_k\right]\| + \|A^2\|) \\
&\leq \frac{1}{s^2}(\|\|y_k\|\mathbb{E}[[Y_k]] + \|A\|^2) \\
&\leq \frac{\alpha^2(M^2 + \alpha^2)}{s^2}.
\end{aligned}$$

Here we compute $Y_k^2 = y_k y_k^\intercal y_k y_k^\intercal = \|y_k\|^2 Y_k$. Sum over k to get $\sigma^2 \leq \alpha^2(M^2 + \alpha^2)/s$. And

$$\|\sum_{k=1}^{s}\mathbb{E}[X_k^2]\| = s\|\mathbb{E}[X_k^2]\| \leq \frac{\alpha^2(M^2 + \alpha^2)}{s},$$

since $X_k, k = 1, \cdots, s$ are i.i.d..

Plug the bound $R \leq (M^2 + \alpha^2)/s, \sigma^2 \leq \alpha^2(M^2 + \alpha^2)/s$ into inequality (4.5) and rearrange the terms, we get the desired result.

$$\Pr\left(\|\frac{1}{s}\sum_{k=1}^{s} y_k y_k^\intercal - \mathbb{E}[y_1 y_1^\intercal]\| \geq \epsilon\right) \leq 2n \exp\left(-\frac{3s\epsilon^2}{6\alpha^2(M^2+\alpha^2) + 2\epsilon(M^2+\alpha^2)}\right).$$

□

We shall apply this concentration result to our row sampling scheme.

Theorem 4.1.5. *Let A be a matrix with size $m \times n$. Let $C = \frac{2}{3}(6\|A\|^2 + 2\epsilon)(1 - \log_n(\delta/2))$ and $s = C\epsilon^{-2} n \log n$. Assume $\|A\|_F^2 = n$ and A_s is a sampled matrix obtained by Algorithm 1 with sampling density* (4.3)*. Then*

$$\|A_s^\intercal A_s - A^\intercal A\| \leq \epsilon \quad \textit{with probability at least } 1 - \delta. \tag{4.6}$$

Proof. Let y be a random variable taking value $a_i^\intercal/\sqrt{p_i}$ with probability $p_i, 1 \leq i \leq m$. And $y_k, k = 1, 2, \ldots, s$ be i.i.d. copies of y. Then

$$A_s^\intercal A_s = \frac{1}{s}\sum_{k=1}^{s} y_k y_k^\intercal,$$

and

$$\mathbb{E}[y_k y_k^\intercal] = \sum_{i=1}^{m} a_i^\intercal a_i = A^\intercal A.$$

Thus we have the bound

$$\|\mathbb{E}[y_k y_k^\intercal]\| = \|A^\intercal A\| = \lambda_{\max}(A^\intercal A) = \|A\|^2,$$

and the bound

$$\|y_k\| \leq \max_{1\leq i\leq m} \frac{\|a_i\|}{\sqrt{p_i}} = \|A\|_F = \sqrt{n}.$$

By Corollary 4.1.4, for all $0 \le \epsilon \le 1$

$$\Pr\left(\|\frac{1}{s}\sum_{k=1}^{s} y_k y_k^{\intercal} - \mathbb{E}[y_1 y_1^{\intercal}]\| \ge \epsilon\right) \le 2n \exp\left(-\frac{3Cn\log(n)}{(6\|A\|^2 + 2\epsilon)(n + \|A\|^2)}\right).$$

i.e.

$$\begin{aligned} \Pr\left(\|A_s^{\intercal}A_s - A^{\intercal}A\| \ge \epsilon\right) &\le 2n\exp\left(-\frac{3Cn\log(n)}{(6\|A\|^2+2\epsilon)(n+\|A\|^2)}\right) \\ &\le 2n\exp\left(-\frac{3C\log(n)}{2(6\|A\|^2+2\epsilon)}\right) \\ &= 2n^{1-\frac{3C}{2(6\|A\|^2+2\epsilon)}}. \end{aligned}$$

To satisfy $\Pr\left(\|A_s^{\intercal}A_s - A^{\intercal}A\| \ge \epsilon\right) < \delta$, we need

$$C \ge \frac{2}{3}(6\|A\|^2 + 2\epsilon)(1 - \log_n(\delta/2)).$$

$\square$

Remark 4.1.6. Constant C used in Theorem 4.1.5 is not practical since the lower bound of C is quit big. For example, when $\delta = \frac{1}{20}, \epsilon = \frac{1}{2}, \|A\| \le 1$ and $n = 300$, $C\epsilon^{-2}$ should be greater or equal than $\frac{56}{3}(1 + \log_{300}(40)) \approx 30.74$. In practice, $C\epsilon^{-2} = 4$ is good enough to get a reasonable sampling matrix. $\square$

Corollary 4.1.7. *With the same setting in Theorem 4.1.5, the following bound hold with high probability*

$$\lambda_{\min}(A^{\intercal}A) - \epsilon \le \lambda_{\min}(A_s^{\intercal}A_s) \le \lambda_{\max}(A_s^{\intercal}A_s) \le \lambda_{\max}(A^{\intercal}A) + \epsilon.$$

Proof. By the triangle inequality, we immediately get

$$\|A_s^{\intercal}A_s x\| \le \|A^{\intercal}Ax\| + \epsilon\|x\| \le \left(\lambda_{\max}(A^{\intercal}A) + \epsilon\right)\|x\|,$$

which implies the desired inequality as $A_s^\intercal A_s$ is symmetric. The lower bound of $\lambda_{\min}(A_s^\intercal A_s)$ can be proved similarly. $\square$

Notice that the spectrum bound obtained in Corollary will not imply the bound of the preconditioned system $(A_s^\intercal A_s)^{-1}A^\intercal A$.

We call $x \in \mathbb{R}^n$ is of high frequency if the inequality

$$\lambda_{\max}(A^\intercal A)\|x\|^2 \leq C_f(A^\intercal Ax, x), \tag{4.7}$$

holds with a universal constant. Consider the decomposition of x using the eigen-vector bases of $A^\intercal A$. Inequality (4.7) implies x is mainly expanded by eigen-vectors of high frequency. The constant C in (4.7) is introduced to include not only the highest frequency but a range of frequencies comparable to the highest one.

Note that (4.6) only implies A_s captures the high frequency component of a vector x. To apply the sampling theory, we should rescale the matrix A to $A/\|A\|$ such that $\|A/\|A\|\| \leq 1$ and thus with high probability we have

$$\|(A_s - A)x\| \leq \epsilon\sigma_{max}(A)\|x\|.$$

With the property of high frequency vector, we know that

$$\|A_s x\| \leq (1 + C\epsilon)\|Ax\|.$$

For high frequency vectors, we will have

$$(A_s^\intercal A_s x, x) \leq (A^\intercal Ax, x) + \epsilon(x, x) \leq \left[1 + \frac{C_f\epsilon}{\lambda_{\max}(A^\intercal A)}\right](A^\intercal Ax, x),$$

and similarly

$$\left[1 - \frac{C_f \epsilon}{\lambda_{\max}(A^\intercal A)}\right] (A^\intercal A x, x) \leq (A_s^\intercal A_s x, x).$$

This implies $(A_s^\intercal A_s)^{-1}$ is an effective smoother for $A^\intercal A$. Since Gauss-Seidel iteration can smooth out the high frequency very quickly, we apply several symmetric Gauss-Seidel iterations instead of computing $(A_s^\intercal A_s)^{-1}$ in practice.

4.1.4 Sampling Analysis

In Mahoney's lecture notes of Randomized Numerical Linear Algebra, he provided the basic approximation properties of random sampling matrix product CR to AB [27]. We applied the results to our random sampling preconditioner to obtain the approximation of sampled matrix $A_s^\intercal A_s$ to original system $A^\intercal A$.

Unbiased Estimator

Followed by Lemma 1 of Chapter 2 in [27], the lemma below shows that $A_s^\intercal A_s$ is an unbiased estimator for $A^\intercal A$.

Lemma 4.1.8. *Let A_s be chosen using the sampling algorithm 1. We then have*

$$\mathbb{E}[A_s^\intercal A_s] = A^\intercal A, \tag{4.8}$$

$$Var[(A_s^\intercal A_s)_{ij}] = \frac{1}{s} \sum_{k=1}^{n} \frac{1}{p_k} A_{ki}^2 A_{kj}^2 - \frac{1}{r} (A^\intercal A)_{ij}^2. \tag{4.9}$$

4.1.5 Graphs of $A_s^\intercal A_s$ and $A^\intercal A$

To illustrate the approximation of the sampled matrix, we plot the graph of $A^\intercal A$ and $A_s^\intercal A_s$ below. The matrix A is of size $m \times n$, where $m = 9314$ and $n = 100$. The sampled matrix A_s is of size

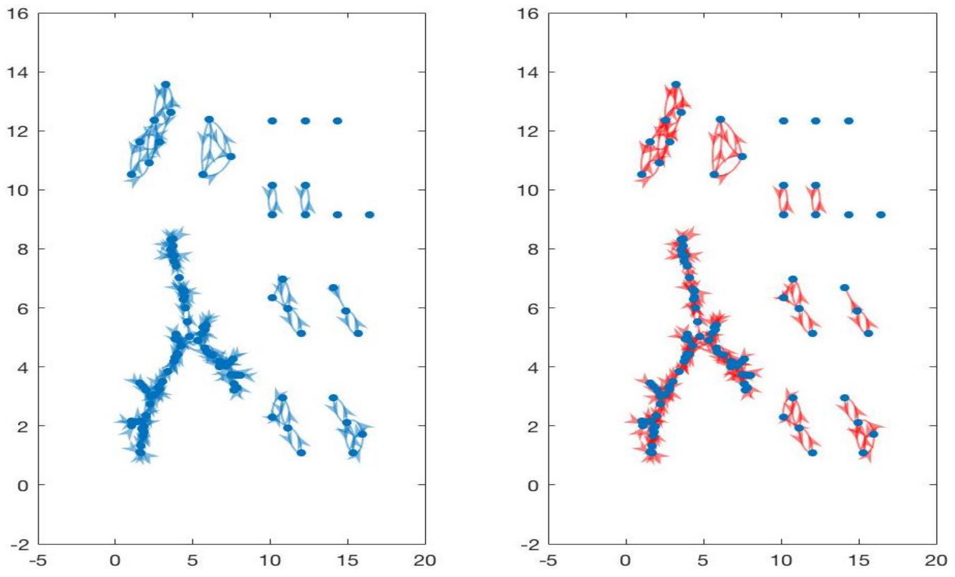


Figure 4.1: Graphs of matrices $A^\intercal A$ (left) and $A_s^\intercal A_s$ (right). The matrix A is of size $m \times n$, where $m = 9314$ and $n = 100$. The sampled matrix A_s is of size $s \times n$ with $s = 1843$ and $n = 100$. The matrix A is rescaled so that the diagonal of $A^\intercal A$ is one. The entries which have small absolute values less than a threshold $\theta = 0.125$ in the matrix $A^\intercal A$ and $A_s^\intercal A_s$ is filtered out and not plot in the graph.

$s \times n$ with $s = 1843$. The matrix A is rescaled so that the diagonal of $A^\intercal A$ is unit. The entries which have small absolute values less than a threshold $\theta = 0.125$ in the matrix $A^\intercal A$ and $A_s^\intercal A_s$ is filtered out and not shown in the graph. Each edge in the graph represents one entry in the matrix and the thickness of edge represent the magnitude respectively. From the figure, we find out that the two graphs are almost identical which means the sampling strategy is able to capture the entries in the normal matrix with a large absolute value.

4.2 PCG with a Preconditioner based on Row Sampling

In this section, we present our algorithm by constructing a fast, efficient and easy to implement randomized row sampling preconditioner and apply PCG to solve the normal equation.

4.2.1 Algorithms

We first normalize the matrix A to make the column vectors have unit length, which enables all diagonal entries of $A^\intercal A$ are one and $\|A\|_F^2 = n$. We then apply the row sampling to get a smaller matrix A_s of size $s \times n$ by randomly choosing $s = O(n \log(n))$ rows of the normalized matrix $A_{m\times n}$ with sampling density $p_i = \|a_i\|_2^2/n$. We build our preconditioner by using a few steps of symmetric Gauss-Seidel (SGS) iteration to solve the approximate problem $A_s^\intercal A_s e = r$. After all, we apply PCG to the normal equation $A^\intercal Ax = A^\intercal b$ with this preconditioner.

Input: $A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^m$, convergence threshold $\epsilon \in (0, 1)$.

Output: approximated $\tilde{x}_{\text{opt}} \in \mathbb{R}^m$.

1. **Normalization:** $A \leftarrow AD^{-1}$, where $D_{jj} = \|a^j\|_2$, with a^j being the jth column vector of A for $1 \le j \le n$.

2. **Sampling:** Sample the row of A to get A_s of size $s \times n$ with $s = 4n \log(n)$ by row sampling Algorithm 1 with probability (4.3).

3. **Preconditioner:** Construct preconditioner $e = Pr$ by solving $A_s^\intercal A_s e = r$ via several symmetric Gauss Seidel iterations; see Algorithm 3.

4. **PCG:** Use PCG to solve $A^\intercal Ax = A^\intercal b$ with the preconditioner constructed in Step 3. Stop when the relative residual is below ϵ.

Algorithm 2: Randomized Sampling Preconditioned PCG

For easy of understanding and completeness, the symmetric Gauss-Seidel method is presented below in Algorithm 3.

```
**Input**: Sampled matrix $A_s \in \mathbb{R}^{s\times n}$, residual $r \in \mathbb{R}^n$, number of symmetric Gauss-Seidel
  iterations $t \in \mathbb{Z}^+$.
**Output**: Correction $e \in \mathbb{R}^n$.
  for $i = 1 : t$ do
      $$e \leftarrow e + B^{-1}(r - A_s^\intercal A_s e)$$
    with $B$ the lower triangular part of $A_s^\intercal A_s$.
  end for
  for $i = 1 : t$ do
      $$e \leftarrow e + (B^\intercal)^{-1}(r - A_s^\intercal A_s e)$$
    with $B^\intercal$ the upper triangular part of $A_s^\intercal A_s$.
  end for
```

Algorithm 3: The Preconditioner using Symmetric Gauss-Seidel Iterations.

4.2.2 Complexity

We compute the complexity of PCG with the randomized sampling preconditioner for both dense matrices and sparse matrices. We use $s = 4n\log(n)$ as the default sample size. Here the factor 4 is chosen to balance the set up time and solver time. Similarly the number of SGS is set as 5 to balance the inner iteration of preconditioner and outer iteration of PCG. For example, in the 'sprand' case with $m = 90000, n = 300$. If the sampling size is decreased to $s = 2n\log(n) \approx 3432$, the PCG iterations increase by about 15%. If the sampling size is increased to $s = 8n\log(n) \approx 13689$, iterations would be 16% less. Thus, we choose $s = 4n\log(n)$ as an optimal point balancing the sampling size and iterations in order to minimize the computation cost.

For dense matrices, in the normalization step, we need $O(mn)$ to calculate the norm of each column $\|a^j\|_2$ and $O(mn)$ for the matrix multiplication AD^{-1}. Sampling costs $O(sn) = O(n^2\log(n))$. The matrix multiplication $A_s^\intercal A_s$ costs $O(sn^2) = O(n^3\log(n))$. The preconditioner in PCG, i.e. several symmetric Gauss Seidel is applied to $A_s^\intercal A_s e = r$ needs $O(n^2)$. Finally, PCG iteration steps $k = O(|\log(\epsilon)|\kappa(PA))$ until reaching tolerance ϵ costs $O(kmn) = O(|\log(\epsilon)|\kappa(PA)mn)$.

Note that since we only do matrix vector multiplication with Ax and $A^\intercal(Ax)$ instead of matrix product $A^\intercal A$, the computation cost for each PCG step is only $O(mn)$ not $O(mn^2)$. Thus the total complexity is $O(|\log(\epsilon)|\kappa(PA)(mn+n^2)) + O(n^3\log(n))$ when A is dense.

Complexity would be reduced significantly when the matrix is sparse. Let $nnz(M)$ be the number of nonzero elements of matrix M. In the normalization step, the cost is reduced to $O(nnz(A))$ for both the column calculation and matrix multiplication AD^{-1}. In the sampling step, the sample size is s and the complexity is reduced to at most $O(nnz(A))$. The matrix product of $A_s^\intercal A_s$ costs between $O(nnz(A_s))$ and $O(n \cdot nnz(A_s))$. The preconditioner costs $O(nnz(A_s^\intercal A_s))$. And $k = O(|\log(\epsilon)|\kappa(PA))$ PCG iterations needed. The total complexity is $O(|\log(\epsilon)|\kappa(PA)(nnz(A) + nnz(A_s^\intercal A_s))) + O(\alpha nnz(A_s))$ for sparse matrices, where $\alpha \in [1, n]$ depends on the sparse pattern of A_s. For sparse matrix A with $nnz(A) \ll mn$, the proposed solver is thus more efficient.

Table 4.1: Complexity of Algorithm 2 for Dense and Sparse Matrices

	Dense Matrix	Sparse Matrix
Normalization	$O(mn)$	$O(nnz(A))$
Sampling	$O(n^2\log(n))$	$O(nnz(A))$
$A_s^\intercal A_s$	$O(n^3\log(n))$	$O(nnz(A_s))$ to $O(n \cdot nnz(A_s))$
Preconditioner	$O(n^2)$	$O(nnz(A_s^\intercal A_s))$
CG iteration	$O(\|\log(\epsilon)\|\kappa(PA)mn)$	$O(\|\log(\epsilon)\|\kappa(PA)nnz(A))$

Theoretically we cannot find a uniform control of the condition number of the preconditioned matrix PA.

4.3 Numerical Results

We shall compare PCG with our randomized sampling preconditioner with CG for the normalized matrix which is equivalent to use PCG for the original matrix with a diagonal preconditioner. The column with prefix 'Setup' in tables is the CPU time for preprocess including sampling and normalization for RS and only normalization for CG. The column ‘Time’ is the CPU time for iterative methods. Thus the sum of these two are the CPU time for the whole algorithms. The column ‘$\kappa(A^\intercal A)$’ lists the condition number of $A^\intercal A$ and $\mu(A)$ is the coherence number with normalized A. We list the coherence number here to emphasize the weighted row sampling works well and robust to the coherence number. It is shown in [4] that the uniform sampling fails when the coherence number of the matrix is large. In our sampling algorithm, we use the sampling density proportional to the squared norm of each row. Tolerance for PCG or CG is set to be 10^{-7} and maximum iteration steps is set to be 500. Notice that iterative methods may end without reaching the tolerance.

As the sampling is random, for each category, we pick up a typical matrix and run our solver 10 times and compute the mean and standard derivation.

We tested several classes of matrices – including well conditioned matrices, ill conditioned matrices, incoherent matrices and coherent matrices; see Table **??**.

Table 4.2: Classes of Matrices

	incoherent	coherent
well conditioned	Gaussian (Example 1)	semi Gaussian (Example 2)
ill conditioned	UDV, sprand (Example 3, 4)	graph Laplacian (Example 5)

4.3.1 Gaussian Matrix

The Gaussian matrix is constructed by MATLAB command

```
A = randn(m,n)
```

with each entry of A is generated independent and identically by a standard normal random variable . The matrix $A^\intercal A$ has a small condition number followed by Bai and Yin’ results [5].

Theorem 1 in [5] claims that if a matrix $X_{p\times n}$ with each element being a random number with mean zero and variance 1 generated independent and identically, and let

$$S = \frac{1}{n} X X^{\intercal}.$$

Then, if $\mathbb{E}|X_{11}|^4 \leq \infty$, as $n \to \infty, p \to \infty, p/n \to y \in (0,1)$,

$$\lim_{n\to\infty} \lambda_{\min}(S) = (1-\sqrt{y})^2 \quad \text{a.s.}$$

$$\lim_{n\to\infty} \lambda_{\max}(S) = (1+\sqrt{y})^2 \quad \text{a.s.}$$

where $\lambda_{\min}$ and $\lambda_{\max}$ are the smallest and largest eigenvalues of S respectively.

In our case, $\frac{1}{m}A^{\intercal}A$ is the sample covariance matrix.

More precisely by Theorem 2 in [5], the limit of condition number of $A^{\intercal}A$ can be calculated as

$$\kappa(A^{\intercal}A) = \frac{\lambda_{\max}(A^{\intercal}A)}{\lambda_{\min}(A^{\intercal}A)} \to \frac{m(1+\sqrt{n/m})^2}{m(1-\sqrt{n/m})^2} = \frac{\sqrt{m}+\sqrt{n}}{\sqrt{m}-\sqrt{n}}.$$

Thus $A^{\intercal}A$ is well conditioned as long as the matrix size n, m is large enough, and A has a rectangular shape, i.e. $m \gg n$. When $m = n^2$, $\lim_{n\to\infty} \kappa(A^{\intercal}A) = \frac{(\sqrt{n}+1)^2}{(\sqrt{n}-1)^2}$ almost surely.

Since each element is generated independent and identically, the Q factor of A's QR decomposition has evenly distributed magnitude in each row. Thus the coherence number $\mu(A)$ of Gaussian matrix is also small. In summary the Gaussian matrix belongs to the category– 'well conditioned and incoherent matrices'.

Table 4.3: Gaussian Matrix: Residual and Iteration Steps

n	m	$nnz(A)$	$\kappa(A^\intercal A)$	$\mu(A)$	Residual.CG	Iter.CG	Residual.RS	Iter.RS
109	3000	11881	8.39	0.05	5.03e-08	10	7.00e-08	11
141	5000	19881	8.30	0.01	8.60e-08	9	5.19e-08	11
200	10000	40000	7.70	0.02	1.70e-08	9	4.42e-08	11
282	20000	79524	7.22	0.05	4.03e-08	8	2.70e-08	11
400	40000	160000	7.40	0.09	9.62e-08	7	2.64e-08	11

Table 4.4: Gaussian Matrix: Elapsed CPU Time

n	m	Time.CG	Setup.CG	Sum.CG	Time.RS	Setup.RS	Sum.RS
109	3000	2.56e-03	2.32e-03	4.88e-03	3.33e-03	6.33e-03	9.66e-03
141	5000	5.34e-03	5.56e-03	1.09e-02	8.32e-03	1.30e-02	2.13e-02
200	10000	1.90e-02	1.95e-02	3.84e-02	2.51e-02	4.05e-02	6.56e-02
282	20000	3.46e-02	4.34e-02	7.80e-02	4.09e-02	7.31e-02	1.14e-01
400	40000	1.00e-01	1.26e-01	2.27e-01	1.43e-01	2.06e-01	3.48e-01

Table 4.5: Gaussian Matrix: Mean and Sample Standard Deviation

n	m	Iter.Mean	Iter.Std	Time.Mean	Time.Std	Setup.Mean	Setup.Std
109	3000	11	0	3.37e-03	5.39e-04	5.64e-03	9.15e-04
141	5000	11	0	1.0e-02	1.58e-03	1.40e-02	1.80e-03
200	10000	11	0	1.47e-02	2.62e-03	2.56e-02	4.18e-03
282	20000	11	0	4.04e-02	3.59e-03	7.25e-02	7.14e-03
400	40000	10.9	0.31	1.40e-01	5.94e-03	2.12e-01	1.42e-02

4.3.2 'Semi Gaussian' Matrix

The 'semi Gaussian' matrix used in [4] has the following block structure. The left upper block B is a Gaussian matrix of size $(m - n/2) \times n/2$ and the right lower block $I_{n/2}$ is an identity matrix of size $n/2 \times n/2$.

$$A_{m\times n} = \begin{bmatrix} B & 0 \\ 0 & I_{n/2} \end{bmatrix}.$$

It belongs to the category– 'well conditioned and coherent matrices'. For such 'semi Gaussian' matrices, the coherence number $\mu(A) = 1$. It is shown in [4] that the uniform sampling fails for this example.

They are also well conditioned since

$$A^\intercal A = \begin{bmatrix} B^\intercal & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} B & 0 \\ 0 & I \end{bmatrix} = \begin{bmatrix} B^\intercal B & 0 \\ 0 & I \end{bmatrix},$$

and when $\lambda_{\max}(B^\intercal B) \geq 1$

$$\kappa(A^\intercal A) \leq \kappa(B^\intercal B).$$

The Gaussian matrix B is well conditioned by the analysis in previous section 4.3.1. So is A.

