

# Tuning Doubly Randomized Block Kaczmarz Method

*Rahul Jain*



Electrical Engineering and Computer Sciences  
University of California, Berkeley

Technical Report No. UCB/EECS-2022-52

<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2022/EECS-2022-52.html>

May 10, 2022

Copyright © 2022, by the author(s).  
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

### Acknowledgement

This work would not have been possible without the endless and constant support of so many people. I would like to thank my advisor Professor Jim Demmel for his overall guidance, mentorship and his kind words of encouragement. I would like to thank my postdoctoral fellow Riley Murray, Hengrui Luo, Younghyun Cho, Xiaoye Sherry. Li, Yang Liu, and the rest of the GPTune team for their supervision. I learned so much from the numerous and countless feedback that was given to me. I would also like to thank Professor Mahoney for his valuable editorial feedback as my second thesis reader.

This would not have been possible without my mom and dad and my sister for their unconditional support. I would also like to thank my friends for being a constant source of encouragement for me.

---

# Tuning Doubly Randomized Block Kaczmarz Method

by Rahul Jain

---

## Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences,  
University of California at Berkeley, in partial satisfaction of the requirements for the  
degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

### Committee:



---

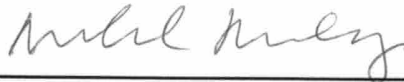
Professor James Demmel  
Research Advisor

8 May 2022

---

(Date)

\*\*\*\*\*



---

Professor Michael Mahoney  
Second Reader

5/8/22.

---

(Date)

Tuning Doubly Randomized Block Kaczmarz Method

by

Rahul Jain

A thesis submitted in partial satisfaction of the  
requirements for the degree of

5th Years

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Jim Demmel, Chair  
Professor Michael Mahoney

Spring 2022

Abstract

Tuning Doubly Randomized Block Kaczmarz Method

by

Rahul Jain

5th Years in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Jim Demmel, Chair

In the past, algorithms for solving linear systems of equations have focused on finding a solution that is not only stable with respect to small changes to the input, but with a very small error with respect to the analytical solution. However, this comes at the cost of an increased runtime. There has been an increased need to find a solution to linear system in a small amount of time, requiring modest accuracy. Randomized algorithms are quite beneficial in this aspect in that they can have a smaller runtime than their deterministic counterparts. In this thesis, we explore and then modify Randomized Block Kaczmarz, a randomized algorithm for solving overdetermined linear systems of equations, to see how practically effective it can be.

# Contents

<b>Contents</b>	<b>i</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Contribution . . . . .	1
<b>2 Kaczmarz Method</b>	<b>3</b>
2.1 Simple Kaczmarz Method . . . . .	3
2.2 Block Kaczmarz Method . . . . .	4
<b>3 Randomized Linear Algebra</b>	<b>7</b>
3.1 Introduction . . . . .	7
3.2 Preliminaries/Notation . . . . .	7
3.3 Subspace Embeddings . . . . .	8
3.4 Sketch and Precondition . . . . .	10
<b>4 Iterative Refinement</b>	<b>11</b>
4.1 Setup . . . . .	11
4.2 Iterative Improvement . . . . .	13
<b>5 Doubly Randomized Block Kaczmarz Method</b>	<b>16</b>
5.1 Preliminaries . . . . .	16
5.2 Algorithm . . . . .	16
5.3 Numerical Experiments . . . . .	18
<b>6 Tuning Experiments</b>	<b>20</b>
6.1 Using Auto-tuners . . . . .	20
6.2 Numerical Experiments . . . . .	21
<b>7 Related Work</b>	<b>25</b>
7.1 Clustered Kaczmarz Method . . . . .	25
7.2 Applications of Block Kaczmarz . . . . .	26
<b>8 Conclusion</b>	<b>28</b>

**Bibliography**

## Acknowledgments

This work would not have been possible without the endless and constant support of so many people. First and foremost, I would like to thank my advisor Professor Jim Demmel for his overall guidance, mentorship and his kind words of encouragement. I would like to thank my postdoctoral fellow Riley Murray, Hengrui Luo, Younghyun Cho, Xiaoye Sherry. Li, Yang Liu, and the rest of the GPTune team for their supervision and always being there to help me out and always dealing with my questions as I worked through this thesis. I learned so much from the numerous and countless feedback that was given to me. I would also like to thank Professor Mahoney for his valuable editorial feedback as my second thesis reader. I am ever so grateful to have worked with such an amazing team here at Berkeley.

This would not have been possible without my mom and dad and my sister for their unconditional support throughout my entire time here at Berkeley. I would also like to thank my friends for being a constant source of encouragement for me as I worked through this thesis and for celebrating with me every accomplishment along the way.



# Chapter 1

## Introduction

Linear systems of equations play a fundamental role in numerical linear algebra as they are used to model various problems in fields ranging from economics to physics to computer science to statistics. Furthermore, linear systems are used as they tend to be more simple and interpretable than other modeling techniques. These systems of equations can be represented in matrix form,  $Kz = h$ <sup>1</sup>, where  $K \in \mathbb{R}^{m,n}$  is the coefficient matrix and  $h \in \mathbb{R}^m$  is called the response vector and  $z \in \mathbb{R}^n$  is the vector of unknowns that one is trying to solve for. There has been much research done into developing computational algorithms to solve linear systems of equations. These algorithms range from direct algorithms such as Gaussian elimination in cases when  $m = n$  and QR factorization in cases when  $m \geq n$  to iterative algorithms, which generate a sequence of approximate answers. Examples of common iterative algorithms to solve linear systems are Conjugate Gradient method, Jacobi method, Gauss-Seidel, and successive over-relaxation [7]. There is an increasing need to make these computational algorithms as efficient as possible with regards to runtime and accuracy of the solution. For the most part, many traditional algorithms for solving linear systems of equations have focused on finding a solution that is as accurate as can be. However, in many fields, a highly accurate answer is not the priority but rather the time it takes to get a solution. As a result, there has been a rise in the development of scientific algorithms that incorporate randomization.