Table 4.6: 'Semi Gaussian' Matrix: Residual and Iteration Steps

n	m	$nnz(A)$	$\kappa(A^\intercal A)$	$\mu(A)$	Residual.CG	Iter.CG	Residual.RS	Iter.RS
62	1000	992	1.99e+03	1	2.25e-08	9	6.85e-08	11
108	3000	2970	5.82e+03	1	6.87e-08	8	3.70e-08	12
140	5000	4970	9.80e+03	1	2.43e-08	8	6.09e-08	11
200	10000	10100	1.95e+04	1	6.48e-08	7	6.79e-08	11
282	20000	20022	3.84e+04	1	2.28e-08	7	7.37e-08	11

Table 4.7: 'Semi Gaussian' Matrix: Elapsed CPU Time

n	m	Time.CG	Setup.CG	Sum.CG	Time.RS	Setup.RS	Sum.RS
62	1000	7.03e-04	3.96e-04	1.10e-03	1.78e-03	2.16e-03	3.94e-03
108	3000	2.31e-03	2.95e-03	5.26e-03	4.03e-03	7.25e-03	1.13e-02
140	5000	5.14e-03	6.31e-03	1.14e-02	9.95e-03	1.49e-02	2.49e-02
200	10000	1.54e-02	1.77e-02	3.31e-02	2.47e-02	3.84e-02	6.31e-02
282	20000	3.19e-02	4.28e-02	7.47e-02	4.36e-02	7.83e-02	1.22e-01

Table 4.8: 'Semi Gaussian' Matrix: Mean and Sample Standard Deviation

n	m	Iter.Mean	Iter.Std	Time.Mean	Time.Std	Setup.Mean	Setup.Std
62	1000	11.7	0.68	1.18e-03	7.75e-05	1.35e-03	4.27e-04
108	3000	11.7	0.48	2.65e-03	1.30e-04	5.31e-03	3.12e-03
140	5000	11.9	0.57	6.36e-03	8.77e-04	1.14e-02	4.87e-03
200	10000	11.7	0.67	1.51e-02	2.04e-03	2.69e-02	5.00e-03
282	20000	11.3	0.48	4.51e-02	6.93e-03	7.69e-02	8.21e-03

4.3.3 'Sprand' Matrix

The 'sprand' (sparse random) matrix is generated by Matlab function

```
A = sprand(m,n,s,1/c),
```

where m is the number of rows, n is the number of columns, s is the sparsity and c is the estimated condition number. We can control the condition number by the input c. When c is large, the generated matrix is ill conditioned. The coherence number is still small due to the randomness. Thus it belongs to the category 'ill conditioned and incoherent matrices'.

To test the robustness to the condition number, we fix $m = 90000, n = 300$ and the sparsity $s = 0.25$ and change c to get several matrices with large condition number; see Table 4.9-4.11.

Table 4.9: 'Sprand' Matrix $m = 90000, n = 300$: Residual and Iteration Steps.

$nnz(A)$	$\kappa(A^\intercal A)$	$\mu(A)$	Residual.CG	Iter.CG	Residual.RS	Iter.RS
87248	3.87e+03	7.17e-03	9.05e-08	98	9.97e-08	21
87208	1.91e+04	5.86e-03	8.70e-08	181	6.80e-08	38
86278	7.55e+04	3.91e-03	8.36e-08	264	7.94e-08	58
86654	2.89e+05	7.81e-03	9.17e-08	233	9.37e-08	50
86816	7.40e+05	7.71e-03	8.18e-07	296	4.77e-08	70

Table 4.10: 'Sprand' Matrix $m = 90000, n = 300$: Elapsed CPU Time

$nnz(A)$	Time.CG	Setup.CG	Sum.CG	Time.RS	Setup.RS	Sum.RS
87248	1.66	0.35	2.01	0.52	0.68	1.20
87208	3.23	0.43	3.65	0.88	0.65	1.53
86278	4.39	0.33	4.72	1.29	0.57	1.86
86654	3.92	0.35	4.27	1.13	0.56	1.69
86816	5.00	0.33	5.33	1.56	0.54	2.10

Notice that for very ill-conditioned matrices, CG without preconditioners will not reach the tolerance 10^{-7}; see row 5 in Table 4.9. Although theoretically CG will result in the exact solution with at most n-steps, the large condition number causes the instability. Our preconditioner is effective and PCG converges within 100 steps.

Table 4.11: 'Sprand' Matrix $m = 90000, n = 300$: Mean and Sample Standard Deviation

Iter.Mean	Iter.Std	Time.Mean	Time.Std	Setup.Mean	Setup.Std
23.3	1.95	0.57	4.45e-02	0.59	3.73e-02
39	1.70	0.92	4.13e-02	0.57	1.55e-02
60.9	3.63	1.41	7.98e-02	0.56	1.03e-02
51.2	2.44	1.18	5.73e-02	0.55	2.61e-02
69.4	2.63	1.60	6.14e-02	0.56	1.85e-02

Table 4.12: 'Sprand' Matrix $m = 40000$: Mean and Sample Standard Deviation

n	$\kappa(A^\intercal A)$	Iter.Mean	Iter.Std	Time.Mean	Time.Std	Setup.Mean	Setup.Std
50	28323	17.2	1.48	0.03	7.74e-03	0.04	1.37e-02
100	31278	26.3	1.70	0.09	6.29e-03	0.06	2.42e-03
200	60858	54.6	2.46	0.36	2.00e-02	0.16	5.84e-03
400	88807	72.1	4.84	1.07	8.05e-02	0.62	1.91e-02
800	1.13e+05	86.3	2.41	3.00	8.59e-02	3.04	8.30e-02

We fix $m = 40,000, c = 100, s = 0.25$ and vary n. Again our preconditioned PCG works well. The relation between column numbers n and averaged iteration steps are plotted in figure 4.2.

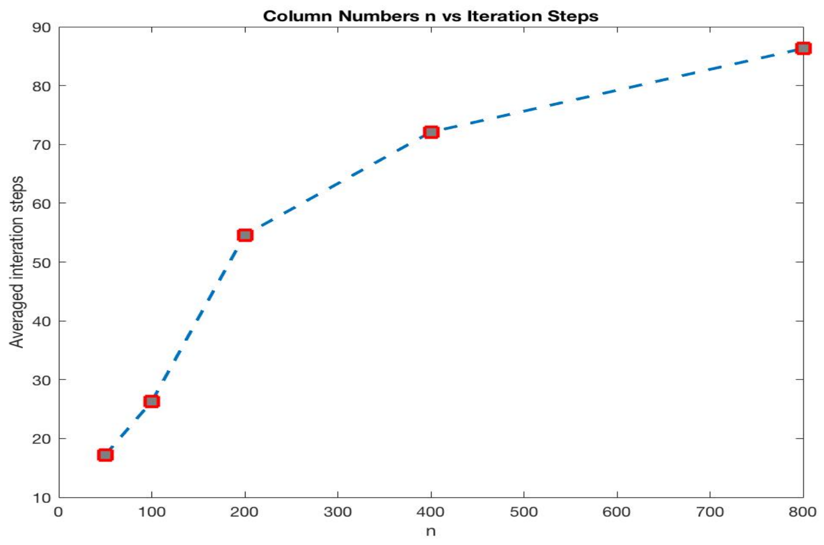


Figure 4.2: steps vs number of columns

We then fix $n = 200$ and $c = 100$ and vary m. The iteration steps are uniform to m. Notice that for fixed $n = 200$, the sample size $s = 4n \log n \approx 4239$ is fixed which is very small portion for large m.

Table 4.13: ‘Sprand’ Matrix $n = 200$: Mean and Sample Standard Deviation

m	$\kappa(A^\intercal A)$	Iter.Mean	Iter.Std	Time.Mean	Time.Std	Setup.Mean	Setup.Std
10000	58417	54.2	3.82	0.11	1.12e-02	0.07	4.95e-03
20000	48309	39.5	1.78	0.14	7.75e-03	0.10	6.81e-03
40000	69855	45.2	3.36	0.32	2.55e-02	0.17	5.40e-03
70000	49447	46.8	2.66	0.56	2.85e-02	0.26	9.24e-03
90000	73177	56	2.21	0.86	3.08e-02	0.34	9.22e-03

4.3.4 UDV Matrix

The UDV matrices are a random matrix generated by $A = UDV$, where U is an $m \times n$ random orthonormal matrix, V is an $n \times n$ random orthonormal matrix and $D = \text{diag}[1, 1+(c-1)/n, \cdots, c]$ and c is the estimated condition number. For this kind of matrices, we can control the condition number by parameter c. When c is large enough, it belongs to the category ‘ill conditioned and incoherent matrices’.

Table 4.14: ‘UDV’ Matrix $m = 90000, n = 300, nnz(A) = 90000$: Residual and Iteration Steps

$\kappa(A^\intercal A)$	$\mu(A)$	Residual.CG	Iter.CG	Residual.RS	Iter.RS
5936	4.81e-03	9.68e-08	116	7.21e-08	24
18853	4.61e-03	9.68e-08	202	8.81e-08	38
1.44e+05	4.66e-03	4.42e-07	294	8.44e-08	68
4.75e+05	4.65e-03	1.32e-05	369	7.92e-08	86
1.07e+06	4.72e-03	9.89e-06	253	4.44e-08	91

Again CG fails to converge for the last three matrices in ‘UDV’ group when the condition number is large.

Table 4.15: ‘UDV’ Matrix $m = 90000, n = 300, nnz(A) = 90000$: Elapsed CPU Time

$\kappa(A^\intercal A)$	$\mu(A)$	Time.CG	Setup.CG	Sum.CG	Time.RS	Setup.RS	Sum.RS
5936	4.81e-03	2.44	0.42	2.86	0.49	0.47	0.96
18853	4.61e-03	3.77	0.43	4.20	0.71	0.45	1.16
1.44e+05	4.66e-03	5.47	0.42	5.89	1.30	0.45	1.75
4.75e+05	4.65e-03	5.88	0.44	6.32	1.57	0.45	2.02
1.07e+06	4.72e-03	5.53	0.43	5.96	1.65	0.44	2.09

Table 4.16: ‘UDV’ Matrix $m = 90000, n = 300, nnz(A) = 90000$: Mean and Sample Standard Deviation

$\kappa(A^\intercal A)$	Iter.Mean	Iter.Std	Time.Mean	Time.Std	Setup.Mean	Setup.Std
5936	23.1	0.57	0.51	7.8e-02	0.45	3.79e-02
18853	38.8	0.63	0.76	7.51e-02	0.43	6.91e-03
1.44e+05	72	0.82	1.42	1.67e-01	0.48	9.76e-02
4.75e+05	86.4	0.52	1.86	2.37e-01	0.47	5.84e-02
1.07e+06	90.2	0.42	1.81	1.98e-01	0.48	5.91e-02

4.3.5 Graph Laplacian Matrix

The graph Laplacian matrices are generated based on the graph of sparse random matrices. We extract positions of all nonzero entries from matrix A generated in the previous test, i.e. the ’sprand’ group. And use them to construct edge incidence matrix B. The corresponding row in matrix B represents the edge with weight and direction in the graph generated by $A^\intercal A$. For example, if $(A^\intercal A)_{13} = 2$, this means there is an edge with weight 2 pointing from vertex v_1 to vertex v_3 in the graph. The corresponding row in B is $b_k = \begin{bmatrix} 2 & 0 & -2 \end{bmatrix}$, where k is the index of the edge pointing from vertex v_1 to vertex v_3. The extend B by a scaled identity matrix – $\tilde{B} = \begin{bmatrix} B \\ cI_n \end{bmatrix}$, with c a real number between 0 and 1. The extension is to make the matrix product $B^\intercal B$ is nonsingular. Again we normalize $\tilde{B} \leftarrow \tilde{B}D^{-1}$, where $D_{jj} = \|b^j\|_2, j \in [n]$ and b^j is the jth column of $\tilde{B}$.The scaled matrix $\tilde{B}_{(m+n)\times n}$ is the graph Laplacian edge matrix.

Table 4.17: Graph Laplacian Matrix $n = 300$: Residual and Iteration Steps

m	$nnz(A)$	$\kappa(A^\intercal A)$	$\mu(A)$	Residual.CG	Iter.CG	Residual.RS	Iter.RS
87548	87248	11.14	1	4.52e-08	17	4.64e-08	12
87508	87208	153.7	1	9.49e-08	32	5.42e-08	13
86578	86278	2276.4	1	8.69e-08	47	2.54e-08	20
86954	86654	31910	1	8.35e-08	48	5.81e-08	17
87116	86816	622310	1	7.50e-08	29	3.80e-08	20

Table 4.18: Graph Laplacian Matrix $n = 300$: Elapsed CPU Time

m	Time.CG	Setup.CG	Sum.CG	Time.RS	Setup.RS	Sum.RS
87548	1.29e-02	7.03e-03	1.99e-02	1.17e-02	4.57e-02	5.74e-02
87508	1.73e-02	5.64e-03	2.29e-02	1.12e-02	3.63e-02	4.75e-02
86578	2.61e-02	5.13e-03	3.12e-02	1.62e-02	2.99e-02	4.61e-02
86954	2.71e-02	5.61e-03	3.27e-02	1.72e-02	3.70e-02	5.42e-02
87116	1.73e-02	5.63e-03	2.29e-02	1.72e-02	3.63e-02	5.35e-02

Table 4.19: Graph Laplacian Matrix $n = 300$: Mean and Sample Standard Deviation

Iter.Mean	Iter.Std	Time.Mean	Time.Std	Setup.Mean	Setup.Std
12.4	0.70	8.93e-03	5.06e-03	1.62e-02	6.07e-03
13.8	0.42	1.07e-02	7.99e-04	1.46e-02	6.33e-04
19.1	0.57	1.63e-02	1.54e-03	1.56e-02	7.74e-04
17.6	0.52	1.41e-02	6.40e-04	1.49e-02	1.31e-04
19.5	0.71	1.65e-02	1.05e-03	1.55e-02	1.14e-03

4.3.6 Non-negative Constrained Problem

Our preconditioner also works for non-negative constrained least squares problem.

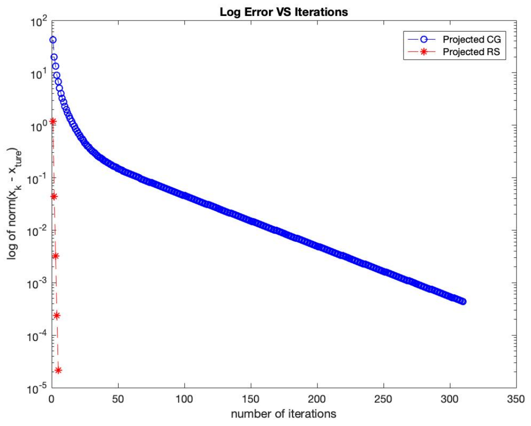
$$Ax = b \text{ subject to } x \in \mathcal{C}, \tag{4.10}$$

where $A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^m, \mathcal{C} = \mathbb{R}^n_+$. In order to satisfy the non-negative constrain, a projection step is implemented every 5 steps of CG. The projection is defined as

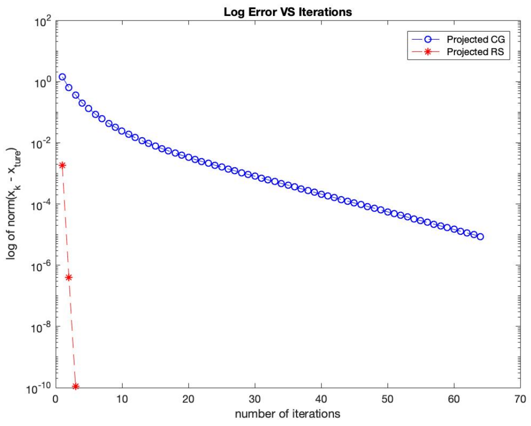
$$\mathcal{P}_{\mathcal{C}}(x)_i = \begin{cases} x_i, \ x_i \geq 0 \\ 0, \ x_i < 0; \end{cases}$$

for $i = 1, \cdots, n$.

The data comes from public domain MATLAB package AIR Tools II [1]. In test problems, we can generate a sparse matrix A, a right hand side b and an exact solution x. We tested on three problems including ‘fancurvedtomo’ and ‘sphericaltomo’. CG and CG with Gauss Seidel preconditioner are compared. Since all three matrices A have a small number of columns, non sampling is done in



(a)



(b)

Figure 4.3: (a) fancurvedtomo (b) sphericaltomo

this example. The tolerance is set to be 10^{-5} while maximum iteration steps is set to be 5000. Numerical results are presented in TABLE 4.20. The decreasing of error $\|x_k - x_*\|$ as iterations increasing are shown in FIGURE 4.3.

Table 4.20: Non-negative Constrained Problem $m = 10260, n = 1600$

	$\kappa(A)$	error.CG	time.CG	iter.CG	error.RS	time.RS	iter.RS
'fancurvedtomo'	154.22	4.36e-04	2.02	310	2.14e-05	1.11	5
'sphericaltomo'	71.93	8.66e-05	0.40	64	1.10e-10	0.73	3

Compared to diagonal preconditioned CG with projection, our preconditioner exhibits a significant improvement on the numerical performance. The iteration steps are decreased from 310 to 5 and 64 to 3 for 'fancurvedtomo' and 'sphericaltomo' respectively. This result shows a potential capability of dealing with constrained problem in future.

4.3.7 Summary of Numerical Results

It is well known that CG works well for the well conditioned matrices. However, CG does not converge in the fifth matrix for the 'sprand' group and the last three matrices in UDV group, see Tables 4.9 and 4.14. Round off errors occur in orthogonalization make CG fail to converge within n steps. With our random sampling preconditioner, PCG converges for all of them. The iterative steps and elapsed CPU time are reduced to $1/3$ for ill conditioned matrices in both the 'sprand' group and UDV group. In the last case, we tests on the graph Laplacian matrices. Due to the simple structure of the matrices, both methods converge with 0.04 for elapsed CPU time. But our method almost halve the iterative steps to converge. Although we cannot prove the uniform convergence, the performances listed ahead indicates that our random sampling preconditioner is efficient and effective.

Another remark is the standard deviation of iterations and CPU time are acceptable. The randomness comes from sampling which results a small variance in iterative steps and CPU time. Most of the examples, the ration of standard deviation to mean of iterative steps is from less than 0.1% to 2%. Only for the 'sprand' cases, we have a ratio ranges from 2% to 5%.

Chapter 5

Review of Coordinate Descent Type Methods

In this chapter we review popular methods for nonconstraint optimization problems (5.1) mentioned in Chapter 1.

$$\min_{x} f(x), \tag{5.1}$$

with assumptions f is convex or strongly convex and ∇f is Lipschitz continuous.

A generic iterative method for solving (5.1) is in the form

$$x^{k+1} = x^k + \alpha_k s_k, \tag{5.2}$$

where x^k is current point, α_k is a step size and s_k is a search direction. Gradient-type method searches in direction related to gradient direction. For example, gradient descent (GD) method updates with negative gradient direction, i.e. $s_k = -\nabla f(x_k)$. In huge-scale optimization problem, the computation of a function value or simplest full-dimensional gradient would be very expensive [31]. This is the motivation to update one or few components of gradient every update. Coordinate descent (CD) method updates with one component of negative gradient direction while keeping other components fixed. Block coordinate descent (BCD) method updates with several components of negative gradient direction while keep the remaining components fixed. When ∇f is Lipschitz continuous, we can prove sub-linear convergence for GD for convex f with bounded-level assumption and linear convergence for strongly-convex f. There are two fashions of CD – cyclic CD (CCD) and randomized CD (RCD). CCD updates by traversing components cyclicly while RCD chooses components to update randomly.

For cyclic CD (CCD), it is almost impossible to estimate the rate of convergence [31]. Luo and Tseng proved the local rate of convergence for cyclic CD for objective function of the form

$$f(x) = g(Ex) + \langle b, x\rangle,$$

where g is a proper closed convex function in $\mathbb{R}^m$, E an $m \times n$ matrix having no zero column, and b a vector in $\mathbb{R}^n$ with almost cyclic rule and Gauss-Southwell Rule in [26].

When adding randomness to CD, i.e. choosing component via some probability in each update, enables global convergence analysis feasible. Let index i_k be a random variable at update k. Denote $\nabla_{i_k} f(x^k) = [\nabla f(x^k)]_{i_k}$. Taking expectation of $\nabla_{i_k} f(x^k)$ with respect to random variable i_k results in an unbiased estimate of full gradient $\nabla f(x^k)$ with some probability. Then RCD converges similarly as CD with a smaller convergence rate in expectation sense. Nesterov gives a type of probability in [31], a chosen strategy is given by

$$p_i^\alpha = L_i^\alpha \left(\sum_{j=1}^{n} L_j^\alpha\right)^{-1}, \ i \in 1, \cdots, n, \tag{5.3}$$

where L_i is Lipschitz constant at each subspace $\mathcal{V}_i$ consisting of one component or several components and provides convergence of Randomized Block Coordinate Descent (RCD) via (5.3).

Nesterov also shows the performance of RCD is better than standard worst-case bounds for deterministic algorithm [31]. Wright provides a simple version of convergence of RCD with uniform sampling, fixed step size of convex function f under bounded level assumptions in [48].

We shall present convergence analysis for CD and RCD, RBCD with uniform sampling ($\alpha = 0$ in (5.3)) or importance sampling ($\alpha = 1$ in (5.3)), fixed step size of convex or strongly-convex function f for comparisons of our methods randomized fast subspace descent methods (RFASD).

Besides this, more work have been done for Coordinate Descent Type methods. For example, Lu developed a randomized block proximal damped newton method (RBPDN) in [25] for composite minimization problem

$$\min_{x\in\mathbb{R}^N}\{F(x) := f(x) + \Psi(x)\},$$

where $f(x)$ is convex and differentiable and $\Psi(x) = \sum_{i=1}^{n} \Psi_i(x_i)$ has a block separable structure. When RBPDN applied to smooth convex minimization problem, it uses Newton method in each block, i.e. update the search direction by $H_{ii}^{-1}(x)[\nabla f(x)]_i$. The complexity reduces compared to Newton's method since only Newton's iterations on local problems are needed, i.e. the size of Hessian is reduced to the block size. There is a tradeoff between convergence rate and complexity considering the dimension of subspace. If the dimension of subspace is too small, i.e. 1 or 2, the subspace Hessian loses lots of information leading to slow convergence in each update. While if the dimension of subspace is large, for example, $\frac{N}{2}$, the computation of subspace Hessian and Hessian inverse is still costing.

Another approach besides RCD which is also a light version of GD is stochastic gradient descent methods (SGD). Instead of RCD which picks up one component of x to update, SGD picks one instance of data and associated objective function to compute gradient. Similarly, we can increase the size of data to update in one iteration which corresponds to mini-batch SGD.

Apply RCD to linear equation (1.1) gives randomized Gauss Seidel Method (RGS) and apply SGD to l(1.1) gives randomized Kaczmarz (RK). RGS and RK are mathematically equivalent when applying to primal and dual problems. Enlightened by this face, RCD and SGD can also solve primal and dual problems.

In section 2 we present algorithms of GD and convergence analysis of GD. In section 3, we review algorithms of CCD, RCD and RBCD and convergence of RCD and BRCD. In section 4, we give brief introduction of SGD and mini-batch SGD. Last but not least, in section 5, we give the duality relation of RCD and SGD and apply them to linear equation to get RGS and RK respectively.

5.1 Gradient Descent Methods

5.1.1 Gradient Descent Scheme

This first method introduced here is Gradient Descent Method (GD), which is a fundamental algorithm in convex optimization. The update form of GD is

$$x^{k+1} = x^k - \alpha_k \nabla f(x^k), \tag{5.4}$$

where $\nabla f(x^k)$ is gradient of function f evaluated at current point x^k and α_k is a value chosen either by (1) performing exact minimization, (2) or choosing a value satisfying traditional line search conditions, (3) or making a "short-step" based on prior knowledge of the properties of f [48] .

Algorithm of GD for (5.1) is presented in Algorithm 4.

```
1: Choose $x^0$ and $k \leftarrow 0$;
2: for $k = 0, 1, \cdots$ do
3:     Compute gradient of $f$ and evaluated on $x^k$ to get $\nabla f(x^k)$;
4:     Update with step size $\alpha^k > 0$
           $x^{k+1} \leftarrow x^k - \alpha_k \nabla f(x^k);$
           $k \leftarrow k + 1;$
5: end for
```

Algorithm 4: Gradient Descent Method

We recall basic assumptions for objective function f in Chapter 1.

- (LC) The first order derivative of f is Lipschitz continuous with Lipschitz constant L, i.e.,

$$\|\nabla f(x) - \nabla f(y)\|_{\mathcal{V}'} \leq L\|x - y\|_{\mathcal{V}}, \quad \forall\, x,\, y \in \mathcal{V}.$$

- (C) f is convex, i.e.,

$$f(x) \geq f(y) + \langle \nabla f(y), (x - y)\rangle, \quad \forall\, x,\, y \in \mathcal{V}.$$

- (SC) f is strongly convex with strong convexity constant $\mu > 0$, i.e.,

$$\langle \nabla f(x) - \nabla f(y), x - y\rangle \geq \mu\|x - y\|_{\mathcal{V}}^2, \quad \forall\, x,\, y \in \mathcal{V}.$$

Under Lipschitz continuous (LC) assumption for ∇f, GD achieves linear convergence for strongly convex f and sub-linear convergence for strongly convex f with fixed step-size $0 < \alpha_k < \frac{2}{L}$ and optimal convergence rate achieved when $\alpha_k = \frac{1}{L}$, where L is Lipschitz constant of ∇f.

5.1.2 Convergence of Gradient Descent Methods

Theorem 5.1.1. *Suppose Assumption (LC) holds. Suppose $\alpha_k := \alpha$ is constant and $\alpha \in (0, \frac{2}{L})$ in Algorithm 4. The sequence $\{f(x^k)\}$ generated by Algorithm 4 is strictly decreasing for $k > 0$.*

Proof. (LC) is mathematical equivalently to

$$f(x) - f(y) \leq \langle \nabla f(y), x - y \rangle + \frac{L}{2}\|x - y\|_{\mathcal{V}}^2. \tag{5.5}$$

Then combine with GD update form (5.4). Furthermore, assume $\alpha_k := \alpha$ is fixed.

$$\begin{aligned} f(x^{k+1}) - f(x^k) &\leq \langle \nabla f(x^k), x^{k+1} - x^k \rangle + \frac{L}{2}\|x^{k+1} - x^k\|_{\mathcal{V}}^2 \\ &= -\alpha\|\nabla f(x^k)\|_{\mathcal{V}'}^2 + \frac{L}{2}\alpha^2\|\nabla f(x^k)\|_2^2 \\ &= -\alpha(1 - \frac{L}{2}\alpha)\|\nabla f(x^k)\|_{\mathcal{V}'}^2 \end{aligned}$$

Let $0 \leq \alpha \leq \frac{2}{L}$, we have $\alpha(1 - \frac{L}{2}\alpha) > 0$. Thus $f(x^{k+1}) - f(x^k) < 0$, which implies the sequence generated by Algorithm 4 is strictly decreasing. □

5.1.3 Linear Convergence of Strongly Convex Function

Now we present the result that GD achieves linear convergence for strongly convex objective function f.

Theorem 5.1.2. *Suppose Assumption (LC) and (SC) hold. Set constant step size $\alpha = \frac{1}{L}$. For all $k > 0$, we have*

$$f(x^k) - f^* \leq \left(1 - \frac{\mu}{L}\right)^k (f(x^0) - f^*). \tag{5.6}$$

To achieve accuracy ϵ, we need $k = O\left(\frac{L}{\mu}\ln\left(\frac{1}{\epsilon}\right)\right)$ iterations.

Proof. By the proof of Theorem 5.1.1, we have

$$f(x^{k+1}) - f(x^k) \leq -\alpha(1 - \frac{L}{2}\alpha)\|\nabla f(x^k)\|^2_{\mathcal{V}'} \tag{5.7}$$

Inserting f^* and $-f^*$ into L.H.S. of (5.7) gives

$$f(x^{k+1}) - f^* + f^* - f(x^k) \leq -\alpha(1 - \frac{L}{2}\alpha)\|\nabla f(x^k)\|^2_{\mathcal{V}'}$$

Denote $d^k = f(x^k) - f^*$, for $k > 0$.

$$d^{k+1} - d^k \leq -\alpha(1 - \frac{L}{2}\alpha)\|\nabla f(x^k)\|^2_{\mathcal{V}'}$$

Let $\alpha = \frac{1}{L}$, $-\alpha(1 - \frac{L}{2}\alpha)$ achieves maximum $-\frac{1}{2L}$. R.H.S of (5.7) changes to

$$-\alpha(1 - \frac{L}{2}\alpha)\|\nabla f(x^k)\|^2_{\mathcal{V}'} = -\frac{1}{2L}\|\nabla f(x^k)\|^2_{\mathcal{V}'}$$

By Assumption (SC), we have

$$\|\nabla f(x^k)\|^2_{\mathcal{V}'} \geq 2\mu(f(x^k) - f^*) = 2\mu d^k$$

Thus

$$d^{k+1} - d^k \leq -\frac{\mu}{L}d^k$$

i.e.

$$d^{k+1} \leq \left(1 - \frac{\mu}{L}\right)d^k \leq \left(1 - \frac{\mu}{L}\right)^k d^0$$

This gives convergence property of GD for strongly-convex objective function f.

In order to achieve accuracy ϵ, we require

$$f(x^k) - f^* \leq \left(1 - \frac{\mu}{L}\right)^k (f(x^0) - f^*) \leq \epsilon$$

$$\begin{aligned}\left(1 - \frac{\mu}{L}\right)^k &\leq \frac{\epsilon}{d^0} \\ k &\geq O\left(\frac{\ln(\epsilon)}{\ln\left(1 - \frac{\mu}{L}\right)}\right) \approx O\left(\frac{\ln \epsilon}{-\frac{\mu}{L}}\right) \\ &\approx O\left(\frac{L}{\mu} \ln\left(\frac{1}{\epsilon}\right)\right)\end{aligned}$$

We use the fact that the natural logarithm (with base e) has Maclaurin series

$$\ln(1 - x) = -\sum_{n=1}^{\infty} \frac{x^n}{n}.$$

$\square$

5.1.4 Sub-linear Convergence of Convex Function

In this section we present the sub-linear convergence property of convex objective function f. Before move to convex case, one more assumption need to be made.

- (BL) Bounded level set: f is convex and uniformly Lipschitz continuously differentiable, and attains its minimum value f^* on a set S. There is a finite constant R_0 such that the level set for f defined by x^0 is bounded, that is,

$$\max_{x^* \in S} \max_x \{\|x - x^*\| : f(x) \leq f(x^0)\} \leq R_0. \tag{5.8}$$

Theorem 5.1.3. *Suppose Assumption (LC), (C) and (BL) hold. For all $k > 0$, we have*

$$f(x^k) - f^* \leq \frac{2LR_0^2}{k}, \tag{5.9}$$

To achieve accuracy ϵ, we need $k = O\left(\frac{L}{\epsilon}\right)$ iterations.

Proof. Since f is convex,

$$f(x^k) - f^* \leq \langle \nabla f(x^k), x^k - x^* \rangle \leq \|\nabla f(x^k)\|_{\mathcal{V}'} \|x^k - x^*\|_{\mathcal{V}} \leq R_0 \|\nabla f(x^k)\|_{\mathcal{V}'}.$$

By Assumption (LC) with $\alpha \in (0, \frac{1}{L})$

$$f(x^{k+1}) - f(x^k) \leq -\frac{1}{2L} \|\nabla f(x^k)\|_{\mathcal{V}'}^2$$

Denote $d^k = f(x^k) - f^*$. Then

$$d^{k+1} - d^k \leq -\frac{1}{2L} \|\nabla f(x^k)\|_{\mathcal{V}'}^2 \leq -\frac{1}{2LR_0^2} (d^k)^2.$$

Divide both sides by $(d^k)^2$, we have

$$\frac{1}{d^{k+1}} - \frac{1}{d^k} = \frac{d^k - d^{k+1}}{d^k d^{k+1}} \geq \frac{d^k - d^{k+1}}{(d^k)^2} \geq \frac{1}{2LR_0^2}.$$

Apply this result recursively, we have

$$\frac{1}{d^k} \geq \frac{1}{d^0} + \frac{k}{2LR_0^2} \geq \frac{k}{2LR_0^2}$$

In order to achieve accuracy ϵ, we have

$$\frac{2LR_0^2}{k} \le \epsilon$$

$$k \ge \frac{2LR_0^2}{\epsilon} = O\left(\frac{L}{\epsilon}\right)$$

$\square$

5.2 Randomized Coordinate Descent Methods

In Wright's paper of Coordinate Descent Algorithms [48], he describes 'coordinate Descent (CD) are iterative methods whose each iterate is obtained by fixing most components of the variable vector x at their values from the current iteration, and approximately minimizing the objective with respect to the remaining components.' The difference between GD and CD is for GD, each iterate update all components of x in gradient descent direction while for CD, each iterate only updates one or several components of x and keep remaining unchanged. Here is the update of CD.

$$x^{k+1} = x^k - \alpha_k [\nabla f(x^k)]_{i_k} e_{i_k} \tag{5.10}$$

Two essential questions about CD update form is (1) how to choose α_k and (2) how to choose component index i_k? Turning to step size α_k, three strategies can be applied. The choice of step size can be implemented in a similar way as we discuss for GD, i.e., exact line search, or traditional line search or predefined "short step" based on prior knowledge of f [48].

To answer question (2), there are two popular ways of selections.

- (Cyclic Selection)

$$i_k = [k \mod n] + 1, \quad k = 1, 2, \cdots \tag{5.11}$$

- (Randomized Selection) i_k's with $k \in \{0, 1, 2, \cdots\}$ are independent identical distributed random variables with mass probability function defined in 5.12.

$$P\{i_k = j\} = p_j, \quad j \in \{1, 2, \cdots, n\} \tag{5.12}$$

If i_k is chosen in cyclic fashion, i_0 starts from 1 and move forward one step in each update until reach the last component of x. Then i_k comes back to 1 again and updates cyclicly. In this case, each component would be updated once in n updates. In randomized version, i_k is chosen according to some probability p_j with $\sum_j p_j = 1$ and $0 \leq p_j \leq 1$ for $j \in \{1, \cdots, n\}$. In this case, not every component will definitely get updated in n iterations. Algorithm will pay more emphasis on component associated with larger probability and less on the ones with smaller probability. In randomized version, there are various sampling strategies including sampling without replacement (shuffling) and with replacement, uniform sampling or importance sampling. These variants would be discussed later sections.

The algorithm of cyclic coordinate descent (CCD) is presented in Algorithm 5.

```
1: Choose $x^0$ and $k \leftarrow 0$;
2: for $k = 0, 1, \cdots$ do
3:     Choose index $i_k \in \{1, \cdots, n\}$ via cyclic selection (5.11)
4:     Evaluate gradient $\nabla f$ of component $i_k$ at current point $x^k$;
5:     Update with step size $\alpha^k > 0$
           $x^{k+1} \leftarrow x^k - \alpha^k [\nabla f(x^k)]_{i_k} e_{i_k};$
           $k \leftarrow k + 1;$
6: end for
```

Algorithm 5: Cyclic Coordinate Gradient Descent Method

The algorithm of randomized coordinate descent (RCD) is presented in Algorithm 6.

```
1: Choose $x^0$ and $k \leftarrow 0$;
2: for $k = 0, 1, \cdots$ do
3:     Choose index $i_k \in \{1, \cdots, n\}$ via randomized selection (5.12);
4:     Evaluate gradient $\nabla f$ of component $i_k$ at current point $x^k$;
5:
6:     Update with step size $\alpha^k > 0$;
           $x^{k+1} \leftarrow x^k - \alpha^k [\nabla f(x^k)]_{i_k} e_{i_k};$
           $k \leftarrow k + 1;$
7: end for
```

Algorithm 6: Randomized Coordinate Descent Method

5.2.1 Convergence of Randomized Coordinate Descent Method

For cyclic CD (CCD), there is few job to prove and convergence and estimate the rate of convergence [31]. A local rate of convergence of block coordinate descent method for convex differentiable minimization with almost cyclic and Gauss-Seidel rules can be found in [26] and for nondifferentiable minimization in [46]. Full approximation storage scheme also gives a frame work for convergence analysis of CCD since Cauchy Schwarz holds for coordinate decomposition and dependance on the scaled condition number is a constant [11].

Now we discuss the global convergence of randomized CD (RCD) for strongly convex function and convex function.

Euclidean space $\mathbb{R}^n$ has a natural space decomposition – decomposed as direct sum of each component.

$$\mathbb{R}^n = \oplus_{i=1}^n \text{span}\{e_i\} \tag{5.13}$$

In order to analysis convergence of RCD, we assume component-wise Lipschitz continuous on f.

- (LCi1) Lipschitz continuous restricted in each component with Lipschitz constant L_i, i.e.,

$$\|[\nabla f(x+te_i)]_i - [\nabla f(x)]_i\| \leq L_i|t|, \quad \forall\, t \in \mathbb{R}$$

In Nestrov's paper [31], a strategy to randomly choose index i is given by

$$p_i^\alpha = L_i^\alpha \left(\sum_{j=1}^n L_j^\alpha\right)^{-1}, \, i \in 1, \cdots, n. \tag{5.14}$$

We specifies two strategies with corresponding optimal fixed step size and give convergence results for both strongly convex objective function and convex function.

Let $\alpha = 0$ gives first sampling strategy – uniform sampling. For uniform sampling, optimal step size is given by

$$\alpha \equiv 1/L_{\max}, \quad L_{\max} = \max_{i=1}^{n} L_i.$$

- **Uniform Samnpling**. The probability of choosing i-th subspace is

$$p_i = \frac{1}{n}, \quad i \in \{1, 2, \dots, n\}. \tag{5.15}$$

Let $\alpha = 1$ gives another popular sampling strategy – importance sampling. For importance sampling, optimal step size is given by

$$\alpha_{i_k} = 1/L_{i_k},\, i_k \in \{1, \cdots, n\}.$$

- **Importance Sampling**. The probability of choosing i-th subspace is

$$p_i = \frac{L_i}{\ell}, \quad i \in \{1, 2, \dots, n\}, \tag{5.16}$$

  where $\ell = \sum_{i=1}^{n} L_i$.

Wright defines coordinate Lipschitz constant $L_{\max} = \max_{i=1}^{n} L_i$ proves linear convergence for strongly convex objective function and sub-linear convergence for convex objective function in expectation sense with fixed step size $\alpha \equiv 1/L_{\max}$ in [48]. The proof is a modification of convergence analysis of GD. This is the case applying uniform sampling with fixed step size.

We now present both sampling strategies with corresponding step size for strongly-convex function and convex function respectively in Theorem 5.2.1, Theorem 5.2.2 , Theorem 5.2.3 and Theorem 5.2.4 respectively. The convergence results are similar to Gradient Descent case Theorem 5.1.2 and Theorem 5.1.3 but in expectation.

Theorem 5.2.1. *Suppose Assumption (LCi1) and (SC) hold. Set constant step size* $\alpha \equiv 1/L_{\max}$

. Let x^k be approximation obtained by Algorithm 6 after $k-1$ update. Randomly choose i_k with uniform distribution independently from previous choice. For all $k>0$, we have

$$\mathbb{E}(f(x^k)) - f^* \le \left(1 - \frac{\mu}{nL_{\max}}\right)^k (f(x^0) - f^*). \tag{5.17}$$

To achieve accuracy ϵ, we need $k = O\left(\frac{nL_{\max}}{\mu}\ln\left(\frac{1}{\epsilon}\right)\right)$ iterations.

Theorem 5.2.2. *Suppose Assumption (LCi), (C) and (BL) hold. Let x^k be approximation obtained by Algorithm 6 after $k-1$ update. For all $k>0$, we have*

$$\mathbb{E}[f(x^k)] - f^* \le \frac{2nL_{\max}R_0^2}{k}, \tag{5.18}$$

To achieve accuracy ϵ, we need $k = O\left(\frac{nL_{\max}}{\epsilon}\right)$ iterations.

Theorem 5.2.3. *Suppose Assumption (LCi1) and (SC) hold. Let x^k be approximation obtained by Algorithm 6 after $k-1$ update. Randomly choose i_k with importance sampling 5.16 independently from previous choice. Set constant step size $\alpha_{i_k} = 1/L_{i_k}$. For all $k>0$, we have*

$$\mathbb{E}(f(x^k)) - f^* \le \left(1 - \frac{\mu}{\ell}\right)^k (f(x^0) - f^*), \tag{5.19}$$

where $\ell = \sum_{i=1}^n L_i$. To achieve accuracy ϵ, we need $k = O\left(\frac{nL_{\max}}{\mu}\ln\left(\frac{1}{\epsilon}\right)\right)$ iterations.

Theorem 5.2.4. *Suppose Assumption (LCi), (C) and (BL) hold. Let x^k be approximation obtained by Algorithm 6 after $k-1$ update. Randomly choose i_k with importance sampling 5.16 independently from previous choice. Set constant step size $\alpha_{i_k} = 1/L_{i_k}$. For all $k>0$, we have*

$$\mathbb{E}[f(x^k)] - f^* \le \frac{2\ell}{R_0^2}k, \tag{5.20}$$

where $\ell = \sum_{i=1}^n L_i$. To achieve accuracy ϵ, we need $k = O\left(\frac{\ell}{\epsilon}\right)$ iterations.

To conclude, RCD achieves linear convergence of strongly convex function f and sub-linear convergence of convex function f with fixed step size $\alpha \equiv 1/L_{\max}$ in expectation sense. The benefits of RCD compared to GD is instead of computing gradient of all components in one update, it needs gradient of one component at each update, which reduce the complexity cost. The drawbacks of RCD are it needs component-wise Lipschitz constant L_i to decide step size and sacrifices convergence rate from $(1 - \frac{\mu}{L})$ to $(1 - \frac{\mu}{nL_{\max}})$ with uniform sampling and to $(1 - \frac{\mu}{\ell})$ with importance sampling in strongly-convex case and $\frac{L}{k}$ to $\frac{nL_{\max}}{k}$ with uniform sampling and to $\frac{\ell}{k}$ with importance sampling for convex case.

5.2.2 Block Randomized Coordinate Descent Method

Instead of updating one component every iteration, Block Randomized Coordinate Descent (RBCD) updates several components at one iteration and keeps other components fixing. We use the same setting as in Nesterov's paper [31].

Consider a decomposition of $\mathbb{R}^n$ on J subspaces:

$$\mathbb{R}^n = \oplus_{i=i}^{J} \mathbb{R}^{n_i}, \; n = \sum_{i=1}^{J} n_i.$$

The unit matrix can be partitioned correspondingly.

$$I_n = (U_1, \cdots, U_J) \in \mathbb{R}^{n \times n},$$

where

- $U_i \in \mathbb{R}^{n \times n_i}$ maps from $\mathbb{R}^{n_i}$ to $\mathbb{R}^n$ is called prolongation or inclusion;
- $U_i^{\intercal} \in \mathbb{R}^{n_i \times n}$ maps from $\mathbb{R}^n$ to $\mathbb{R}^{n_i}$ is called restriction or projection.

Then any vector $x = \begin{bmatrix} x_{(1)}, \cdots, x_{(J)} \end{bmatrix} \in \mathbb{R}^n$ where $x_{(i)} \in \mathbb{R}^{n_i}$ is decomposed as

$$x = \sum_{i=1}^{J} U_i x_{(i)}$$

The partial gradient $\nabla f_i(x)$ with respect to $x_{(i)}$ is defined as

$$\nabla_i f(x) = U_i^\intercal \nabla f(x) \in \mathbb{R}^{n_i},\ x \in \mathbb{R}^n.$$

Similarly, we requires the Lipschitz continuous conditions on each subspace $\mathbb{R}^{n_i}$, $i \in 1, \cdots, J$.

- (LCi2) Lipschitz continuous restricted in each subspace with Lipschitz constant L_i, i.e.,

$$\|\nabla_i f(x + U_i h_i) - \nabla_i f(x)\ \leq L_i \|h_i\|,\ h_i \in \mathbb{R}^{n_i},\ i = 1, \cdots, J,\ x \in \mathbb{R}^n.$$

We describes algorithm of RBCD as follows.

```
1: Choose $x^0$ and $k \leftarrow 0$;
2: for $k = 0, 1, \cdots$ do
3:     Choose index $i_k \in \{1, \cdots, J\}$ according to distribution $p$;
4:     Evaluate gradient $\nabla f$ of component $i$ at current point $x^k$;
5:     Update with step size $\alpha^k > 0$;
           $x^{k+1} \leftarrow x^k - \alpha^k U_{i_k} \nabla_{i_k} f(x);$
           $k \leftarrow k + 1;$
6: end for
```

Algorithm 7: Randomized Block Coordinate Descent Algorithm [31]

Convergence results are provided in following theorems.

Theorem 5.2.5 (Uniform sampling & strongly convex case)**.** *Suppose Assumption (LCi2) and (SC) hold. Set constant step size $\alpha \equiv 1/L_{\max}$. Randomly choose i_k with uniform sampling independently from previous choice. For all $k > 0$, we have*

$$\mathbb{E}(f(x^k)) - f^* \leq \left(1 - \frac{\mu}{JL_{\max}}\right)^k (f(x^0) - f^*). \tag{5.21}$$

To achieve accuracy ϵ, we need $k = O\left(\frac{JL_{\max}}{\mu} \ln\left(\frac{1}{\epsilon}\right)\right)$ iterations.

Theorem 5.2.6 (Importance sampling & strongly convex case)**.** *Suppose Assumption (LCi2) and (SC) hold. Set constant step size $\alpha_{i_k} = 1/L_{_k}$. Randomly choose i_k with importance sampling independently from previous choice. For all $k > 0$, we have*

$$\mathbb{E}(f(x^k)) - f^* \leq \left(1 - \frac{\mu}{\ell}\right)^k (f(x^0) - f^*). \tag{5.22}$$

To achieve accuracy ϵ, we need $k = O\left(\frac{\ell}{\mu} \ln\left(\frac{1}{\epsilon}\right)\right)$ iterations.

Theorem 5.2.7 (Uniform sampling & convex case)**.** *Suppose Assumption (LCi2), (C) and (BL) hold. Randomly choose i_k with uniform sampling independently from previous choice. For all $k > 0$, we have*

$$\mathbb{E}[f(x^k)] - f^* \leq \frac{2JL_{\max}R_0^2}{k}, \tag{5.23}$$

To achieve accuracy ϵ, we need $k = O\left(\frac{JL_{\max}}{\epsilon}\right)$ iterations.

Theorem 5.2.8 (Importance sampling & convex case)**.** *Suppose Assumption (LCi2), (C) and (BL) hold. Randomly choose i_k with importance sampling independently from previous choice. For all $k > 0$, we have*

$$\mathbb{E}[f(x^k)] - f^* \leq \frac{2\ell R_0^2}{k}, \tag{5.24}$$

To achieve accuracy ϵ, we need $k = O\left(\frac{\ell}{\epsilon}\right)$ iterations.

When we apply importance sampling, it is required to known the Lipschitz constants in each subspace $\mathbb{R}^{n_i}$.

5.2.3 Randomized Gauss-Seidel Method

To solve linear equation

$$Ax = b,$$

we can define corresponding minimization problem as least squares problem (5.25) mentioned in Chapter 1.

$$f(x) = \frac{1}{2}\|Ax - b\|^2. \tag{5.25}$$

In [48], Wright points out that CD highly related to standard Gauss Seidel method or Success Over Relaxation (SOR) method applying to normal equation. Now we give a brief derivation to show applying RCD to (5.25) is equivalently to applying randomized forward Gauss Seidel to normal equation (2.2).

Apply RCD to (5.25). For simplification, we denote the random variable index with i instead of i_k. Let $\alpha_k = \frac{1}{\|a^i\|^2}$. The update form is

$$\begin{aligned}
x^{k+1} &= x^k - \alpha_k[\nabla f(x^k)]_i \\
&= x^k - \frac{1}{\|a^i\|^2}[A^\intercal(Ax^k - b)]_i \\
&= x^k + \frac{1}{\|a^i\|^2}[A^\intercal b - A^\intercal Ax^k]_i \\
&= x^k + \frac{1}{[A^\intercal A]_{ii}}[A^\intercal b - A^\intercal Ax^k]_i
\end{aligned}$$

which is the update formula when applying randomized Gauss Seidel to linear equation $A^\intercal Ax = A^\intercal b$. Here we use the fact that column of a is normalized, i.e.,

$$[A^\intercal A]_{ii} = (a^i)^\intercal a^i = 1.$$

Wright also mentioned in [48] that applying RCD to linear system (5.25) update $w^k = A^\intercal x^k$ to the primal problem

$$\min_w \frac{1}{2}\|w\|_2^2 \text{ subject to } Aw = b. \tag{5.26}$$

This duality relation will be discussed in later section in this chapter.

5.2.4 Summary of Randomized Coordinate Descent Type Methods

The convergence results of GD, RCD, RBCD with uniform or importance sampling for convex or strongly-convex functions are listed in Table 5.1 which provides a good reference when we compare RFASD with CD type methods. To conclude,

- CD comes with the idea to reduce complexity of evaluate full gradient in huge-scale optimization problem.
- RCD is an improvement of CCD due to easy analysis of convergence and better convergence performance than cyclic version.
- RBCD is a block version of RCD with more components updated in each iteration, which is a balance of complexity and storage between CD and RCD.

Table 5.1: Examples: Number of iterations to achieve accuracy ϵ for the optimality gap.

	Convex	Strongly convex
GD	$O\left(\frac{L}{\epsilon}\right)$	$O\left(\frac{L}{\mu}\log\frac{1}{\epsilon}\right)$
RCD	$O\left(\frac{NL}{\epsilon}\right)$	$O\left(\frac{NL}{\mu}\log\frac{1}{\epsilon}\right)$
Uniform BCD $n = J$	$O\left(\frac{JL}{\epsilon}\right)$	$O\left(\frac{JL}{\mu}\log\frac{1}{\epsilon}\right)$
Importance BCD $n = J$	$O\left(\frac{\ell}{\epsilon}\right)$	$O\left(\frac{\ell}{\mu}\log\frac{1}{\epsilon}\right)$

5.3 Stochastic Gradient Descent Type Methods

To discuss stochastic gradient descent type methods including stochastic gradient descent method (SGD), mini-batch stochastic gradient descent method (mini-batch SDG) and stochastic Newton method or quasi Newton method, we follow closely the notations in review paper [8]. SGD is another updated version of GD from the perspective that compute gradient is consuming in many machine learning or deep learning problems. It updates gradient corresponds to one sample of data, i.e., (x_i, y_i) instead of gradient evaluated on all data sets. Before introduce the detail of SGD, we have assumptions of objective function f. Suppose objective function is either expectation risk or empirical risk, i.e.

$$f(w) = \begin{cases} R(w) = \mathbb{E}[f(w,\xi)] \\ R_m(w) = \frac{1}{m}\sum_{i=1}^{m} f_i(w), \end{cases} \tag{5.27}$$

where $R(w)$ is called expectation risk with ξ a random variable related to sample pair (x_{i_k}, y_{i_k}) and $R_m(w)$ is called empirical risk with i related to sample pair (x_i, y_i). Empirical risk can be viewed as a special case of expectation risk by viewing i a random variable with equal probability on value

$\{1, \cdots, m\}$, i.e.

$$P(i = j) = \frac{1}{m},\ j \in \{1, \cdots, m\}. \tag{5.28}$$

The update form can be defined as

$$w^{k+1} = w^k - \alpha_k g(w^k, \xi_k), \tag{5.29}$$

where

$$g(w^k, \xi_k) = \begin{cases} \nabla f(w^k, \xi_k) & (5.30) \\ \left(\frac{1}{n_k}\right) \sum_{i=1}^{n_k} \nabla f(w^k; \xi_{k,i}) & (5.31) \\ H_k \left(\frac{1}{n_k}\right) \sum_{i=1}^{n_k} \nabla f(w^k; \xi_{k,i}) & (5.32) \end{cases}$$

(5.30) corresponds to Stochastic Gradient Descent (SGD), (5.31) corresponds to mini-batch SGD and (5.32) corresponds to stochastic Newton or quasi Newton with n_k the size of mini-batch and H_k a symmetric definite scaling matrix.

For simplicity of discussion, we denote $\nabla f_i(x) = \nabla f(x, \xi_i)$, where ξ_i is a realization of random variable ξ.

5.3.1 Stochastic Gradient Descent Method

The algorithm of SGD is described as follows.

```
1: Choose $w^0$ and $k \leftarrow 0$;
2: for $k = 0, 1, \cdots$ do
3:     Choose index $i_k \in \{1, \cdots, m\}$ according to distribution $p$;
4:     Evaluate gradient $\nabla f_{i_k}(w^k)$ at sample pair $(x_{i_k}, y_{i_k})$;
5:     Update with step size $\alpha^k > 0$;
           $$w^{k+1} \leftarrow w^k - \alpha^k \nabla f_{i_k}(w^k);$$
           $$k \leftarrow k + 1;$$
6: end for
```

Algorithm 8: Stochastic Gradient Descent Algorithm

5.3.2 Mini-batch Stochastic Gradient Descent Method

The algorithm of mini-batch SGD is described as follows.

```
1: Choose $w^0$ and $k \leftarrow 0$;
2: for $k = 0, 1, \cdots$ do
3:     Choose index $\{1, \cdots, i_{n_k}\} \subset \{1, \cdots, m\}$ according to distribution $p$;
4:     Evaluate gradient $\nabla f_i(w^k)$ at sample pairs $\{(x_i, y_i)\}_{i=1}^{n_k}$;
5:     Update with step size $\alpha^k > 0$;
           $$w^{k+1} \leftarrow w^k - \alpha^k \sum_{i=1}^{n_k} \nabla f_i(w^k);$$
           $$k \leftarrow k + 1;$$
6: end for
```

Algorithm 9: Mini-batch Stochastic Gradient Descent Algorithm

5.3.3 Convergence Results of Stochastic Gradient Descent Type Methods

Now to come to discuss the convergence of SGD methods. Before that, we recall basic assumptions as we have had for CD Type methods.

- (LC) The first order derivative of f is Lipschitz continuous with Lipschitz constant L, i.e.,

$$\|\nabla f(u) - \nabla f(v)\|_{\mathcal{V}'} \le L\|u - v\|_{\mathcal{V}}, \quad \forall\, u,\, v \in \mathcal{V}.$$

- (LCi) The first order derivative of f_i is Lipschitz continuous with Lipschitz constant L_i, i.e.,

$$\|\nabla f_i(u) - \nabla f_i(v)\|_{\mathcal{V}'} \le L_i\|u - v\|_{\mathcal{V}}, \quad \forall\, u,\, v \in \mathcal{V}.$$

- (SC) f is strongly convex with strong convexity constant $\mu > 0$, i.e.,

$$\langle \nabla f(u) - \nabla f(v), u - v \rangle \geq \mu \|u - v\|_{\mathcal{V}}^2, \quad \forall\, u,\, v \in \mathcal{V}.$$

Various discussion about convergence of SGD can be found in different scenarios. For simplicity, we lists the convergence results of applying SGD to strongly convex function f with fixed step size here. Suppose w^* is minimum point achieved in minimization problem (5.27).

Definition 5.3.1. *The residual of f_i at minimum point is*

$$\sigma^2 = \mathbb{E}[\nabla f_i(w^*)]^2 \tag{5.33}$$

If ∇f_i, $i \sim p$ with some distribution p is unbiased estimate of ∇f, we have σ^2 is variance of f_i at minimum point, i.e.,

$$\mathrm{Var}_i(f_i(w^*)) = \mathbb{E}_i[\|f_i(w^*)\|^2] - \|\,\mathbb{E}_i[f_i(w^*)]\|^2.$$

Let $\epsilon_0 = \|f(w^0) - f(w^*)\|$ and $\epsilon = \|f(x) - f(w^*)\|$ be the distance between initial point to minimum point and distance between current point to minimum point respectively. Bach and Moulines prove convergence rate depends on $\frac{EL_i^2}{\mu^2}$. In order to achieve accuracy ϵ, the expected steps is at least $k = 2\log(\epsilon/\epsilon_0)\left(\frac{EL_i}{\mu^2} + \frac{\sigma^2}{\mu^2\epsilon}\right)$ with a more general setting [28]. Deann et al prove a convergence rate with a linear dependence on the uniform bound $\sup(L_i/\mu)$ via uniform sampling strategy. They bring the idea of exponential convergence of randomized kaczmarz [42] and prove linear convergence depending on expected bound $\mathbb{E}[L_i/\mu]$ via importance sampling. The steps to achieve accuracy ϵ is given by $k = 2\log(2\epsilon_0/\epsilon)\left(\frac{\sup L_i}{\mu} + \frac{\sigma^2}{\mu^2\epsilon}\right)$ with uniform sampling and $k = 2\log(\epsilon_0/\epsilon)\left(\frac{\mathbb{E}\,L_i}{\mu} + \frac{\mathbb{E}\,L_i}{\inf L_i}\frac{\sigma^2}{\mu^2\epsilon}\right)$. Importance sampling reduces the dependence on smoothness term to $\frac{\mathbb{E}\,L_i}{\mu}$ but increases dependence on residual term. A way to balance this is use re-weighted sampling with the cost of biased estimate of gradient ∇f. More generally, when update $g(x;\xi_i)$ is not an unbiased estimate for gradient $\nabla f(x)$, Léon et al give convergence analysis under first and second moment limit of $g(x;\xi_i)$ for strongly convex function f, which also shows a linear dependence of $\mathbb{E}[L_i]/\mu$ in expectation sense [9].

5.3.4 Randomized Kaczmarz Method

Objective function in least squares problem (2.1) can be viewed as an empirical risk function, i.e.,

$$f(w) = \frac{1}{2}\|Aw - b\|_2^2 = \frac{1}{2m}\sum_{i=1}^{m} m\left(\langle a_i, x\rangle - b_i\right)^2. \tag{5.34}$$

Denote $f_i(w) = \frac{m}{2}\left(\langle a_i, w\rangle - b_i\right)^2$, $i \in \{1, \cdots, m\}$. The gradient of each f_i is

$$\nabla f_i(w) = m\left(\langle a_i, w\rangle - b_i\right) a_i$$

Apply SGD to (5.34), we have update as

$$\begin{aligned} w^{k+1} &= w^k - \alpha_k \nabla f_i(w^k) \\ &= w^k - \alpha_k m \left(\langle a_i, w^k\rangle - b_i\right) a_i \end{aligned}$$

Let $\alpha_k = \frac{1}{m\|a_i\|^2}$ and do sampling with distribution proportional to norm of each row $\|a_i\|^2$. It gives update of randomized kaczmarz method [41].

$$w^{k+1} = w^k - \left(\langle a_i, w^k\rangle - b_i\right)\frac{a_i}{\|a_i\|^2}.$$

In Deanna et al' s paper [30], they show the connection between SGD and RK with uniform sampling and importance sampling and bring the sampling strategy from RK to SGD to improve convergence rate and find a trade-off between dependence on smoothness term $\mathbb{E}[L_i]/\mu$ and residual term σ^2 defined in (5.33).

To conclude, SGD type methods require the objective function has the form of empirical risk or expectation risk 5.27. For each update, a single f_i can be chosen or a batch os f_i can be chosen. The batch size is an parameter to tune in implementation. Step size can be chosen either to be fixed with pre knowledge of objective function or diminishing. Sampling strategy can be either uniform sampling, importance sampling or combination of these two. Convergence of SGD depends linearly on condition number of problem $\frac{\mathbb{E} L_i}{\mu}$ or $\frac{\sup L_i}{\mu}$. There is a trade-off between smoothness term and residual term, which gives hint when choosing sampling strategy. Apply SGD to least squares problem (1.1) gives RK.

5.4 Duality

5.4.1 Randomized Kaczmarz Methods and Randomized Gauss Seidel Methods

Theorem 5.4.1. *Applying Randomized Gauss-Seidel methods (RGS) to* (5.35)

$$AA^{\intercal}w = b, \ \textit{where}\ A^{\intercal}w = x. \tag{5.35}$$

is mathematically equivalent to applying randomized kaczmarz methods (RK) to (1.1).

Proof. By applying RGS, we have update

$$w^{k+1} = w^k + (A^{\intercal}A)^{-1}_{ii}(b_i - (AA^{\intercal}w^k)_i)e_i. \tag{5.36}$$

Multiply both sides by $A^\intercal$, we have

$$\begin{aligned}
A^\intercal w^{k+1} &= A^\intercal w^k + (AA^\intercal)^{-1}_{ii}(b_i - (AA^\intercal w^k)_i)a_i \\
x^{k+1} &= x^k + \frac{1}{\|a_i\|^2}(b_i - \langle a_i, x^k\rangle)a_i
\end{aligned}$$

□

Hefny et al [17] mention RK selects row in each update while RGS selects column in each update. Also, using RK to equation $A\alpha = b$ with $A = XX^\intercal$ and $b = y$ can is the same as using RGS to equation $A'\beta = b'$ with $A' = X^\intercal X$ and $b = X^\intercal y$ with a primal-dual mapping $\beta = X^\intercal \alpha$. They also give suggestions to use RGS when $m > n$ (num of rows greater than columns) and RK when $n > m$ (number of columns greater than rows).

5.4.2 Randomized Coordinate Descent Methods and Stochastic Gradient Descent Methods

A popular optimization problem in machine learning is empirical risk minimization (ERM). We introduce the problem following setting in [14]. Let $X \in \mathbb{R}^{d\times n}$ be the data set and each row x_i, $i \in 1, \cdots, d$ corresponds to one data with n features. Let ϕ_j be associated loss function of x_i. Let λ be a positive regularized parameter. The primal formula of L2-regularized ERM problem is

$$\min_{w\in\mathbb{R}^d} \left(P(w) := \frac{1}{n}\sum_{j=1}^{n} \phi_j(\langle x_j, w\rangle) + \frac{\lambda}{2}\|w\|_2^2 \right), \tag{5.37}$$

The dual formula of (5.37) is

$$\max_{\alpha\in\mathbb{R}^n} \left(D(\alpha) := -\frac{1}{2\lambda n}\|X\alpha\|^2 - \frac{1}{n}\sum_{j=1}^{n} \phi_j^*(-\alpha_j) \right), \tag{5.38}$$

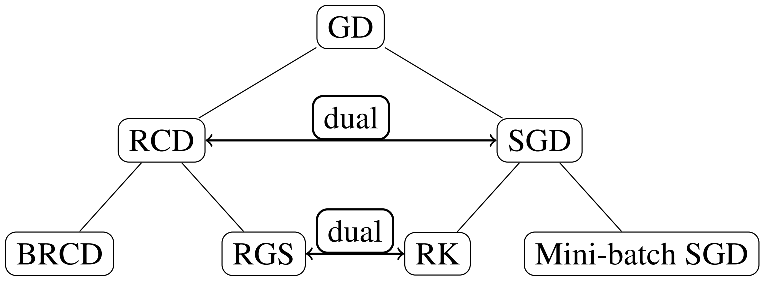


Figure 5.1: Conclusions of Gradient Type Methods

where $\phi_j^*(s) := \sup\{st - \phi_j(t) : t \in \mathbb{R}\}$ is convex conjugate of ϕ_j in the Fenchel dual problem.

We can apply SGD to solve primal problem (5.37) while apply RCD for dual problem (5.38), which is analogous to linear case described in Section 5.4.1. There is an issue when applying RCD to (5.38), ϕ_j^* are not necessary smooth. In this situation, we can use a proximal variance of RCD [14].

More discussions about solving dual problems can be found in [38, 44, 39, 19, 13]

In this chapter, we review gradient descent methods (GD), randomized coordinate descent type methods (RCD), stochastic gradient descent type methods (SGD). RC and SGD are two fundamental variants of GD when the computational complexity is not easy to handle. RCD or Block RCD (BRCD) updates with one of few components of gradient while SGD or mini-batch SGD update with one of few data. SGD requires the objective function to be in the form of expectation. One realization of random variable index i or ξ_i corresponds to one instance $\{x_i, y_i\}$ in data sets. Apply RCD to linear equation (1.1) gives Randomized Gauss-Seidel Methods (RGD) while apply SGD to linear equation (1.1) gives Randomized Kaczmarz methods (RK). RGD and RK are connected while applying to equivalent linear equations and SGD is mathematically equivalent to RCD when applying to dual problems. Here is a graph to conclude this chapter.

Chapter 6

Randomized Fast Subspace Descent Methods

In this chapter, we present an optimization method – randomized fast subspace descent method RFASD) for general non-constraint minimization problem with smooth and convex objective function. This method is an improvement of randomized coordinate descent (RCD).

Let $\mathcal{V} = \sum_i \mathcal{V}_i$ be a space decomposition. Instead of do gradient descent on one or few coordinates, we do gradient descent on some subspace of $\mathcal{V}$. With a proper space decomposition, the local problem will be of small size and have a small condition number so that preconditioned coordinate descent method inside the subspace is fast and stable.

Consider non-constraint minimization problem

$$\min_{x\in\mathcal{V}} f(x), \tag{6.1}$$

where f is a smooth and convex function and its derivative is Lipschitz continuous with constant L and $\mathcal{V}$ is a Hilbert space. In practice, $\mathcal{V} = \mathbb{R}^N$ but might be assigned with an inner products other than the standard l^2 inner product. Solving minimization problem (1.2) is a central task with wide applications in fields of scientific computing, machine learning and data science etc.

Coordinate descent methods decompose $\mathcal{V}$ by coordinates, i.e., $\mathcal{V} = \oplus\{e_i\}_{i=1}^N$. Dividing $\mathcal{V}$ by coordinates is not the only way to decompose a space. Multigrid method [43] provide a multilevel space decomposition. With a proper space decomposition, the local problem will have a small condition number so that preconditioned gradient descent method inside the subsapce is fast and stable. With the hierarchical structure of multilevel subspace decomposition, the condition number is also reduced and leads to faster convergence rate.

In [11], Chen, Hu and Wise borrow the idea of multigrid methods for solving the nonlinear equations to develop fast subspace descent (FASD) methods. In this chapter, we provide a general scheme for randomized version of fast subspace descent (RFASD) method.

We assume Lipschitz continuous (LC) and strong convexity (SC) of f on $\mathcal{V}$. Suppose there exists a space decomposition: $\mathcal{V} = \mathcal{V}_1 + \mathcal{V}_2 + \cdots \mathcal{V}_J$, with $\mathcal{V}_i \subset \mathcal{V}$, $i = 1, \cdots, J$. And f restricted on subspace $\mathcal{V}_i$ satisfies Lipschitz continuous condition (6.3).

For each subspace $\mathcal{V}_i$, we assign a new inner product given by a symmetric and positive definite matrix A_i with eigenvalue bound

$$\mu_i(v_i, v_i)_{\mathcal{V}} \leq (A_i v_i, v_i) \leq \lambda_i(v_i, v_i)_{\mathcal{V}}, \ \forall v_i \in \mathcal{V}_i. \tag{6.2}$$

An outline of RFASD is as follows. Randomly choose a subspace $\mathcal{V}_{i_k}$ according to some sampling distribution. Compute subspace search direction s_{i_k} such descent direction (6.6) and approximate gradient conditions (6.7).

Choose the step size $\alpha^k = \frac{\delta \mu_{i_k}}{L_{i_k}}$ and update via subspace correction

$$x^{k+1} = x^k + \alpha^k s_{i_k}.$$

The sampling distribution can be either uniform or non-uniform with probability proportional to the local Lipschitz constant L_i.

Denote $\ell = \sum_{i=1}^{J} L_i$ and $\ell_S = \sum_{i=1}^{J} \sqrt{L_i}$. $\kappa_{\max} = \max_i \kappa_i$.

RFASD achieves linear convergence for strongly convex function and sub-linear convergence for convex function with complexity $O\left(\frac{\gamma^2}{\delta^2}\frac{JL}{\epsilon}\kappa_{\max}\right)$ and $O\left(\frac{\gamma^2}{\delta^2}\frac{JL}{\mu}\kappa_{\max}\log\frac{1}{\epsilon}\right)$ with uniform sampling and $O\left(\frac{\gamma^2}{\delta^2}\frac{\ell}{\epsilon}\kappa_{\max}\right)$, $O\left(\frac{\gamma^2}{\delta^2}\frac{\ell}{\mu}\kappa_{\max}\log\frac{1}{\epsilon}\right)$ with non-uniform sampling. The complexity depends on three factors: $\frac{\gamma^2}{\delta^2}$ measures how good the descent direction approximates negative gradient direction, $\frac{L}{\mu}$ is the condition number of global space $\mathcal{V}$ and $\kappa_{\max} = \max_i \kappa_i$ where $\kappa_i = \frac{L_i}{\mu_i}$ is the condition number of subspace $\mathcal{V}_i$. Viewed in A norm, the global condition number $\frac{L}{\mu}$ might be reduced.

Construct a stable space decomposition

- (SD) Stable decomposition: there exists a constant $C_A > 0$, such that

$$\forall v \in \mathcal{V}, \quad v = \sum_{i=1}^{N} v_i, \quad \text{and} \quad \sum_{i=1}^{N} \|v_i\|_{\mathcal{V}}^2 \leq C_A \|v\|_{\mathcal{V}}^2.$$

Then $\frac{\gamma^2}{\delta^2}$ is bounded by C_A. Subspace condition number is controlled as $O(1)$ via subspace preconditioner A_i. Numerical experiments of RFASD with preconditioner introduced in Section 6.4.3 (pre-RFASD) and RCD are provided on Nesterov's worst function [32]. pre-RFASD converges uniformly as dimension of $\mathcal{V}$ increases while iteration steps of RCD increases tremendously.

We further develop two acceleration methods for RFASD. AFASD-I is for case $(\sum_{i=1}^{J} \mu_i s_i, v)_{\mathcal{V}} = \langle -\nabla f(x^k), v\rangle$. In this situation, Nesterov acceleration can be applied with a different nonuniform sampling strategy [33]. A modified descent direction (6.25) is required. AFASD-I achieves complexity bound $O(\frac{\ell_S}{\sqrt{\mu}}\log\frac{1}{\epsilon})$ for strongly convex function and $O(\frac{\ell_S}{\sqrt{\epsilon}})$ where $\ell_S = \sum_{i=1}^{J} \sqrt{L_i}$, which shows an improvement with RFASD.

As mentioned in [21], a method may be accelerated if it has linear convergence rate for strongly convex problems. We introduced the Catalyst acceleration of RFASD (AFASD-II), which uses RFASD in an inner-loop to solve a regularized optimization problem and Nestrov acceleration in an outer-loop. Convergence analysis and complexity estimate are provided. By choosing the optimal parameter, AFASD-II achieves optimal complexity $\tilde{\mathcal{O}}\left(\sqrt{\frac{L}{\mu}}\log\frac{1}{\epsilon}\right)$ for strongly convex function and $\tilde{\mathcal{O}}\left(\sqrt{\frac{L}{\epsilon}}\log\frac{1}{\epsilon}\right)$ for convex function. Here a log factor dependent on parameters is hidden in the notation $\tilde{\mathcal{O}}$.

6.1 Assumptions

We consider the minimization problem (1.2) with problem settings in Chapter 1. Furthermore, we require a space decomposition and Lipschitz continuous on each subspace.

Suppose we have a decomposition of space

$$\mathcal{V} = \mathcal{V}_1 + \mathcal{V}_2 + \cdots + \mathcal{V}_J, \quad \mathcal{V}_i \subset \mathcal{V}, \quad i = 1, \cdots, J,$$

with the following assumption:

- (LCi) Lipschitz continuous restricted in each subspace with Lipschitz constant L_i, i.e.,

$$\|\nabla f(x + v_i) - \nabla f(x)\|_{\mathcal{V}'} \leq L_i \|v_i\|_{\mathcal{V}}, \quad \forall\, v_i \in \mathcal{V}_i \tag{6.3}$$

For each subspace $\mathcal{V}_i$, we assign a SPD matrix A_i. The equivalent relation between ℓ_2 norm and A_i norm is described in following inequality.

- (SE) In subspace $\mathcal{V}_i$, ℓ_2-norm and A_i-norm are equivalent, i.e., there exist constants $\lambda_i, \mu_i, i =$

$1, \cdots, J$ such that

$$\mu_i (v_i, v_i)_{\mathcal{V}} \le (A_i v_i, v_i) \le \lambda_i (v_i, v_i)_{\mathcal{V}},\ \forall v_i \in \mathcal{V}_i.$$

6.2 Algorithms

We propose the randomized fast subspace descent (RFASD) algorithm (Algorithm 10).

```
1: Choose $x^0$ and $k \leftarrow 0$
2: **for** $k = 0, 1, \cdots$ **do**
3:     Choose an index of subspace $i_k$ from $\{1, \cdots, J\}$
4:     Compute a subspace search direction $s_{i_k} \in \mathcal{V}_{i_k}$ based on $x^k$
5:     Update the subspace correction: $\alpha^k > 0$
           $$x^{k+1} := x^k + \alpha^k s_{i_k}.$$
6: **end for**
```

Algorithm 10: Randomized Fast Subspace Descent Method

There are two candidates for distribution p used in the sampling step 3.

- **Uniform Sampling**. The probability of choosing i-th subspace is

$$p_i = \frac{1}{J}, \quad i = 1, 2, \ldots, J. \tag{6.4}$$

- **Importance Sampling** (with distribution proportional to subspace Lipschitz numbers). The probability of choosing i-th subspace is

$$p_i = \frac{L_i}{\ell}, \quad i = 1, 2, \ldots, J, \tag{6.5}$$

where $\ell = \sum_{i=1}^{J} L_i$.

6.3 Convergence Analysis of RFASD

In this section, we discuss the convergence of RFASD. We consider the randomized version with uniform and non-uniform sampling. We assume each A_i satisfies (SE) with constants μ_i and λ_i. In RFASD Algorithm 10, we have freedom to choose the search direction s_i, $i = 1, 2, \cdots, J$, at each iteration. In general, it should approximate the gradient in certain sense in order to provide a convergent algorithm. Therefore, we make the following assumptions:

- (DD) Descent direction: let s_i be computed using x^k and satisfies

$$\langle -\nabla f(x^k), \sum_{i=1}^{J} s_i \rangle \geq \delta \left(\sum_{i=1}^{J} \mu_i \|s_i\|_{A_i}^2 \right) \geq 0 \quad \text{with some } \delta > 0 \tag{6.6}$$

- (AP) Approximate gradient: there exists a constant γ such that

$$\|\nabla f(x^k)\|_{\mathcal{V}'} \leq \gamma \left(\sum_{i=1}^{J} \lambda_i \|s_i\|_{A_i}^2 \right)^{\frac{1}{2}} \tag{6.7}$$

6.3.1 Linear convergence rate for strongly convex functions

We start with the case that f is strongly convex, i.e., Assumption (A2) holds, and show that RFASD convergence linearly with properly chosen step size α^k for uniform sampling (assume we do not know L_i).

Theorem 6.3.1 (Uniform sampling & strongly convex case)**.** *Suppose Assumption (LC) and (SC) hold. Assume the search direction s_i satisfies (DD) and (AP) for $i = 1, 2, \cdots, J$. Furthermore, assume index i_k is chosen uniformly and $\alpha = \frac{\mu_{i_k}\delta}{L}$, then for all $k > 0$, we have*

$$\mathbb{E}\left(f(x^k)\right) - f^* \leq \left(1 - \frac{\mu}{JL}\frac{\delta^2}{\gamma^2}\frac{1}{\kappa_{\max}}\right)^k \left(f(x^0) - f^*\right), \tag{6.8}$$

where $\kappa_i = \frac{\lambda_i}{\mu_i}$ is the condition number of subspace $\mathcal{V}_i$ and $\kappa_{\max} = \max \kappa_i$.

Proof. By the Lipschitz continuity (LC) and subspace norm equivalence (SE) , we have

$$\begin{aligned} f(x^{k+1}) &\leq f(x^k) + \alpha\langle\nabla f(x^k), s_{i_k}\rangle + \frac{L}{2}\alpha^2\|s_{i_k}\|^2_{\mathcal{V}} \\ &\leq f(x^k) - \alpha\langle -\nabla f(x^k), s_{i_k}\rangle + \frac{L}{2\mu_{i_k}}\alpha^2\|s_{i_k}\|^2_{A_{i_k}} \end{aligned}$$

Choose $\alpha = \frac{\delta\mu_{i_k}}{L}$, we have

$$f(x^{k+1}) \leq f(x^k) - \frac{\delta\mu_{i_k}}{L}\langle -\nabla f(x^k), s_{i_k}\rangle + \frac{\delta^2\mu_{i_k}}{2L}\|s_{i_k}\|^2_{A_{i_k}}.$$

Take conditional expectation conditioned by x^k with probability $p_i = \frac{1}{J}$, $j = 1, \cdots, J$.

$$\begin{aligned} \mathbb{E}\left(f(x^{k+1})\right) &\leq f(x^k) - \frac{\delta}{JL}\langle -\nabla f(x^k), \sum_{i=1}^J \mu_i s_i\rangle + \frac{\delta^2}{2JL}\sum_{i=1}^J \mu_i\|s_i\|^2_{A_i} \\ \text{Assumption (DD)} &\leq f(x^k) - \frac{\delta^2}{2JL}\sum_{i=1}^J \mu_i\|s_i\|^2_{A_i} \\ &\leq f(x^k) - \frac{\delta^2}{2JL}\frac{1}{\kappa_{\max}}\sum_{i=1}^J \lambda_i\|s_i\|^2_{A_i} \\ \text{Assumption (AP)} &\leq f(x^k) - \frac{\delta^2}{2JL\gamma^2}\frac{1}{\kappa_{\max}}\|\nabla f(x^k)\|^2_{\mathcal{V}'}. \end{aligned} \tag{6.9}$$

From strong convexity (SC), we have $f(x^k) - f^* \leq \frac{1}{2\mu}\|\nabla f(x^k)\|^2_{\mathcal{V}'}$. Use this inequality and subtract f^* from both sides of the above inequality, we have

$$\mathbb{E}\left(f(x^{k+1})\right) - f^* \leq \left(1 - \frac{\mu}{JL}\frac{\delta^2}{\gamma^2}\frac{1}{\kappa_{\max}}\right)\left(f(x^k) - f^*\right).$$

By taking expectation with respect to x^k on both side, we obtain (6.8). $\square$

The convergence rate $1 - \frac{\mu}{JL}\frac{\delta^2}{\gamma^2}\frac{1}{\kappa_{\max}}$ is determined by 3 terms where

- $\frac{\mu}{JL} = \frac{1}{J\kappa}$ contains information of $\kappa = \frac{L}{\mu}$ the condition number of global problem and J the number of subspaces,
- $\frac{\delta^2}{\gamma^2}$ measures how good $\sum_i^J s_i$ approximate $\nabla f(x^k)$,
- $\frac{1}{\kappa_{\max}}$ is determined by condition number of subspace $\mathcal{V}_i$.

Here is an estimate of $\frac{\delta}{\gamma}$.

$$\begin{aligned}
|\langle -f(x^k), \sum_{i=1}^J s_i\rangle| &\leq \|f(x^k)\|_{\mathcal{V}'}\|\sum_{i=1}^J s_i\|_{\mathcal{V}} \\
&\leq \|f(x^k)\|_{\mathcal{V}'}\sum_{i=1}^J \|s_i\|_{\mathcal{V}} \\
&\leq \|f(x^k)\|_{\mathcal{V}'}\sqrt{J}\left(\sum_{i=1}^J \|s_i\|_{\mathcal{V}}^2\right)^{\frac{1}{2}} \\
&\leq \|f(x^k)\|_{\mathcal{V}'}\sqrt{J}\left(\sum_{i=1}^J \lambda_i\|s_i\|_{A_i}^2\right)^{\frac{1}{2}}
\end{aligned}$$

By (AP) and (DD)

$$\delta\left(\sum_{i=1}^J \mu_i\|s_i\|_{A_i}^2\right) \leq \gamma\left(\sum_{i=1}^J \lambda_i\|s_i\|_{A_i}^2\right)^{\frac{1}{2}}\sqrt{J}\left(\sum_{i=1}^J \lambda_i\|s_i\|_{A_i}^2\right)^{\frac{1}{2}}$$

$$\frac{\delta}{\gamma} \leq \sqrt{J}\frac{\sum_{i=1}^{J}\lambda_i\|s_i\|_{A_i}^2}{\sum_{i=1}^{J}\mu_i\|s_i\|_{A_i}^2} \leq \sqrt{J}\kappa_{\max}.$$

Similarly, we can analyze the convergence of RFASD when i_k is sampled non-uniformly if subspace Lipschitz constant L_i provided. The result is presented in the following theorem.

Theorem 6.3.2 (Importance sampling & strongly convex case)**.** *Suppose Assumption (SC) and (LCi) hold. Assume the search direction s_i satisfies (DD) and (AP) for $i = 1, 2, \cdots, N$. Furthermore, assume index i_k is chosen non-uniformly with probability $\frac{L_{i_k}}{\sum_{i=1}^{J} L_i}$ and $\alpha^k = \frac{\delta\mu_{i_k}}{L_{i_k}}$, then for all $k > 0$, we have*

$$\mathbb{E}\left(f(x^k)\right) - f^* \leq \left(1 - \frac{\mu}{\ell}\frac{\delta^2}{\gamma^2}\frac{1}{\kappa_{\max}}\right)^k \left(f(x^0) - f^*\right), \tag{6.10}$$

where $\ell = \sum_{i=1}^{J} L_i$, $\kappa_i = \frac{\lambda_i}{\mu_i}$ is the condition number of subspace $\mathcal{V}_i$ and $\kappa_{\max} = \max \kappa_i$.

Proof. By the Lipschitz continuity on each subspace (LCi) and subspace norm, we have

$$\begin{aligned} f(x^{k+1}) &\leq f(x^k) + \alpha\langle\nabla f(x^k), s_{i_k}\rangle + \frac{L}{2}\alpha^2\|s_{i_k}\|_{\mathcal{V}}^2 \\ &\leq f(x^k) - \alpha\langle -\nabla f(x^k), s_{i_k}\rangle + \frac{L}{2\mu_{i_k}}\alpha^2\|s_{i_k}\|_{A_{i_k}}^2 \end{aligned}$$

Choose $\alpha = \frac{\delta\mu_{i_k}}{L_{i_k}}$, we have

$$f(x^{k+1}) \leq f(x^k) - \frac{\delta\mu_{i_k}}{L_{i_k}}\langle -\nabla f(x^k), s_{i_k}\rangle + \frac{\delta^2\mu_{i_k}}{2L_{i_k}}\|s_{i_k}\|_{A_{i_k}}^2.$$

Take conditional expectation conditioned by x^k with probability $p_i = \frac{L_i}{\ell}$, $j = 1, \cdots, J$.

$$\begin{aligned}
\mathbb{E}\left(f(x^{k+1})\right) &\le f(x^k) - \frac{\delta}{\ell}\langle -\nabla f(x^k), \sum_{i=1}^{J} \mu_i s_i\rangle + \frac{\delta^2}{2\ell}\sum_{i=1}^{J}\mu_i \|s_i\|_{A_i}^2 \\
\text{Assumption (DD)} &\le f(x^k) - \frac{\delta^2}{2\ell}\sum_{i=1}^{J}\mu_i\|s_i\|_{A_i}^2 \\
&\le f(x^k) - \frac{\delta^2}{2\ell}\frac{1}{\kappa_{\max}}\sum_{i=1}^{J}\lambda_i\|s_i\|_{A_i}^2 \\
\text{Assumption (AP)} &\le f(x^k) - \frac{\delta^2}{2\ell\gamma^2}\frac{1}{\kappa_{\max}}\|\nabla f(x^k)\|_{\mathcal{V}'}^2. \qquad (6.11)
\end{aligned}$$

From strong convexity (SC), we have $f(x^k) - f^* \le \frac{1}{2\mu}\|\nabla f(x^k)\|_{\mathcal{V}'}^2$. Use this inequality and subtract f^* from both sides of the above inequality, we have

$$\mathbb{E}\left(f(x^{k+1})\right) - f^* \le \left(1 - \frac{\mu}{\ell}\frac{\delta^2}{\gamma^2}\frac{1}{\kappa_{\max}}\right)\left(f(x^k) - f^*\right).$$

By taking expectation with respect to x^k on both side, we obtain (6.8). □

With non-uniform sampling, convergence rate also contains 3 terms including $\frac{\mu}{\ell}$, how good $\sum_i^J s_i$ approximates $\nabla f(x^k)$ and local condition number. Since $\ell = \sum_{i=1}^J L_i \le NL$, we have

$$1 - \frac{\mu}{\ell}\frac{\delta^2}{\gamma^2}\frac{1}{\kappa_{\max}} \le 1 - \frac{\mu}{JL}\frac{\delta^2}{\gamma^2}\frac{1}{\kappa_{\max}},$$

which means that non-uniform sampling is better then uniform sampling. Of course, in practice, Lipschitz constants L_i might not known which makes non-uniform sampling difficult to use.

6.3.2 Sublinear convergence for convex functions

Next, we give the convergence result without strong convexity, i.e. $\mu = 0$ in Assumption (A2). The following assumption is needed.

- (BL) Bounded level set: f is convex and uniformly Lipschitz continuously differentiable, and attains its minimum value f^* on a set S. There is a finite constant R_0 such that the level set for f defined by x^0 is bounded, that is,

$$\max_{x^* \in S} \max_{x} \{\|x - x^*\| : f(x) \leq f(x^0)\} \leq R_0. \tag{6.12}$$

We still use the same step size and show that RFASD converges sublinearly in this case.

Theorem 6.3.3 (Uniform sampling & convex case)**.** *Suppose the space decomposition satisfies (LC) and the search direction s_i satisfies (DD) and (AP) for $i = 1, 2, \cdots, J$. Assume index i_k is chosen uniformly and $\alpha = \frac{\delta \mu_{i_k}}{L}$, then for all $k > 0$, we have*

$$\mathbb{E}\left(f(x^k)\right) - f^* \leq \frac{2\gamma^2 J L \kappa_{\max} R_0^2}{\delta^2 k}, \tag{6.13}$$

where $\kappa_i = \frac{\lambda_i}{\mu_i}$ is the condition number of subspace $\mathcal{V}_i$ and $\kappa_{\max} = \max \kappa_i$.

Proof. From (6.9) and taking expectation with respect to x^k, we have

$$\mathbb{E}\left(f(x^{k+1})\right) - f^* \leq \mathbb{E}\left(f(x^k)\right) - f^* - \frac{\mu}{2JL}\frac{\delta^2}{\gamma^2}\frac{1}{\kappa_{\max}}\mathbb{E}\left(\|\nabla f(x^k)\|_{\mathcal{V}'}^2\right) \tag{6.14}$$

Denote $d^k = \mathbb{E}\left(f(x^k)\right) - f^*$ and note that

$$f(x^k) - f^* \leq \nabla f(x^k)^T (x^k - x^*) \leq \|\nabla f(x^k)\|_{\mathcal{V}'} \|x^k - x^*\|_{\mathcal{V}} \leq R_0 \|\nabla f(x^k)\|_{\mathcal{V}'}.$$

Then, by taking expectation of the above inequality and substitute it back into (6.14), we have

$$d^{k+1} \leq d^k - \frac{\delta^2}{2JL\gamma^2 \kappa_{\max} R_0^2}\left(d^k\right)^2.$$

Based on the above inequality, we obtain

$$\frac{1}{d^{k+1}} - \frac{1}{d^k} = \frac{d^k - d^{k+1}}{d^k d^{k+1}} \geq \frac{d^k - d^{k+1}}{\left(d^k\right)^2} \geq \frac{\delta^2}{2JL\gamma^2\kappa_{\max}R_0^2}.$$

Recursively apply this, we have

$$\frac{1}{d^k} \geq \frac{1}{d^0} + \frac{\delta^2 k}{2JL\gamma^2\kappa_{\max}R_0^2} \geq \frac{\delta^2 k}{2JL\gamma^2\kappa_{\max}R_0^2}$$

which implies (6.13). □

Following the same argument, we can have the similar result for non-uniform sampling case as shown in the following theorem.

Theorem 6.3.4 (Importance sampling & Convex)**.** *Suppose the space decomposition satifies (SA) and the search direction s_i satisfies (DD) and (AP) for $i = 1, 2, \cdots, J$. Assume index i_k is chosen is chosen non-uniformly with probability $\frac{L_{i_k}}{\ell}$, and $\alpha^k = \frac{\delta\mu_{i_k}}{L_{i_k}}$, then for all $k > 0$, we have*

$$\mathbb{E}\left(f(x^k)\right) - f^* \leq \frac{2\gamma^2\ell\kappa_{\max}R_0^2}{\delta^2 k} \tag{6.15}$$

where $\ell = \sum_{i=1}^{J} L_i$ and $\kappa_{\max} = \max_i \frac{\lambda_i}{\mu_i}$ is the condition number of subspace $\mathcal{V}_i$.

Proof. The proof follows directly from (6.11) by using the same argument as the proof of Theorem 6.3.3. □

Again we have

$$\frac{2\gamma^2\ell\kappa_{\max}R_0^2}{\delta^2 k} \leq \frac{2\gamma^2 JL\kappa_{\max}R_0^2}{\delta^2 k},$$

which implies that the non-uniform sampling convergences faster than the uniform sampling case theoretically.

6.4 Examples of Randomized Fast Subspace Descent Methods

In this section, we consider several methods which can be viewed as specific examples of Randomized Fast Subspace Descent Methods.

6.4.1 Gradient Descent method

Let $\mathcal{V} = \mathbb{R}^N$ equipped with standard ℓ^2-norm and trivial space decomposition $\mathcal{V}_i = \mathbb{R}^N,\ i = 1$. Thus $J = 1$. Let $A_1 = I$, which induces $\mu_1 = \lambda_1 = 1$ and $s_i = -\nabla f(x^k)$. The Assumptions (DD) and (AP) can be checked

$$s_1 = -\nabla f(x^k) \text{ and } \|s_1\|^2 = \|\nabla f(x^k)\|^2,$$

which implies $\delta = 1$ and $\gamma = 1$. Set $\alpha = 1$. Since there is only one subspace, we do not need to apply sampling. Theorem 6.3.1 and 6.3.3 recovers the convergence properties of gradient descent.

- Convex : $O\left(\frac{2LR_0^2}{k}\right)$
- Strongly convex: $O\left(\left(1 - \frac{\mu}{L}\right)^k\right)$

Example 6.4.1. *Consider the Nestrov's 'worst' problem: for $x \in \mathbb{R}^N$,*

$$f_{L,r}(x) = \frac{L}{4}\left(\frac{1}{2}\left(x_1^2 + \sum_{i=1}^{r-1}(x_i - x_{i-1})^2 + x_r^2\right) - x_1\right), \tag{6.16}$$

where x_i represents the $i-th$ coordinate of x and $r < N$ is a constant integer that defines the intrinsic dimension of the problem [32]. The minimum value of the function is

$$f_* = \frac{L}{8}\left(-1 + \frac{1}{r+1}\right). \tag{6.17}$$

The condition number of each subspace $\mathcal{V}_i = \{x_i\}$ is $\mathcal{O}(r^2)$. Randomness is memory efficiency but does not contribute to convergence rate due to the ill-conditioning. Notice that the L, μ is measured in ℓ^2-norm and thus could be very large for ill-conditioned problem.

6.4.2 (Block) coordinate descent method

Let $\mathcal{V} = \mathbb{R}^N$ with standard ℓ^2-norm $\|\cdot\|$. Define a partition of the unit matrix

$$I_n = (U_1, \cdots, U_J) \in \mathbb{R}^{N\times N},\ U_i \in \mathbb{R}^{N\times n_i},\ i = 1, \cdots, J.$$

Now we consider the space decomposition $\mathcal{V} = \oplus_{i=1}^{J}\mathcal{V}_i$, where $\mathcal{V}_i = \mathrm{Range}(U_i)$ and $\sum_{i=1}^{J} n_i = N$. Suppose the decomposition satisfies (LCi). The search direction is given by

$$s_i = -U_i U_i^{\intercal} \nabla f(x^k).$$

Let $A_i = I_i$. Thus $\mu_i = \lambda_i = 1$ and $\kappa_{\max} = 1$. Again, it is easy to see that

$$\sum_{i=1}^{J} s_i = -\nabla f(x^k) \quad \text{and} \quad \sum_{i=1}^{J} \|s_i\|^2 = \|\nabla f(x^k)\|^2.$$

This implies Assumption (DD) and (AP) hold with $\delta = \gamma = 1$. Therefore, choosing stepsize $\alpha = 1$, Theorem 6.3.1, 6.3.2, 6.3.3, and 6.3.4 recovery the classical convergence analysis of block CD method [31] as follows,

- Uniform sampling & Convex : $O\left(\frac{2JLR_0^2}{k}\right)$
- Non-uniform sampling & Convex: $O\left(\frac{2\ell R_0^2}{k}\right)$
- Uniform sampling & Strongly convex: $O\left(\left(1 - \frac{\mu}{JL}\right)^k\right)$

- Non-uniform sampling & Strongly convex: $O\left(\left(1-\frac{\mu}{\ell}\right)^k\right)$

In [31], the author assumes f is coordinatewise Lipschitz continuous with constant L_i, i.e

$$\|f_i'(x_i + U_i h_i) - f_i'(x)\|_{(i)}^* \leq L_i \|h_i\|_{(i)}, \quad h_i \in \mathbb{R}^{n_i}, \quad i = 1, \cdots, J, x \in$$

The subspace is chosen with a random counter $\mathcal{A}_\beta, \beta \in \mathbb{R}$. It generates an integer number $i \in \{1, \cdots, n\}$ with probability

$$p_\beta^{(i)} = L_i^\beta \cdot \left[\sum_{j=1}^N L_j^\beta\right]^{-1}, \quad i = 1, \cdots, J.$$

Popular choices of β is $0, \frac{1}{2}, 1$. When $\beta = 0$, the distribution is uniform. When $\beta = 1$, the distribution is proportional to subspace Lipschitz constant L_j. The convergence result on convex function on Nestrov's paper [31] is

$$\mathbb{E}[f(x^k)] - f^* \leq \frac{2}{k+4} \cdot \left(\sum_{j=1}^n L_j^\beta\right) R_{1-\beta}^2(x_0). \tag{6.18}$$

For strongly convex function f, assume the convexity parameter is $\mu > 0$. Theorem 2 in [31] shows that the function f is strongly convex norm $\|\cdot\|_{[1-\beta]}$ with convexity parameter $\mu_{1-\beta} > 0$. And

$$\mathbb{E}[f(x^k)] - f^* \leq \left(1 - \frac{\mu_{1-\beta}}{\sum_{i=1}^n L_i^\beta}\right)(f(x_0) - f^*). \tag{6.19}$$

Let $n = J$ and $\beta = 0, 1$, Nestrov's results (6.18) and (6.19) is

- Uniform sampling & Convex : $O\left(\frac{2JR_1^2}{k+4}\right)$;
- Non-uniform sampling & Convex: $O\left(\frac{2\ell R_0^2}{k+4}\right)$;

- Uniform sampling & Strongly convex: $O\left(\left(1-\frac{\mu_1}{J}\right)^k\right)$;
- Non-uniform sampling & Strongly convex: $O\left(\left(1-\frac{\mu_0}{\ell}\right)^k\right)$,

where $\mu_0 = \mu$.

Since

$$\|x\|_{[1]} = \left(\sum_{i=1}^{N} L_i x_i^2\right)^{\frac{1}{2}} \leq (\max L_i)^{\frac{1}{2}} \left(\sum_{i=1}^{N} x_i^2\right)^{\frac{1}{2}} = L^{\frac{1}{2}} \|x\|_{[0]}. \tag{6.20}$$

and by definition in [31],

$$R_1(x_0) = \max_x \{ \max_{x_* \in X^*} \|x - x_*\|_{[1]} : f(x) \leq f(x_0) \}.$$

We have

$$R_1(x_0) \leq L^{\frac{1}{2}} R_0(x_0).$$

And by definition of strong convexity

$$\langle \nabla f(x) - \nabla f(y), x - y \rangle \geq \mu_1 \|x - y\|_{[1]}^2,$$

$$\langle \nabla f(x) - \nabla f(y), x - y \rangle \geq \mu_0 \|x - y\|_{[0]}^2 \geq \mu_0 \frac{1}{L} \|x - y\|_{[0]}^2.$$

Thus

$$\mu_1 \geq \frac{\mu_0}{L}.$$

Our results recovers Nestrov's results [31].

Remark 6.4.2. *When $J = N$, it is Coordinate Descent Method. As J decreases, the convergence is faster while the complexity in each update increases.*

6.4.3 Preconditioned RFASD

Consider space decomposition $\mathcal{V} = \mathcal{V}_1 + \mathcal{V}_2 + \cdots + \mathcal{V}_J$ and $\mathcal{V}_i \subset \mathcal{V}$ and assume this space decomposition is stable,

- (SD) Stable decomposition: there exists a constant $C_A > 0$, such that

$$\forall v \in \mathcal{V}, \quad v = \sum_{i=1}^{J} v_i, \quad \text{and} \quad \sum_{i=1}^{J} \|v_i\|_{\mathcal{V}}^2 \le C_A \|v\|_{\mathcal{V}}^2.$$

Let A_i be a SPD matrix on subspace $\mathcal{V}_i$ satisfying the assumption (SE). A_i is served as preconditioner.

The search direction s_i is chosen as follows,

$$A_i s_i = -R_i \nabla f(x^k),$$

wherer R_i is restriction operator $\mathcal{V}' \to \mathcal{V}_i'$. Let $I_i : \mathcal{V}_i \mapsto \mathcal{V}$ be the natural inclusion and then $R_i = I_i^\intercal$ since $(R_i^\intercal v_i, v) = (v_i, R_i v) = (v_i, v) = (I_i v_i, v), \quad \forall v_i \in \mathcal{V}_i, v \in \mathcal{V}$.

To verify (DD), note that

$$\langle -\nabla f(x^k), \mu_i s_i \rangle = \mu_i \|s_i\|_{A_i}^2$$

Summing over i,

$$\langle -\nabla f(x^k), \sum_{i=1}^{J} \mu_i s_i \rangle = \sum_{i=1}^{J} \mu_i \|s_i\|_{A_i}^2.$$

Therefore Assumption (DD) holds with $\delta = 1$.

Lemma 6.4.3. *Assume the space decomposition $\mathcal{V} = \sum_{i=1}^{J} \mathcal{V}_i$ satisfy the stable decomposition assumption (SD). Let $g = -\nabla f(x^k), g_i = R_i g$ for $i = 1, 2, \ldots, N$. Then*

$$\|\nabla f(x^k)\|_{\mathcal{V}'}^2 \le C_A \sum_{i=1}^{J} \|g_i\|_{\mathcal{V}_i'}^2. \tag{6.21}$$

Proof. For any $w \in \mathcal{V}$, we chose a stable decomposition $w = \sum_{i=1}^J w_i, w_i \in \mathcal{V}_i$. Then

$$\begin{aligned}\langle \nabla f(x^k), w\rangle &= \sum_{i=1}^J \langle \nabla f(x^k), w_i\rangle = \sum_{i=1}^J \langle g_i, w_i\rangle \\ &\leq \left(\sum_{i=1}^J \|g_i\|_{\mathcal{V}_i'}^2\right)^{1/2} \left(\sum_{i=1}^J \|w_i\|_{\mathcal{V}}^2\right)^{1/2} \\ &\leq C_A^{1/2} \left(\sum_{i=1}^J \|g_i\|_{\mathcal{V}_i'}^2\right)^{1/2} \|w\|_{\mathcal{V}}.\end{aligned}$$

Thus,

$$\|\nabla f(x^k)\|_{\mathcal{V}'} = \sup_{w\in\mathcal{V}} \frac{\langle \nabla f(x^k), w\rangle}{\|w\|_{\mathcal{V}}} \leq C_A^{1/2} \left(\sum_{i=1}^J \|g_i\|_{\mathcal{V}_i'}^2\right)^{1/2},$$

and the proof is completed by taking squares. □

Lemma 6.4.3 implies that Assumption (AP) holds with $\gamma = C_A^{1/2}$.

Now we have the following convergence results of preconditioned RFASD (pre-RFASD) methods.

Table 6.1: Complexity of pre-RFASD

	Convex	Strongly convex
Uniform Sampling	$O\left(\frac{C_A J L \kappa_{\max}}{\epsilon}\right)$	$O\left(\frac{C_A J L \kappa_{\max}}{\mu} \log \frac{1}{\epsilon}\right)$
Importance Sampling	$O\left(\frac{C_A \ell \kappa_{\max}}{\epsilon}\right)$	$O\left(\frac{C_A \ell \kappa_{\max}}{\mu} \log \frac{1}{\epsilon}\right)$

For BCD, the convergence rate for strongly convex function with uniform sampling is $1 - \frac{1}{J}\frac{\mu}{L}$, while for RFASD the convergence rate is $1 - \frac{1}{J}\frac{1}{C_A}\frac{\mu}{L}\frac{1}{\kappa_{\max}}$. The convergence rate of BCD is simply depends on $\frac{L}{\mu}$– the condition number of function f in space $\mathcal{V}$ and J– the number of subspaces. The convergence rate of RFASD is influenced by $\frac{L}{\mu}$– the condition number of function f in space $\mathcal{V}$, J– the number of subspaces, C_A– the constant in stable decomposition and $\kappa_{\max}$– the maximum condition number of f on subspaces $\mathcal{V}_i$. The issue for BCD is the condition number $\frac{L}{\mu}$ can be very large for ill-conditioned problem. Thus the convergence is slow. However, if we apply RFASD, the condition number $\frac{L}{\mu}$ is reduced to $O(1)$ if measured in $\|\cdot\|_A$. The local condition number

$\kappa_{\max}$ can also be reduced to $O(1)$ if proper preconditioner A_i is found. And C_A is controllable by stable decomposition of space $\mathcal{V}$. Thus $\frac{1}{C_A}\frac{\mu}{L}\frac{1}{\kappa_{\max}}$ in RFASD is much smaller than $\frac{\mu}{L}$ in BCD, which accelerates convergence significantly.

6.4.4 Randomized Full Approximation Storage Scheme

For full approximation storage (FAS) scheme, consider the space decomposition $\mathcal{V} = \mathcal{V}_1 + \mathcal{V}_2 + \cdots + \mathcal{V}_J$ and $\mathcal{V}_i \subset \mathcal{V}$. We assume the space decomposition satisfies the following assumption

Let $I_i : \mathcal{V}_i \mapsto \mathcal{V}$ be the natrual inclusion and $R_i = I_i^{\intercal}$. Let $Q_i : \mathcal{V} \mapsto \mathcal{Q}_i$ be a projection operator and, ideally, $Q_i v$ should provide a good approximation of v in the subspace $\mathcal{V}_i$. In addition, $f_i(x) : \mathcal{V}_i \mapsto \mathbb{R}$ is the local objective functions. Here, we have freedom to choose f_i on subspace $\mathcal{V}_i$ and assume the following conditions on f_i hold.

- (LCfi) Lipschitz continuity of the first order dervative: for all $x, y \in \mathcal{V}_i$

$$\|\nabla f_i(x) - \nabla f_i(y)\|_{\mathcal{V}'} \le L_i \|x - y\|_{\mathcal{V}}.$$

- (SCfi) Strong convexity: for all $x, y \in \mathcal{V}_i$

$$\langle \nabla f_i(x) - \nabla f_i(y), x - y \rangle \ge \mu_i \|x - y\|_{\mathcal{V}}^2.$$

For FAS, the search direction s_i, $i = 1, 2, \cdots, N$, is chosen in the following way,

$$s_i := (\eta_i - Q_i x^k), \quad \langle \nabla f_i(\eta_i), v_i \rangle = \langle \nabla f_i(Q_i x^k) - R_i \nabla f(x^k), v_i \rangle, \ \forall v_i \in \mathcal{V}_i.$$

Assume $\nabla^2 f$ exists. Define $A_i = \int_0^1 \nabla^2 f_i(Q_i x^k + t s_i) dt$. Then

$$\mu_i (v_i, v_i)_{\mathcal{V}} \le \langle A_i s_i, s_i \rangle \le \lambda_i (v_i, v_i)_{\mathcal{V}}.$$

(SE) holds with λ_i and μ_i defined in (LCfi) and (SCfi).

Now let us verify Assumption (DD) and (AP). For (DD),

$$\begin{aligned}\langle -\nabla f(x^k), \mu_i s_i\rangle &= \mu_i \langle \nabla f_i(Q_i x^k + s_i) - \nabla f_i(Q_i x^k), s_i\rangle \\ &= \mu_i \langle A_i s_i, s_i\rangle \\ &= \mu_i \|s_i\|_{A_i}^2.\end{aligned}$$

Therefore Assumption (DD) holds with $\delta = 1$.

For (AP), by Lemma 6.4.3, we have $\gamma = C_A^{1/2}$.

We have the following convergence results of randomized FAS (RFAS) scheme in the RFASD framework.

Table 6.2: Complexity of RFAS

	Convex	Strongly convex
Uniform Sampling	$O\left(\dfrac{C_A J L \kappa_{\max}}{\epsilon}\right)$	$O\left(\dfrac{C_A J L \kappa_{\max}}{\mu}\log\dfrac{1}{\epsilon}\right)$
Importance Sampling	$O\left(\dfrac{C_A \ell \kappa_{\max}}{\epsilon}\right)$	$O\left(\dfrac{C_A \ell \kappa_{\max}}{\mu}\log\dfrac{1}{\epsilon}\right)$

Remark 6.4.4. *If we use $f_i(w) = \frac{1}{2}\|w - Q_i x^k\|_{A_i}^2$, RFAS is the same as preconditioned RFASD. This illustrates why they have the same convergence results.*

6.5 Numerical Results of Randomized Fast Subspace Descent Methods

In this section, we consider several methods which can be viewed as specific examples of Randomized Fast Subspace Descent Methods.

6.5.1 Randomized CD and Preconditioned FASD

Numerical experiments are done on Nestrov's worst function described in Example 6.4.1. The comparison are made between randomized CD (RCD), Preconditioned RFASD (RFASD) and Preconditioned RFASD with permutation (non-replacement sampling) (RFASD$_{\mathrm{perm}}$). To illustrate the influence of randomization, performance of cyclic CD and FASD are provided.

Preconditioner is constructed by Geometric Mulgirid Methods (GMG). Total number of iterations is listed in column niter1.mean, niter2.mean and niter3.mean for RCD, RFASD and RFASD$_{\mathrm{perm}}$ respectively. The number of iterations niter1dn of RCD is counted by total numbers of sweeps divided by $N-$ dimension of $\mathcal{V}$ while the number of iterations niter2dj of RFASD and niter3dj of RFASD$_{\mathrm{perm}}$ are counted by total number of sweeps divided by $J-$ number of subspaces while for each i, $\dim(\mathcal{V}_i) = 1$.

We compare the numerical results as following. Each experiment is tested on 8 times for RCD and RFASD and statistics of numerical results are listed in TABLE 6.3 and TABLE 6.5. Statistics of numerical results of RFASD and RFASD$_{\mathrm{perm}}$ are compared in TABLE 6.4 and TABLE 6.6. From TABLE 6.3, we find the niter1dn has a sharp increasing as N increases while niter2dj has a uniform bound as J increases. From TABLE 6.4, we find both niter2dj and niter3dj have uniform bounds as J increases while the bound for niter3dj is smaller. Permutation (non-replacement sampling) contributes to fast convergence.

As mentioned in TABLE 6.19, the complexity of RCD is $O\left(\frac{NL}{\mu}\log\frac{1}{\epsilon}\right)$. In ℓ_2 norm, the condition number of Nestrov's worst function in nondegenerate case is $O(N^2)$. The complexity is $O\left(N^3\log\frac{1}{\epsilon}\right)$. Iteration steps increase tremendously as N increases. For preconditioned RFASD, the complexity is $O\left(\frac{C_A JL\kappa_{\max}}{\mu}\log\frac{1}{\epsilon}\right)$. Viewed in A norm, the condition number of Nestrov's worst function is $\frac{L}{\mu} = O(1)$. In A_i norm, the condition number of sub problem is 1. Thus $\kappa_{\max} = 1$. By multigrid subspace decomposition, C_A is controllable. This explains the uniform convergence steps for RFASD in Nestrov's worst function.

The elapsed CPU time for RCD, RFASD and RFASD$_{\mathrm{perm}}$ are shown in TABLE 6.7 and 6.8, which implies RFASD and RFASD$_{\mathrm{perm}}$ increases much faster than RCD in setup time but save a lot in iteration time and thus total CPU time. This is because for RFASD and RFASD$_{\mathrm{perm}}$, prolongation and restriction matrices need to be computed. However, RFASD and RFASD$_{\mathrm{perm}}$ achieve uniform iteration steps. Thus RFASD and RFASD$_{\mathrm{perm}}$ benefits in the total elapsed CPU time.

CPU time for cyclic CD and preconditioned FASD are shown in TABLE 6.12 and 6.13.Similar phenomenon can be observed by the different trend of setup time and iteration time. Cyclic CD needs almost half iteration steps to reach accuracy compared to random CD since it goes through every coordinate of space $\mathcal{V}$ in one epoch. Cyclic FASD also shows a faster convergence than random FASD but not significant as the difference between cyclic CD and random CD. It needs approximately one third less iterations measured in niter2 and 4 5 less steps measures in niter2dj.

Table 6.3: RCD and RFASD iterations mean

RCD				RFASD			
N	niter1.mean	niter1dn.mean	diff1.mean	J	niter2.mean	niter2dj.mean	diff2.mean
7	1147.9	163.98	4.80e-13	11	135.38	12.31	6.55e-12
15	8864.1	590.94	1.93e-12	26	337.5	12.98	1.82e-10
31	66704	2151.7	7.96e-12	57	845.5	14.83	1.61e-09
63	4.88e+5	7741	3.13e-11	120	1814.1	15.12	1.15e-08
127	3.50e+6	27576	1.20e-10	247	3963.9	16.05	1.12e-07
255	2.47e+7	96672	4.65e-10	502	7897	15.73	8.30e-07
511	1.23e+8	2.40e+5	5.71e-08	1013	16044	15.84	7.92e-06

Table 6.4: RFASD and RFASD$_{\mathrm{perm}}$ iterations mean

RFASD				RFASD$_{\mathrm{perm}}$			
J	niter2.mean	niter2dj.mean	diff2.mean	J	niter3.mean	niter3dj.mean	diff3.mean
11	135.38	12.31	6.55e-12	11	39.25	3.57	3.19e-12
26	337.5	12.98	1.82e-10	26	169	6.5	1.33e-10
57	845.5	14.83	1.61e-09	57	393.25	6.90	1.43e-09
120	1814.1	15.12	1.15e-08	120	881.75	7.35	1.33e-08
247	3963.9	16.05	1.12e-07	247	1822	7.38	1.17e-07
502	7897	15.73	8.30e-07	502	3806.8	7.58	1.04e-06
1013	16044	15.84	7.92e-06	1013	7785.6	7.69	8.44e-06

Table 6.5: RCD and RFASD iterations std

RCD				RFASD			
N	niter1.std	niter1dn.std	diff1.std	J	niter2.std	niter2dj.std	diff2.std
7	36.20	5.17	3.54e-14	11	80.75	7.34	7.96e-12
15	142.98	9.53	1.41e-13	26	42.96	1.65	8.02e-11
31	265.99	8.58	2.26e-13	57	129.9	2.28	5.46e-10
63	1585.9	25.17	1.49e-12	120	157.27	1.31	5.41e-09
127	5313.9	41.842	2.5309e-12	247	409.81	1.6591	2.094e-08
255	23946	93.90	6.12e-12	502	708.22	1.41	1.55e-07
511	0	3.11e-11	6.84e-11	1013	1382.7	1.36	1.06e-06

Table 6.6: RFASD and RFASD$_{\rm perm}$ iterations std

RFASD				RFASD$_{\rm perm}$			
J	niter2.std	niter2dj.std	diff2.std	J	niter3.std	niter3dj.std	diff3.std
11	80.75	7.34	7.96e-12	11	28.22	2.57	6.53e-12
26	42.96	1.65	8.02e-11	26	22.46	0.86	8.17e-11
57	129.9	2.28	5.46e-10	57	18.52	0.32	3.33e-10
120	157.27	1.31	5.41e-09	120	81.13	0.68	4.43e-09
247	409.81	1.66	2.09e-08	247	63.99	0.26	3.20e-08
502	708.22	1.41	1.55e-07	502	255.16	0.51	2.34e-07
1013	1382.7	1.36	1.06e-06	1013	385.81	0.38	1.68e-06

Table 6.7: RCD, RFASD and RFASD$_{\rm perm}$ CPU time mean

RCD				RFASD		RFASD$_{\rm perm}$	
N	setup1.mean	iter1.mean	J	setup2.mean	iter2.mean	setup3.mean	iter3.mean
7	2.70e-04	7.84e-03	11	3.75e-04	7.51e-04	2.60e-04	3.75e-04
15	1.37e-04	4.28e-02	26	1.56e-03	8.31e-04	3.02e-04	4.48e-04
31	4.49e-04	3.07e-01	57	c2.52e-03	2.52e-03	3.76e-04	8.82e-04
63	1.24e-04	2.41	120	8.48e-04	9.80e-03	5.06e-04	3.85e-03
127	1.41e-04	18.34	247	1.39e-03	4.82e-02	1.24e-03	1.91e-02
255	1.96e-04	133.68	502	5.74e-03	2.42e-01	4.65e-03	1.21e-01
511	3.24e-04	811.64	1013	3.06e-02	3.71	2.59e-02	1.69

Table 6.8: RCD, RFASD and RFASD$_{\mathrm{perm}}$ CPU time std

RCD			RFASD			RFASD$_{\mathrm{perm}}$		
N	setup1.std	iter1.std	J	setup2.std	iter2.std	J	setup3.std	iter3.std
7	2.09e-04	3.94e-03	11	2.60e-04	1.08e-03	11	1.66e-04	8.85e-04
15	8.05e-05	6.18e-03	26	3.51e-03	3.04e-04	26	2.46e-04	6.24e-04
31	9.65e-04	4.38e-02	57	5.89e-03	4.06e-04	57	1.24e-04	2.68e-04
63	4.49e-05	4.73e-01	120	7.35e-04	7.60e-04	120	6.83e-05	4.23e-04
127	2.41e-05	4.22	247	9.66e-04	9.47e-03	247	4.17e-04	6.81e-04
255	3.80e-05	16.50	502	7.67e-04	2.45e-02	502	6.24e-04	3.34e-02
511	9.84e-05	82.10	1013	5.65e-03	5.43e-01	1013	2.55e-03	3.34e-01

6.5.2 Cyclic CD and Preconditioned FASD

The results of cyclic version CD and preconditioned FASD are also included in TABLE 6.9, 6.10 and 6.11.

Table 6.9: cyclic CD iterations mean

N	niter1.mean	niter1dn.mean	diff1.mean
7	559	79.86	5.57e-13
15	4380	292	3.03e-12
31	32735	1056	1.27e-11
63	2.38e+05	3779	5.16e-11
127	1.70e+06	13360	2.07e-10
255	1.19e+07	46472	8.30e-10
511	8.08e+07	1.58e+05	3.32e-09

Table 6.10: cyclic pre-FASD iterations mean

J	niter2.mean	niter2dj.mean	diff2.mean
11	79	7.18	1.20e-12
26	262	10.08	5.33e-11
57	629	11.04	8.20e-10
120	1392	11.6	1.19e-08
247	2912	11.79	1.27e-07
502	6050	12.05	1.05e-06
1013	12267	12.11	7.86e-06

Table 6.11: cyclic CD and pre-FASD iterations std

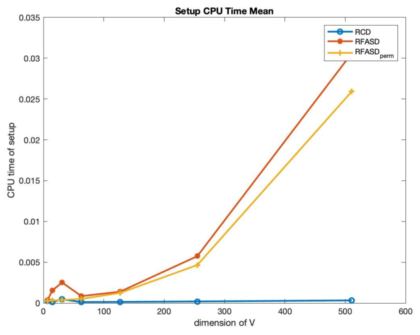
cyclic CD				cyclic pre-FASD			
N	niter1.std	niter1dn.std	diff1.std	J	niter2.std	niter2dj.std	diff2.std
7	0	1.52e-14	0	11	0	9.50e-16	0
15	0	0	0	26	0	1.90e-15	0
31	0	0	0	57	0	0	0
63	0	4.86e-13	0	120	0	1.90e-15	0
127	0	0	0	247	0	1.90e-15	0
255	0	0	0	502	0	1.90e-15	0
511	0	0	0	1013	0	0	0

Table 6.12: cyclic CD and pre-FASD CPU time mean

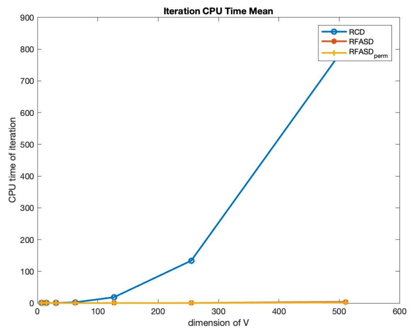
cyclic CD			cyclic pre-FASD		
N	setup1.mean	iter1.mean	J	setup2.mean	iter2.mean
7	6.41e-4	2.68e-3	11	7.36e-4	5.11e-4
15	1.61e-4	1.83e-2	26	3.11e-4	5.80e-4
31	1.89e-4	1.36e-1	57	4.30e-4	1.57e-3
63	1.95e-4	9.90e-1	120	5.79e-4	7.65e-3
127	1.49e-4	7.24	247	1.03e-3	3.46e-2
255	2.00e-4	57.74	502	3.57e-3	1.84e-1
511	3.34e-4	493.18	1013	2.28e-2	2.67

Table 6.13: cyclic CD and pre-FASD CPU time std

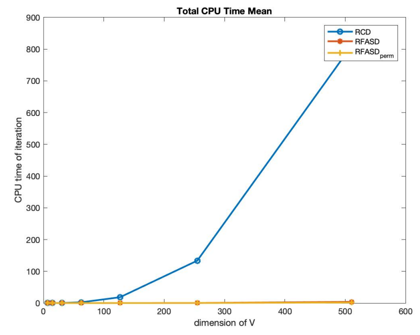
cyclic CD			cyclic pre-FASD		
N	setup1.std	iter1.std	J	setup2.std	iter2.std
7	1.31e-3	8.88e-4	11	1.51e-3	1.01e-3
15	4.96e-5	2.06e-3	26	1.12e-4	2.57e-4
31	8.39e-5	1.69e-2	57	8.33e-5	7.67e-5
63	9.48e-5	1.69e-2	120	9.98e-5	1.07e-3
127	4.55e-5	1.74e-1	247	1.30e-4	3.52e-3
255	3.38e-5	1.30	502	8.52e-4	5.26e-3
511	2.27e-4	7.64	1013	4.22e-3	3.19e-1



(a)

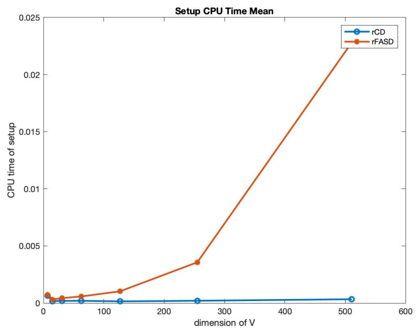


(b)

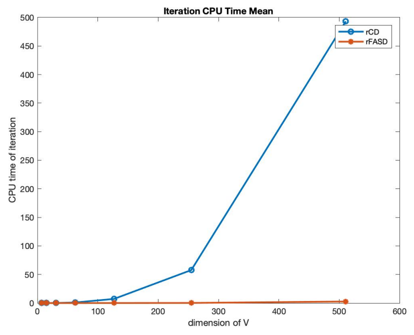


(c)

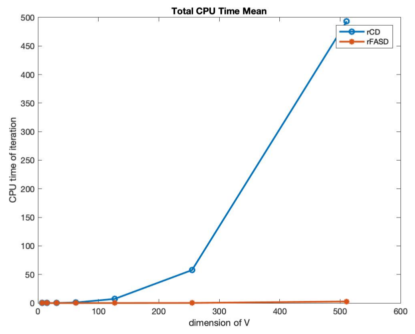
Figure 6.1: Random CD and FASD (a) Setup Time (b) Iteration Time (c) Total Time



(a)



(b)



(c)

Figure 6.2: Cyclic CD and pre-FASD (a) Setup Time (b) Iteration Time (c) Total Time

6.5.3 Linear Regression

We also test RCD and randomized RFASD in linear regression problems and list results in Table 6.14 and 6.15. In this example, RCD fails to converge to minimal.

The objective function of minimization problem in [20] is $f(x) = \frac{1}{m}\|b - Ax\|^2 + \lambda\|x\|_2^2$.

- Data matrix $A_{m\times n}$: each row is independently sampled from a N-dim Gaussian distribution with mean 0 and variance Σ;
- x: randomly select N/10 entries of x, each of which is independently sampled from a uniform distribution over support $(-2, 2)$;
- Label b: generated by linear model $b = Ax + \epsilon,\ \epsilon \sim \mathcal{N}(0, I_n)$;
- Covariance matrix Σ: $\Sigma_{ii} = 1, \Sigma_{ij} \in \{0, 0.5, 0.75, 0.95\}$. The larger of off-diagonal values, the more ill condition of the data matrix A.
- Regularization parameter λ: parameter makes problem less ill-conditioned, $\lambda = 0, 0.1/m$.

Table 6.14: RCD and RFASD iterations: Gaussian Matrices

Problems			randomized CD			randomized pre-FASD		
M	N	f_{opt}	niter1	niter1dn	f_{opt1}	niter2	niter2dj	f_{opt2}
14	7	4.38e-29	1.03e+05	1.48e+04	1.00e-1	62	5.17	8.92e-02
30	15	1.02e-27	6.4e+05	4.27e+04	2.68	618	2.94e+01	9.90e-02
62	31	1.31e-20	1.28e+06	4.12e+04	1.05e+02	2.12e+04	2.23e+02	9.96e-02
126	63	5.36e-18	2.56e+06	4.06e+04	2.05e+04	2.12e+04	2.23e+02	1.00e-01
254	127	2.68e-18	5.12e+06	4.03e+04	5.96e+04	1.94e+04	1.17e+02	1.00e-01
510	255	3.49e-15	1.02e+07	4.01+04	2.45e+04	1.62e+05	5.99e+02	1.00e-01

The following tables 6.16 and 6.17 provide applying RCD and Rpre-FASD to linear regression with A is a UDV matrix defined in Chapter 3.

Table 6.15: RCD and RFASD CPU time: Gaussian Matrices

Problems		randomized CD		randomized pre-FASD	
M	N	setup1	iter1	step2	iter2
14	7	1.29e-03	1.26	1.68e-02	5.70e-04
30	15	1.92e-03	7.34	1.089e-03	4.77e-03
62	31	1.94e-03	1.57e+01	1.33e-03	2.42e-02
126	63	2.59e-03	4.02e+01	1.86e-03	3.06e-01
254	127	4.72e-03	1.22e+02	3.58e-03	4.97e-01

Table 6.16: RCD and RFASD iterations:UDV Matrices

Problems			randomized CD			randomized pre-FASD		
M	N	f_{opt}	niter1	niter1dn	f_{opt1}	niter2	niter2dj	f_{opt2}
14	7	1.72e-01	1.51e+05	2.16e+04	2.73e-1	2.93e+02	2.66e+01	2.72e-01
30	15	3.84e-01	6.4e+05	4.27e+04	1.98	3.21e+02	1.33e+01	4.79e-01
62	31	4.10e-01	1.28e+06	4.13e+04	1.14e+02	1.70e+04	3.39e+02	7.52e-01
126	63	6.52e-01	2.56e+06	4.06e+04	1.77e+03	1.30e+04	1.32e+02	7.52e-01
254	127	7.493-01	5.12e+06	4.03e+04	6.25e+03	4.08e+04	2.47e+02	8.48e-01
510	255	1.17	1.02e+07	4.01+04	2.07e+04	1.59e+05	5.98e+02	1.271

Table 6.17: RCD and RFASD CPU time:UDV Matrices

Problems		randomized CD		randomized pre-FASD	
M	N	setup1	iter1	step2	iter2
14	7	6.17e-03	1.50	8.26e-04	2.38e-03
30	15	6.89e-03	6.74	1.08e-02	2.84e-02
62	31	1.78e-03	1.46e+01	1.37e-03	1.58e-01
126	63	7.34e-02	1.20e+02	3.59e-03	1.07
254	127	7.26e-03	4.15e+02	9.70e-03	7.28

6.6 Accelerated FASD

In this section, we discuss how to accelerate RFASD. We consider two different cases. First case is that $(\sum_{i=1}^{J} \mu_i s_i, v)_{\mathcal{V}} = \langle -\nabla f(x^k), v\rangle$, for any $v \in \mathcal{V}$. In this case, Nesterov-type acceleration [32] can be applied. Examples include coordinate descent and block coordinate descent method. Second case is the general case that $(\sum_{i=1}^{J} \mu_i s_i, v)_{\mathcal{V}} \neq \langle -\nabla f(x^k), v\rangle$, for any $v \in \mathcal{V}$. For this case, directly apply Nesterov acceleration is difficult and we use the catalyst acceleration developed in [22] to accelerate RFASD. The full approximation storage (FAS) scheme from multigrid community belongs to this case.

Denote $\ell = \sum_{i=1}^{J} L_i$ and $\ell_S = \sum_{i=1}^{J} \sqrt{L_i}$.

6.6.1 Case I: $(\sum_{i=1}^{J} \mu_i s_i, v)_{\mathcal{V}} = \langle -\nabla f(x^k), v\rangle$

For this special case, we adopt the acceleration techniques developed in [33] to accelerate the RFASD method. The accelerated algorithm, which we call AFASD-I, is proposed as follows.

Here, we basically used Nesterov accelaration technique with a different nonuniform sampling strategy. Following the idea proposed in [33], we use a modified version of Assumption (DD) as follows.

- (DDm) Descent direction (modified version): let s_i be computed using y^k,

$$\langle -\nabla f(y^k), \sum_{i=1}^{J} \frac{\mu_i}{\sqrt{L_i}} s_i\rangle \geq \delta \left(\sum_{i=1}^{J} \frac{\mu_i}{\sqrt{L_i}} \|s_i\|_{A_i}^2 \right) \geq 0 \qquad (6.25)$$

Essentially, the right hand side is basically a weighted norm used in [33]. In fact, a general version $(\sum_{i=1}^{J} L_i^{\theta} \|s_i\|_{\mathcal{V}}^2)$ with $\theta \in [-1, 1]$ was considered there.

Next theorem shows that AFASD-I (Algorithm 11) achieves acceleration, which shows the accel-

```
1: Choose $v^0 = x^0$, set $A_0 = 0$, $B_0 = 1$
2: for $k = 0, 1, \cdots$ do
3:     Choose a index of subspace $i_k$ from $\{1, \cdots, J\}$ with probability
           $$p_{i_k} = \frac{\sqrt{L_{i_k}}}{\ell_S}, \quad \text{where } \ell_S = \sum_{i=1}^{J} \sqrt{L_i}.$$
4:     Solve $a_{k+1} > 0$ from the following equation
           $$a_{k+1}^2 \frac{\ell_S^2}{\delta^2} = (A_k + a_{k+1})(B_k + \mu a_{k+1}). \tag{6.22}$$
5:     Define
           $$A_{k+1} = A_k + a_{k+1}, \qquad \alpha_k = \frac{a_{k+1}}{A_{k+1}},$$
           $$B_{k+1} = B_k + \mu a_{k+1}, \qquad \beta_k = \frac{\mu a_{k+1}}{B_{k+1}}.$$
6:     Update
           $$y^k = \frac{(1-\alpha_k)x^k + \alpha_k(1-\beta_k)v^k}{1-\alpha_k\beta_k}. \tag{6.23}$$
7:     Compute a subspace search direction $s_{i_k} \in \mathcal{V}_{i_k}$ and update
           $$x^{k+1} = y^k + \frac{\delta \mu_{i_k}}{L_{i_k}} s_{i_k} \tag{6.24}$$
           $$v^{k+1} = (1-\beta_k)v^k + \beta_k y^k + \frac{a_{k+1}}{B_{k+1} p_{i_k}} \mu_{i_k} s_{i_k}.$$
8: end for
```

Algorithm 11: Accelerated RFASD: case I (AFASD-I)

eration still works for our general space decomposition, especially those with redundancy.

Theorem 6.6.1. *Assume the Assumptions (SC), (LCi), (SCi) and (DDm) hold, then the sequences x^k and v^k, $k = 0, 1, \cdots$ generated by AFASD-I (Algorithm 11) satisfy the following convergence estimates,*

$$B_k\mathbb{E}(\|v^k - x^*\|_{\mathcal{V}}^2) + 2A_k\left(\mathbb{E}(f(x^k)) - f(x^*)\right) \le \|x^0 - x^*\|_{\mathcal{V}}^2. \tag{6.26}$$

Proof. Denote $w^k = (1-\beta_k)v^k + \beta_k y^k$, we have,

$$\begin{aligned}\|v^{k+1} - x^*\|_{\mathcal{V}}^2 &= \|w^k + \frac{a_{k+1}}{B_{k+1}p_{i_k}}\mu_{i_k}s_{i_k} - x^*\|_{\mathcal{V}}^2 \\ &= \|w^k - x^*\|_{\mathcal{V}}^2 + \frac{2a_{k+1}}{B_{k+1}p_{i_k}}(\mu_{i_k}s_{i_k}, w^k - x^*)_{\mathcal{V}} + \frac{a_{k+1}^2}{B_{k+1}^2p_{i_k}^2}\mu_{i_k}^2\|s_{i_k}\|_{\mathcal{V}}^2 \\ &\le \|w^k - x^*\|_{\mathcal{V}}^2 + \frac{2a_{k+1}}{B_{k+1}p_{i_k}}(\mu_{i_k}s_{i_k}, w^k - x^*)_{\mathcal{V}} + \frac{a_{k+1}^2}{B_{k+1}^2p_{i_k}^2}\mu_{i_k}\|s_{i_k}\|_{A_{i_k}}^2.\end{aligned}$$

Note that, $\|w^k - x^*\|_{\mathcal{V}}^2 \le (1-\beta_k)\|v^k - x^*\|_{\mathcal{V}}^2 + \beta_k\|y^k - x^*\|_{\mathcal{V}}^2$, we have

$$\begin{aligned}B_{k+1}\|v^{k+1} - x^*\|_{\mathcal{V}}^2 \le (1-\beta_k)B_{k+1}\|v^k - x^*\|_{\mathcal{V}}^2 + \beta_k B_{k+1}\|y^k - x^*\|_{\mathcal{V}}^2 \\ + \frac{2a_{k+1}}{p_{i_k}}(\mu_{i_k}s_{i_k}, w^k - x^*)_{\mathcal{V}} + \frac{a_{k+1}^2\mu_{i_k}}{B_{k+1}p_{i_k}^2}\|s_{i_k}\|_{A_{i_k}}^2\end{aligned}$$

Taking conditional expectation on both sides of the above inequality and use the fact that $(1 -$

$\beta_k)B_{k+1} = B_k$ and $\beta_k B_{k+1} = \mu a_{k+1}$, we get

$$\begin{aligned}
B_{k+1}\mathbb{E}(\|v^{k+1} - x^*\|_{\mathcal{V}}^2) &\le B_k\|v^k - x^*\|_{\mathcal{V}}^2 + \mu a_{k+1}\|y^k - x^*\|_{\mathcal{V}}^2 \\
&\quad + \sum_{i=1}^J p_i \frac{2a_{k+1}}{p_i}(\mu_i s_i, w^k - x^*)_{\mathcal{V}} + \sum_{i=1}^J p_i \frac{a_{k+1}^2 \mu_i}{B_{k+1} p_i^2}\|s_i\|_{A_i}^2 \\
&= B_k\|v^k - x^*\|_{\mathcal{V}}^2 + \mu a_{k+1}\|y^k - x^*\|_{\mathcal{V}}^2 \\
&\quad + 2a_{k+1}\sum_{i=1}^J (\mu_i s_i, w^k - x^*)_{\mathcal{V}} + \frac{a_{k+1}^2}{B_{k+1}}\sum_{i=1}^J \frac{\mu_i}{p_i}\|s_i\|_{A_i}^2 \\
&= B_k\|v^k - x^*\|_{\mathcal{V}}^2 + \mu a_{k+1}\|y^k - x^*\|_{\mathcal{V}}^2 \\
&\quad + 2a_{k+1}\sum_{i=1}^J (\mu_i s_i, w^k - x^*)_{\mathcal{V}} + \frac{a_{k+1}^2 \ell_S}{B_{k+1}}\left(\sum_{i=1}^J L_i^{-\frac{1}{2}}\mu_i\|s_i\|_{A_i}^2\right).
\end{aligned} \tag{6.27}$$

Now we need to estimate the last two terms in the right hand side. For the last term, from (6.24), we obtain

$$f(x^{k+1}) \le f(y^k) - \frac{\delta\mu_{i_k}}{L_{i_k}}\langle -\nabla f(y^k), s_{i_k}\rangle + \frac{\delta^2\mu_{i_k}^2}{2L_{i_k}}\|s_{i_k}\|_{\mathcal{V}}^2.$$

Take the expectation as before, we have

$$\begin{aligned}
\mathbb{E}\left(f(x^{k+1})\right) &\le f(y^k) - \sum_{i=1}^J \frac{\sqrt{L_i}}{\ell_S}\frac{\delta\mu_i}{L_i}\langle -\nabla f(y^k), s_i\rangle + \sum_{i=1}^J \frac{\sqrt{L_i}}{\ell_S}\frac{\delta^2\mu_i^2}{2L_i}\|s_i\|_{\mathcal{V}}^2 \\
\text{Assumption (DDm)} &\le f(y^k) - \frac{\delta^2}{\ell_S}\left(\sum_{i=1}^J \frac{\mu_i}{\sqrt{L_i}}\|s_i\|_{\mathcal{V}}^2\right) + \frac{\delta^2}{2\ell_S}\left(\sum_{i=1}^J \frac{\mu_i}{\sqrt{L_i}}\|s_i\|_{\mathcal{V}}^2\right) \\
&\le f(y^k) - \frac{\delta^2}{2\ell_S}\left(\sum_{i=1}^J \frac{\mu_i}{\sqrt{L_i}}\|s_i\|_{\mathcal{V}}^2\right)
\end{aligned}$$

Therefore, we have

$$\left(\sum_{i=1}^J \frac{\mu_i}{\sqrt{L_i}}\|s_i\|_{\mathcal{V}}^2\right) \le \frac{2\ell_S}{\delta^2}\left(f(y^k) - \mathbb{E}(f(x^{k+1}))\right). \tag{6.28}$$

For the third term, from (6.23) and the definition of w^k, we have

$$\begin{aligned}(1-\alpha_k\beta_k)y^k &= (1-\alpha_k)x^k + \alpha_k(1-\beta_k)v^k \\ &= (1-\alpha_k)x^k + \alpha_k(w^k - \beta_k y^k) \\ &= (1-\alpha_k)x^k + \alpha_k w^k - \alpha_k\beta_k y^k,\end{aligned}$$

which implies $w^k = y^k - \frac{1-\alpha_k}{\alpha_k}(x^k - y^k)$. Note that

$$\begin{aligned}a_{k+1}\sum_{i=1}^{J}(s_i, w^k - x^*)_{\mathcal{V}} &= a_{k+1}\langle \nabla f(y^k), x^* - w^k\rangle \\ &= a_{k+1}\langle \nabla f(y^k), x^* - y^k + \frac{1-\alpha_k}{\alpha_k}(x^k - y^k)\rangle \\ \text{(Assumption (SC))} &\le a_{k+1}\left(f(x^*) - f(y^k)\right) - \frac{1}{2}\mu a_{k+1}\|y^k - x^*\|_{\mathcal{V}}^2 \\ &\quad + a_{k+1}\frac{1-\alpha_k}{\alpha_k}\left(f(x^k) - f(y^k)\right) \\ &= a_{k+1}f(x^*) - A_{k+1}f(y^k) + A_k f(x^k) - \frac{1}{2}\mu a_{k+1}\|y^k - x^*\|_{\mathcal{V}}^2.\end{aligned} \tag{6.29}$$

Plug (6.29) and (6.28) back into (6.27), we have

$$\begin{aligned}B_{k+1}\mathbb{E}(\|v^{k+1} - x^*\|_{\mathcal{V}}^2) &\le B_k\|v^k - x^*\|_{\mathcal{V}}^2 + \mu a_{k+1}\|y^k - x^*\|_{\mathcal{V}}^2 \\ &\quad + 2a_{k+1}f(x^*) - 2A_{k+1}f(y^k) + 2A_k f(x^k) - \mu a_{k+1}\|y^k - x^*\|_{\mathcal{V}}^2 \\ &\quad + \frac{2a_{k+1}^2\ell_S^2}{B_{k+1}\delta^2}\left(f(y^k) - \mathbb{E}(f(x^{k+1}))\right) \\ &= B_k\|v^k - x^*\|_{\mathcal{V}}^2 + 2(A_{k+1} - A_k)f(x^*) - 2A_{k+1}f(y^k) + 2A_k f(x^k) \\ &\quad + 2A_{k+1}\left(f(y^k) - \mathbb{E}(f(x^{k+1}))\right) \\ &= B_k\|v^k - x^*\|_{\mathcal{V}}^2 + 2A_k\left(f(x^k) - f(x^*)\right) - 2A_{k+1}\left(\mathbb{E}(f(x^{k+1})) - f(x^*)\right).\end{aligned}$$

Take the expectation and apply the above inequality recursively, we obtain

$$B_{k+1}\mathbb{E}(\|v^{k+1}-x^*\|_{\mathcal{V}}^2)+2A_{k+1}\left(\mathbb{E}(f(x^{k+1}))-f(x^*)\right)\le\|x^0-x^*\|_{\mathcal{V}}^2,$$

where we use the fact that $A_0=0$, $B_0=1$, and $v^0=x^0$. $\square$

Now we need estimate the growth of the coefficients A_k and B_k, which is given in the following lemma.

Lemma 6.6.2. *The sequences A_k and B_k, $k=0,1,\cdots$, generated by the AFASD-I (Algorithm 11) satisfies*

$$A_{k+1}\ge\frac{\delta^2}{4\ell_S^2}k^2,\qquad B_k\ge 1. \tag{6.30}$$

Moreover, for the strongly convex case, i.e., $\mu>0$, the sequences can be further estimated as follows:

$$A_k\ge\frac{1}{4\mu}\left[(1+\zeta)^k-(1-\zeta)^k\right]^2\ge\frac{\delta^2}{4\ell_S^2}k^2 \tag{6.31}$$

$$B_k\ge\frac{1}{4}\left[(1+\zeta)^k+(1-\zeta)^k\right]^2, \tag{6.32}$$

where $\zeta=\dfrac{\sqrt{\mu}\delta}{2\ell_S}$.

Proof. The proof is essentially the same as the proof given in [33]. We briefly recall the proof here for the completeness.

Note that (6.22) can be rewritten as

$$(A_{k+1}-A_k)^2\frac{\ell_S^2}{\delta^2}=A_{k+1}(1+\mu A_{k+1}),$$

where we use the fact that $B_{k+1} = 1 + \mu A_{k+1}$. Denote $C_k = \sqrt{\mu A_k}$, $k = 0, 1, \cdots$, then

$$\mu^{-1} C_{k+1}^2 (1 + C_{k+1}^2) = \mu^{-2} \frac{\ell_S^2}{\delta^2} (C_{k+1}^2 - C_k^2)^2 \leq 4\mu^{-2} \frac{\ell_S^2}{\delta^2} (C_{k+1} - C_k)^2 C_{k+1}^2.$$

Then use the definition of ζ, we have

$$C_{k+1} - C_k \geq \zeta (1 + C_{k+1}^2)^{\frac{1}{2}} \geq \zeta (1 + C_k^2)^{\frac{1}{2}}.$$

Then by mathematical induction, we have

$$C_k \geq \frac{1}{2} \left[(1+\zeta)^k - (1-\zeta)^k \right] \geq \zeta k,$$

which implies (6.31) if $\mu \geq 0$. If $\mu = 0$, we have $(A_{k+1} - A_k)^2 \frac{\ell_S^2}{\delta^2} = A_{k+1}$ and (6.30) can be derived directly. Finally, we just the fact $B_k = 1 + \mu A_k$ to estimate B_k so that (6.32) and (6.30) follow immediately. $\square$

Now we present the finally convergence result in the following theorem by combing Theorem 6.6.1 and Lemma 6.6.2.

Theorem 6.6.3. *Assume the Assumptions (A1), (A2), (SA), and (DDm) hold. Let sequences x^k and v^k, $k = 0, 1, \cdots$ be generated by AFASD-I (Algorithm 11).*

- *If $\mu = 0$, i.e, f is convex, and Assumption (BL) holds, then we have*

$$\mathbb{E}(f(x^k)) - f(x^*) \leq \frac{1}{2A_k} \|x^0 - x^*\|_{\mathcal{V}}^2 \leq \frac{2\ell_S^2 R_0^2}{\delta^2 k^2} \tag{6.33}$$

- *If $\mu > 0$, i.e., f is strongly convex, then we have*

$$\mathbb{E}(f(v^k)) - f(x^*) \leq \frac{L}{2} \mathbb{E}(\|v^k - x^*\|_{\mathcal{V}}^2) \leq \frac{1}{B_k} \frac{L}{2} \|x^0 - x^*\|_{\mathcal{V}}^2 \leq 2 \left(1 + \frac{\sqrt{\mu}\delta}{2\ell_S} \right)^{-2k} L R_0. \tag{6.34}$$

Proof. (6.33) and (6.34) follow from Thereom 6.6.1 and Lemma (6.6.2). □

Remark 6.6.4. *To see AFASD-I actually accelerates RFASD, we need look at the complexity bound of obtaining an approximate solution within ϵ accuracy. It is reasonable to assume s_i gives reasonable approximation of $-\nabla f$ and, therefore, the constants δ in Assumption (DD) or (DDm) and γ in Assumption (AP) are just some constant indepedent of μ, ℓ_i, and L.*

For strongly convex case ($\mu > 0$), from (6.10) *and* (6.34)*, the complexity bounds are*

$$\begin{aligned} \textit{Algorithm 10 (RFASD):} \quad & \mathcal{O}\left(\frac{\ell}{\mu}\log\frac{1}{\epsilon}\right) \\ \textit{Algorithm 11 (AFASD-I):} \quad & \mathcal{O}\left(\frac{\ell_S}{\sqrt{\mu}}\log\frac{1}{\epsilon}\right) \end{aligned}$$

Note that, if $\frac{\ell}{\mu} \geq N$, we have

$$\frac{\ell_S}{\sqrt{\mu}} = \sum_{i=1}^{J}\sqrt{\frac{L_i}{\mu}} \leq \left(\sum_{i=1}^{J}\frac{L_i}{\mu}\right)^{\frac{1}{2}} J^{\frac{1}{2}} = \sqrt{J}\left(\frac{\ell}{\mu}\right)^{\frac{1}{2}} \leq \frac{\ell}{\mu}.$$

This means we achieve acceleration when $\frac{\ell}{\mu} \geq J$, i.e., the problem is ill-conditioned.

For convex case ($\mu = 0$), from (6.15) *and* (6.33)*, the complexity bounds are*

$$\begin{aligned} \textit{Algorithm 10 (RFASD):} \quad & \mathcal{O}\left(\frac{1}{\epsilon}\ell\right) = \mathcal{O}\left(\frac{1}{\epsilon}\sum_{i=1}^{J} L_i\right) \\ \textit{Algorithm 11 (AFASD-I):} \quad & \mathcal{O}\left(\frac{1}{\sqrt{\epsilon}}\ell_S\right) = \mathcal{O}\left(\frac{1}{\sqrt{\epsilon}}\sum_{i=1}^{J}\sqrt{L_i}\right) \end{aligned}$$

The acceleration is quite clear.

6.6.2 Case II: $(\sum_{i=1}^{J}\mu_i s_i, v)_{\mathcal{V}} \neq \langle -\nabla f(x^k), v\rangle$

In this case, we use the catalyst acceleration developed in [22]. It is a inner-outer iterative method which can be interpreted as an inexact accelerated proximal point algorithm. More precisely, in the inner-loop, it uses a linear convergent algorithm to approximately solve a regularized optimization problem near a given point y^{k-1} defined as follows,

$$f_k(x) := f(x) + \frac{\sigma}{2}\|x - y^{k-1}\|_{\mathcal{V}}^2. \tag{6.35}$$

Then Nesterov acceleration is used as the outer-loop in order to achieve the overall acceleration.

In order to keep the presentation simple and clear, we use a special version of catalyst acceleration in this paper to demonstrate how to accelerate RFASD in general and the algorithm (which is referred as AFASD-II) is presented in Algorithm 12

```
1: Choose $x^0$, $\sigma$, and $\{\varepsilon_k\}_{k\geq 0}$, set $y^{-1} = y^0 = x^0$ and $q = \frac{\mu}{\mu+\sigma}$. If $\mu > 0$, set $\alpha_0 = \sqrt{q}$,
   otherwise $\alpha_0 = 1$.
2: for $k = 1, \cdots$ do
3:     Compute an approximation solution $x^k$ of $f_k(x)$ with RFASD (Algorithm 10),
       $$x^k \approx \operatorname{argmin}_{x\in\mathcal{V}} \left\{ f_k(x) := f(x) + \frac{\sigma}{2}\|x - y^{k-1}\|^2_{\mathcal{V}} \right\},$$
       with initial guess $x^{k-1}$ and stopping criteria
       $$f_k(x^k) - f_k^* \leq \frac{\varepsilon_k \sigma}{2}\|y^{k-1} - x^k\|^2_{\mathcal{V}}. \qquad (6.36)$$
4:     Update $\alpha_k$ by solving
       $$\alpha_k^2 = (1-\alpha_k)\alpha_{k-1}^2 + q\alpha_k.$$
5:     Compute $y^k$
       $$y^k = x^k + \beta_k(x^k - x^{k-1}) \text{ with } \beta_k = \frac{\alpha_{k-1}(1-\alpha_{k-1})}{\alpha_{k-1}^2 + \alpha_k}$$
6: end for
```

Algorithm 12: Accelerated RFASD: case II (AFASD-II)

Next we present the convergence theory of AFASD-II (Algorithm 12). Since the proofs are exactly same as those in [22], we omit them and only state the theories. First, let us look at the convergence of the outer-loop in the following theorem.

Theorem 6.6.5. *Consider the sequences $\{x^k\}$ generated by AFASD-II (Algorithm 12).*

- *For strongly convex case, i.e. Assumption (SC) holds with $\mu > 0$, choose $\varepsilon_k = \frac{\sqrt{q}}{2-\sqrt{q}}$ with $q = \frac{\mu}{\mu+\sigma}$, then we have*

$$f(x^k) - f^* \leq 2\left(1 - \frac{\sqrt{q}}{2}\right)^k \left(f(x^0) - f^*\right).$$

- *For convex case, i.e. Assumption (SC) holds with $\mu = 0$, choose $\varepsilon_k = (\frac{1}{k+1})^2$ and further assume Assumption (BL) holds, then we have,*

$$f(x^k) - f^* \leq \frac{4\sigma R_0^2}{(k+1)^2}.$$

Now we investigate the inner-loop. Since we use RFASD (Algorithm 10) to solve minimization 6.35, apply Theorem 6.3.2, its convergence is

$$\mathbb{E}\left(f_k(x^k)\right) - f_k^* \leq (1-\tau)^t \left(f_k(x^{k-1}) - f_k^*\right),$$

where $\tau = \dfrac{\mu+\delta}{J(L+\delta)}\dfrac{\delta^2}{\gamma^2}\dfrac{1}{k_{\max}}$ with uniform sampling and $\tau = \dfrac{\mu+\delta}{\ell+J\delta}\dfrac{\delta^2}{\gamma^2}\dfrac{1}{k_{\max}}$ with non-uniform sampling proportional to $L_i + \sigma$. Note here, as shown in Algorithm 12, we use x^{k-1} as the initial guess and x^k is the approximate solution satisfies the stopping criteria (6.36) after t inner iterations. Then we have the following result about the inner-loop complexity T_k at outer iteration k, i.e.,

$$T_k := \inf\{t \geq 0, x^k \text{ satisifes (6.36)}\}. \tag{6.37}$$

Theorem 6.6.6. *Assume Assumption (LC) and (LCi) holds,*

- *For strongly convex case, i.e., $\mu > 0$, then we have*

$$\mathbb{E}(T_k) \leq \frac{1}{\tau}\log\left(\frac{8}{\tau}\frac{(L+\sigma)}{\sigma}\frac{(2-\sqrt{q})}{\sqrt{q}}\right) + 1.$$

- *For convex case, i.e., $\mu = 0$, then we have*

$$\mathbb{E}(T_k) \leq \frac{1}{\tau} \log\left(\frac{8}{\tau}\frac{L+\sigma}{\sigma}(k+1)^2\right) + 1.$$

Now we are ready to look at the complexity of the overall algorithm, which can be simply obtained by multiply the inner and outer iterations.

Theorem 6.6.7. *Under the same assumptions of Theorem 6.6.5 and 6.6.6, AFASD-II (Algorithm 12) finds a solution within ϵ accuracy in at most k_{total} iterations.*

- *For strongly convex case, i.e., $\mu > 0$, then we have*

$$\mathbb{E}(k_{total}) \leq \frac{2}{\tau\sqrt{q}}\left[\log\left(\frac{8}{\tau}\frac{(L+\sigma)}{\sigma}\frac{(2-\sqrt{q})}{\sqrt{q}}\right) + 1\right] \log\left(\frac{2(f(x^0) - f^*)}{\epsilon}\right) = \tilde{\mathcal{O}}\left(\frac{1}{\tau\sqrt{q}} \log\frac{1}{\epsilon}\right)$$

- *For convex case, i.e., $\mu = 0$, then we have*

$$\mathbb{E}(k_{total}) \leq \frac{1}{\tau}\sqrt{\frac{4\sigma R_0^2}{\epsilon}}\left[\log\left(\frac{32(L+\sigma)R_0^2}{\tau\epsilon}\right) + 1\right] = \tilde{\mathcal{O}}\left(\frac{1}{\tau}\sqrt{\frac{\sigma}{\epsilon}} \log\frac{1}{\epsilon}\right).$$

Here the notation $\tilde{\mathcal{O}}$ hides some logarithmic dependencies on the L, μ, δ, γ, σ, and R_0.

Now we need to choose the best σ. As suggested in [22], we should choose σ to maximize $\frac{\tau}{\sqrt{\mu+\sigma}}$, i.e.,

- With uniform sampling,

$$\max_{\sigma} \frac{\tau}{\sqrt{\mu+\sigma}} = \max_{\sigma} \frac{\sqrt{\mu+\sigma}}{J(L+\sigma)} \frac{\delta^2}{\gamma^2} \frac{1}{\kappa_{\max}}.$$

- With importance sampling,

$$\max_{\sigma} \frac{\tau}{\sqrt{\mu+\sigma}} = \max_{\sigma} \frac{\sqrt{\mu+\sigma}}{\ell + J\sigma} \frac{\delta^2}{\gamma^2} \frac{1}{\kappa_{\max}}.$$

The maximizer is

- With uniform sampling,

$$\sigma = L - 2\mu, \quad \text{when } L > 2\mu.$$

- With importance sampling,

$$\sigma = \frac{\ell}{J} - 2\mu, \quad \text{when } \ell > 2J\mu.$$

Therefore, the have the following complexity bounds of Algorithm 12 (AFASD-II).

- Uniform sampling & Convex: $\tilde{\mathcal{O}}\left(\frac{\gamma^2}{\delta^2} J \sqrt{\frac{L}{\epsilon}} \kappa_{\max} \log \frac{1}{\epsilon}\right)$
- Importance sampling & Convex: $\tilde{\mathcal{O}}\left(\frac{\gamma^2}{\delta^2} \sqrt{J} \sqrt{\frac{\ell}{\epsilon}} \kappa_{\max} \log \frac{1}{\epsilon}\right)$
- Uniform sampling & Strongly convex: $\tilde{\mathcal{O}}\left(\frac{\gamma^2}{\delta^2} J \sqrt{\frac{L}{\mu}} \kappa_{\max} \log \frac{1}{\epsilon}\right)$
- Importance sampling & Strongly convex: $\tilde{\mathcal{O}}\left(\frac{\gamma^2}{\delta^2} \sqrt{J} \sqrt{\frac{\ell}{\mu}} \kappa_{\max} \log \frac{1}{\epsilon}\right)$

As we can see, those bounds are near-optimal up to logarithmic constants according to the first-order lower bound, see [32].

To conclude, in this chapter, we present a new generic scheme randomized fast subspace descent methods for non-constraint problems (1.2). The way to construct the scheme is dividing space $\mathcal{V}$ into several subspaces $\mathcal{V}_i$ which satisfies stable decomposition assumption. Randomly choose a subspace $\mathcal{V}_{i_k}$ with either uniform or non-uniform sampling. Then compute search direction s_{i_k} such descent direction assumption (6.6) and approximate conditions (6.7). Apply gradient descent with optimal step size on subspace $\mathcal{V}_{i_k}$.

The difference between Uniform RFASD and Non-uniform RFASD is the constant factor JL and $\sum_{i=1}^{J} L_i$. Since $L = \max_i L_i$, $JL \geq \sum_{i=1}^{J} L_i$. Non-uniform RFASD would achieve a faster convergence as long as we know the Lipschitz constant on each subspace $\mathcal{V}_i$. If these constants are hard to compute, Uniform RFASD can be applied. And when the Lipschitz constants of subspaces have a small variance, the convergence rate of Uniform RFASD and Non-uniform RFASD would have a small difference. In this situation, Uniform RFASD can be first choice.

We provide two accelerated version of RFASD. In the case $(\sum_{i=1}^{J} \mu_i s_i, v)_{\mathcal{V}} = \langle -\nabla f(x^k), v\rangle$, we combine with Nesterov acceleration with a different nonuniform sampling strategy [33]. In the case $(\sum_{i=1}^{J} \mu_i s_i, v)_{\mathcal{V}} \neq \langle -\nabla f(x^k), v\rangle$, we ensemble the technique of Catalyst acceleration and RFASD gives the second accelerated scheme AFASD-II, which uses RFASD in an inner-loop to solve a regularized optimization problem and Nestrov acceleration in an outer-loop.

For convex functions, we prove a sublinear convergence and for strongly convex function, we get linear convergence rate. In order to achieve accuracy ϵ, the complexity of RFASD, AFASD-I and AFASD-II schemes are listed in TABLE 6.18. Remark: A $\log$ factor dependent on parameters is

Table 6.18: RFASD & AFASD: Number of iterations to achieve accuracy ϵ for the optimality gap

	Convex	Strongly Convex
Uniform RFASD	$O\left(\frac{\gamma^2}{\delta^2}\frac{JL}{\epsilon}\kappa_{\max}\right)$	$O\left(\frac{\gamma^2}{\delta^2}\frac{JL}{\mu}\kappa_{\max}\log\frac{1}{\epsilon}\right)$
Importance RFASD	$O\left(\frac{\gamma^2}{\delta^2}\frac{\ell}{\epsilon}\kappa_{\max}\right)$	$O\left(\frac{\gamma^2}{\delta^2}\frac{\ell}{\mu}\kappa_{\max}\log\frac{1}{\epsilon}\right)$
AFASD-I	$O\left(\frac{\ell_S}{\sqrt{\epsilon}}\right)$	$O\left(\frac{\ell_S}{\mu}\log\frac{1}{\epsilon}\right)$
Uniform AFASD-II	$\tilde{\mathcal{O}}\left(\frac{\gamma^2}{\delta^2}J\sqrt{\frac{L}{\epsilon}}\kappa_{\max}\log\frac{1}{\epsilon}\right)$	$\tilde{\mathcal{O}}\left(\frac{\gamma^2}{\delta^2}J\sqrt{\frac{L}{\mu}}\kappa_{\max}\log\frac{1}{\epsilon}\right)$
Importance AFASD-II	$\tilde{\mathcal{O}}\left(\frac{\gamma^2}{\delta^2}\sqrt{J}\sqrt{\frac{\ell}{\epsilon}}\kappa_{\max}\log\frac{1}{\epsilon}\right)$	$\tilde{\mathcal{O}}\left(\frac{\gamma^2}{\delta^2}\sqrt{J}\sqrt{\frac{\ell}{\mu}}\kappa_{\max}\log\frac{1}{\epsilon}\right)$

hidden in the notation $\tilde{\mathcal{O}}$.

Gradient Descent (GD), Coordinate Descent (CD), Block Coordinate Descent (BCD), Preconditioned Randomized Fast Subspace Descent (Pre-RFASD) and Randomized Full Approximation Storage Scheme (RFAS) are specific examples of RFASD scheme.Convergence analysis of these methods can be recovered by analysis of RFASD. Complexity of them are listed in TABLE 6.19.

Table 6.19: Examples: Number of iterations to achieve accuracy ϵ for the optimality gap.

	Convex	Strongly convex	Cost of one iteration
GD	$O\left(\frac{L}{\epsilon}\right)$	$O\left(\frac{L}{\mu}\log\frac{1}{\epsilon}\right)$	$O(N^2)$
CD	$O\left(\frac{NL}{\epsilon}\right)$	$O\left(\frac{NL}{\mu}\log\frac{1}{\epsilon}\right)$	$O(N)$
Uniform BCD $n = J$	$O\left(\frac{JL}{\epsilon}\right)$	$O\left(\frac{JL}{\mu}\log\frac{1}{\epsilon}\right)$	$O(JN)$
Importance BCD $n = J$	$O\left(\frac{\ell}{\epsilon}\right)$	$O\left(\frac{\ell}{\mu}\log\frac{1}{\epsilon}\right)$	$O(JN)$
Uniform Pre-RFASD	$O\left(\frac{C_A JL\kappa_{\max}}{\epsilon}\right)$	$O\left(\frac{C_A JL\kappa_{\max}}{\mu}\log\frac{1}{\epsilon}\right)$	$O(N)$
Importance Pre-RFASD	$O\left(\frac{C_A \ell\kappa_{\max}}{\epsilon}\right)$	$O\left(\frac{C_A \ell\kappa_{\max}}{\mu}\log\frac{1}{\epsilon}\right)$	$O(N)$
Uniform FAS	$O\left(\frac{C_A JL\kappa_{\max}}{\epsilon}\right)$	$O\left(\frac{C_A JL\kappa_{\max}}{\mu}\log\frac{1}{\epsilon}\right)$	$O(N)$
Importance FAS	$O\left(\frac{C_A \ell\kappa_{\max}}{\epsilon}\right)$	$O\left(\frac{C_A \ell\kappa_{\max}}{\mu}\log\frac{1}{\epsilon}\right)$	$O(N)$

Chapter 7

Conclusion

In this thesis we proposed one randomized fast solver based on row sampling for least squares problem and one general scheme for non-constraint convex and smooth convex optimization methods, which exhibit huge potential for solving optimization problems in data science including but not limited to linear regression, logistic regression and support vector machines. Introduction of linear and nonlinear problems arising in data science are introduced in Chapter 2.

In the first job presented in thesis, we construct a randomized row sampling method which aims to solve the least squares problems with matrix A is ill conditioned, sparse, overdetermined matrix of size $m \times n$ with $m \gg n$. By row sampling , we capture the behavior of the high frequency of the matrix and also reduce the size of matrix. The the preconditioner is constructed by applying Gauss Seidel to approximate the solution of sampled system $A_s^\intercal A_s e = r$. After we get a good approximation via preconditioner, CG is applied to get the solution. The preconditioner we constructed is good since sampling captures the high frequency of the matrix and Gauss Seidel for sampled system guarantee a good guess for the original system. This preprocess also reduce complexity since by sampling we do not have to form the original system $A^\intercal A$ of complexity $O(mn^2)$ and via Gauss Seidel process, we do not have to compute the inverse of $A_s^\intercal A_s$, which costs $O(n^3)$.

The last but not least is that this RS method is easy to implement. All we need to implement is the row sampling, Gauss Seidel process and PCG. Existing methods for least squares problem is presented in Chapter 3. Our construction of non-uniform row sampling fast least squares solver and numerical results of our solver compared to diagonal preconditioned CG on various examples are listed in Chapter 4.

For the second job, we provide a general scheme RFASD for solving convex or strongly convex minimization problem $\min_{x \in \mathcal{V}} f(x)$. This method is constructed via subspace decomposition of $\mathcal{V}$. The general update scheme is

$$x^{k+1} = x^k + \alpha^k s_{i_k}.$$

where s_{i_k} satisfies descent direction and approximate gradient assumptions and α^k is chosen to be optimal. RFASD achieves linear convergence for strongly-convex problem and sub-linear convergence for convex problem. The convergence rate and complexity depend on three factors: $\frac{\gamma^2}{\delta^2}$ measures how good direction s_{i_k} approximate negative gradient direction, $\frac{L}{\mu}$ is condition number of f on $\mathcal{V}$ while $\kappa_{\max}$ is the largest condition number of f_i on subspace $\mathcal{V}_i$. It shows a trade-off in practice. The optimal strategy is find a global preconditioner to reduce $\frac{L}{\mu}$, a stable space decomposition such that $\frac{\gamma^2}{\delta^2}$ is controllable and subspace preconditioner A_i to reduce $\kappa_{\max}$. The review of development of gradient descent type methods including gradient descent, randomized coordinate descent, block randomized coordinate descent, stochastic gradient descent and mini-batch stochastic gradient descent methods, their application to linear problems and dual relation are presented in Chapter 5. In Chapter 6, we present the RFASD as a general scheme. Convergence in expectation for smooth convex and strongly convex objective functions are shown. Specifically, we point out GD, CD, BCD and RFAS are specific examples of RFASD. Numerical experiments on Nesterov's worst function [32] and two accelerated schemes of RFASD are provided.

Bibliography

[1] Air tools ii, version 1.0. 56

[2] Randomized kazmarz method. `https://en.wikipedia.org/wiki/Kaczmarz_method`, note = Accessed: 2016-08-15. 28

[3] G. Aurélien. *Hands-on Machine Learning with Scikit-Learn & TensorFlow: Concepts, Tools, and Techniques to build Intelligent Systems*. OReilly Media, 2017. 11

[4] H. Avron, P. Maymounkov, and S. Toledo. Blendenpik: Supercharging LAPACK's least-squares solver. *SIAM Journal on Scientific Computing*, 32(3):1217–1236, 2010. 2, 31, 33, 36, 48, 50

[5] Z. Bai and Y. Yin. Limit of the smallest eigenvalue of a large dimensional sample covariance matrix. *Advances In Statistics*, pages 108–127, 2008. 48, 49

[6] M. R. M. Biemond J, Lagendijk R L. Iterative methods for image deblurring. *Proceedings of the IEEE*, 78(5):856–883, 1990. 5

[7] A. Bjorck. *Numerical methods for least squares problems*. Siam, 1996. 27

[8] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 06 2018. 79

[9] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018. 84

[10] L. Chen. Classical iterative methods. `http://www.math.uci.edu/~chenlong/226/Ch6IterativeMethod.pdf`. Accessed: 2016-08-15. 25

[11] L. CHEN, X. HU, and S. M. WISE. Convergence analysis of the fast subspace descent (fasd) method for convex optimization problems: Expanded edition. 3, 71, 89

[12] L. Chen and H. Wu. A preconditioner based on non-uniform row sampling for linear least squares problems. *arXiv preprint arXiv:1806.02968*, 2018. 3

[13] P. Chen, J. Huang, and X. Zhang. A primal-dual fixed point algorithm for minimization of the sum of three convex separable functions. *Fixed Point Theory and Applications*, 2016(1):54, 2016. 87

[14] D. Csiba and P. Richtárik. Coordinate descent face-off: Primal or dual? *arXiv preprint arXiv:1605.08982*, 2016. 86, 87

[15] P. Drineas, M. W. Mahoney, S. Muthukrishnan, and T. Sarlos. Faster least squares approximation. *arXiv preprint arXiv:0710.1435*, 2007. 2, 32

[16] P. Drineas, M. W. Mahoney, S. Muthukrishnan, and T. Sarlós. Faster least squares approximation. *Numerische mathematik*, 117(2):219–249, 2011. 31, 33, 37

[17] A. Hefny, D. Needell, and A. Ramdas. Rows vs. columns: Randomized kaczmarz or gauss-seidel for ridge regression. *arXiv preprint arXiv:1507.05844*, 2015. 86

[18] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009. 5

[19] T. Le, T. Nguyen, V. Nguyen, and D. Phung. Dual space gradient descent for online learning. In *Advances in Neural Information Processing Systems*, pages 4583–4591, 2016. 87

[20] X. Li, T. Zhao, R. Arora, H. Liu, and M. Hong. On faster convergence of cyclic block coordinate descent-type methods for strongly convex minimization. *Journal of Machine Learning Research*, 18:184–1, 2017. 14, 115

[21] H. Lin, J. Mairal, and Z. Harchaoui. A universal catalyst for first-order optimization. In *Advances in Neural Information Processing Systems*, pages 3384–3392, 2015. 91

[22] H. Lin, J. Mairal, and Z. Harchaoui. Catalyst Acceleration for First-order Convex Optimization: From Theory to Practice. 2018. 117, 125, 126, 128

[23] F. Lindsten, T. B. Schön, A. Svensson, and N. Wahlström. Probabilistic modeling–linear regression & gaussian processes. *Uppsala: Uppsala University*, 2017. 7

[24] N. Lord. Functional analysis and differential equations in abstract spaces, by s. zaidman. pp. 226.£ 36.00. 1999. isbn 1 58488 011 2 (chapman & hall/crc). *The Mathematical Gazette*, 84(499):184–185, 2000. 18

[25] Z. Lu. Randomized block proximal damped newton method for composite self-concordant minimization. *SIAM Journal on Optimization*, 27(3):1910–1942, 2017. 61

[26] Z.-Q. Luo and P. Tseng. On the convergence of the coordinate descent method for convex differentiable minimization. *Journal of Optimization Theory and Applications*, 72(1):7–35, 1992. 60, 71

[27] M. W. Mahoney. Lecture notes on randomized linear algebra. *arXiv preprint arXiv:1608.04481*, 2016. 38, 43

[28] E. Moulines and F. R. Bach. Non-asymptotic analysis of stochastic approximation algorithms for machine learning. In *Advances in Neural Information Processing Systems*, pages 451–459, 2011. 83

[29] D. Needell. Randomized Kaczmarz solver for noisy linear systems. *BIT Numerical Mathematics*, 50(2):395–403, 2010. 32

[30] D. Needell, N. Srebro, and R. Ward. Stochastic gradient descent, weighted sampling, and the randomized kaczmarz algorithm. *Mathematical Programming*, 155(1-2):549–573, 2016. 84

[31] Y. Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012. 60, 61, 71, 74, 75, 101, 102, 103

[32] Y. Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013. 4, 90, 100, 117, 129, 133

[33] Y. Nesterov and S. Stich. Efficiency of the Accelerated Coordinate Descent Method on Structured Optimization Problems. *SIAM Journal on Optimization*, 27(1):110–123, Jan. 2017. 90, 117, 122, 130

[34] J. Neter, M. H. Kutner, C. J. Nachtsheim, and W. Wasserman. *Applied linear statistical models*, volume 4. Irwin Chicago, 1996. 5

[35] P. Oswald and W. Zhou. Convergence analysis for Kaczmarz-type methods in a Hilbert space framework. *Linear Algebra and its Applications*, 478:131–161, 2015. 30

[36] C. C. Paige and M. A. Saunders. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM transactions on mathematical software*, 8(1):43–71, 1982. 27

[37] V. Rokhlin and M. Tygert. A fast randomized algorithm for overdetermined linear least-squares regression. *Proceedings of the National Academy of Sciences*, 105(36):13212–13217, 2008. 2, 31, 33

[38] S. Shalev-Shwartz and T. Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, 14(Feb):567–599, 2013. 87

[39] S. Shalev-Shwartz and T. Zhang. Accelerated proximal stochastic dual coordinate ascent for regularized loss minimization. In *International conference on machine learning*, pages 64–72, 2014. 87

[40] C. Shalizi. Logistic regression. `https://www.stat.cmu.edu/~cshalizi/uADA/12/lectures/ch12.pdf`. 10

[41] T. Strohmer and R. Vershynin. A randomized Kaczmarz algorithm with exponential convergence. *Journal of Fourier Analysis and Applications*, 15(2):262, 2009. 31, 32, 84

[42] T. Strohmer and R. Vershynin. A randomized kaczmarz algorithm with exponential convergence. *Journal of Fourier Analysis and Applications*, 15(2):262, 2009. 83

[43] K. Stüben. A review of algebraic multigrid. In *Numerical Analysis: Historical Developments in the 20th Century*, pages 331–359. Elsevier, 2001. 89

[44] T. Suzuki. Stochastic dual coordinate ascent with alternating direction method of multipliers. In *International Conference on Machine Learning*, pages 736–744, 2014. 87

[45] J. A. Tropp. User-friendly tail bounds for sums of random matrices. *Foundations of computational mathematics*, 12(4):389–434, 2012. 38

[46] P. Tseng. Convergence of a block coordinate descent method for nondifferentiable minimization. *Journal of optimization theory and applications*, 109(3):475–494, 2001. 71

[47] C. Wang, A. Agaskar, and Y. M. Lu. Randomized Kaczmarz algorithm for inconsistent linear systems: An exact MSE analysis. In *Sampling Theory and Applications (SampTA), 2015 International Conference on*, pages 498–502. IEEE, 2015. 32

[48] S. J. Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, Jun 2015. 61, 62, 68, 72, 77, 78

[49] J. Xu. Iterative methods by space decomposition and subspace correction. *SIAM review*, 34(4):581–613, 1992. 25

[50] J. Xu. An introduction to multilevel methods. *Wavelets, multilevel methods and elliptic PDEs*, pages 213–302, 1997. 25

[51] A. Zouzias and N. M. Freris. Randomized extended Kaczmarz for solving least squares. *SIAM Journal on Matrix Analysis and Applications*, 34(2):773–793, 2013. 32