### 1.1 Contribution

In this thesis, the focus is primarily on modifying Randomized Block Kaczmarz method, a randomized iterative solver, to solve overdetermined linear systems of equations and evaluating its performance. In particular, we want to see if we can make Randomized Block Kaczmarz practically effective as it has previously not demonstrated practical performance. In Chapter 2, we will go over a variety of Kaczmarz methods and their convergence rates to understand the various aspects of the methods. Then Chapter 3 will focus on providing

---

<sup>1</sup>We will refer to the more common  $Ax = b$  for underdetermined systems later on.

background into randomized linear algebra and the essentials that will be necessary for further understanding. Chapter 4 and Chapter 5 will focus on the development of the modified algorithm, which will be called Doubly Randomized Block Kaczmarz method. In particular, we will also look at its convergence rate as well. Furthermore, the algorithm that we develop has multiple tuning parameters that affect the convergence rate and runtime of the algorithm. To automate this process in a quick manner, we will use auto-tuners to find the optimal tuning parameters to optimize for the runtime of our algorithm. Chapter 6 will provide a background for tuning and will present the results of the auto-tuners run on Doubly Randomized Block Kaczmarz method. In Chapter 7, we will present a variety of different variants of Kaczmarz methods as well as practical applications.

# Chapter 2

## Kaczmarz Method

Kaczmarz method [13] is an iterative solver for solving overdetermined systems of equations. More specifically, given  $K \in \mathbb{R}^{m,n}$  and  $h \in \mathbb{R}^m$  where  $m > n$ ,  $\text{rank}(K) = n$ , the goal is to minimize the following objective function.<sup>1</sup> Find  $z^* \in \mathbb{R}^n$  such that

$$z^* = \underset{z \in \mathbb{R}^n}{\text{argmin}} \|Kz - h\|_2^2$$

This problem is commonly referred to as the “Least Squares” problem. This commonly arises when one wants to fit a model to a set of data points to extract a pattern. Furthermore, let  $z^j \in \mathbb{R}^n$  be the value of  $z$  found at the  $j^{\text{th}}$  iteration.

### 2.1 Simple Kaczmarz Method

Let us use the same setup as above. Additionally, let  $K_t$  denote the  $t^{\text{th}}$  row of  $K$  and let  $h_t$  be the  $t^{\text{th}}$  entry of  $h$ . The simple Kaczmarz method works by generating a sequence of guesses to the solution  $(z^0, z^1, \dots, z^k)$  where  $k$  is the number of iterations for which the method runs. First, we pick a row index  $t = f(k)$ , then

$$z^{k+1} = z^k + \frac{h_t - K_t z^k}{\|K_t\|_2^2} K_t^T, k \geq 0$$

$f(k)$  is referred to as the control strategy that determines which row is chosen at the  $k^{\text{th}}$  iteration. This is repeated until the maximum number of iterations has been completed or a convergence tolerance is met. This can be seen as projecting  $z^k$  onto the hyperplane defined by  $B = \{x \mid K_t x = h_t \mid x \in \mathbb{R}^n\}$ .

#### Row selection

The classic strategy simply cycles through all rows in a sequential manner. However, in many cases, the convergence with this control strategy can be quite slow. In practice, the

---

<sup>1</sup>Can also be within  $\mathbb{C}^{m,n}$ ,  $\mathbb{C}^m$ . Without loss of generality, we will work in  $\mathbb{R}$ .

most common strategy is to randomly select a row  $K_t$  with probability  $\frac{\|K_t\|_2^2}{\|K\|_F^2}$ . The reasoning behind this can be seen as emphasizing rows/equations with a larger norm.

## Convergence

Without loss of generality, let's assume that each row of  $K$  is standardized with respect to the  $\ell_2$  norm.<sup>2</sup> Let us also assume that the control strategy is sampling rows with probability proportional to their norm. Since all the norms are the same, this is equivalent to sampling each row with probability  $\frac{1}{m}$ . Thus, the convergence of the Simple Kaczmarz method is presented in Needell [19]:

$$\mathbb{E}\|z^k - z^*\|_2^2 \leq \left[1 - \frac{\sigma_{\min}^2(K)}{m}\right]^k \|z^0 - z^*\|_2^2 + \frac{m\|Kz^* - h\|_\infty^2}{\sigma_{\min}^2(K)}$$

There are some observations that can be made from the formula for the convergence. First, we can see that the rate of convergence is exponential and is dependent on the smallest singular value of  $K$ , but there does not appear to be a dependence on  $\sigma_{\max}(K)$ . This is mainly because each row has unit norm, so  $1 \leq \sigma_{\max}(K) \leq \sqrt{m}$ , so we should not expect to see  $\sigma_{\max}(K)$  in the convergence bound. It also seems that as  $\sigma_{\min}(K)$  increases, the rate of convergence increases as both terms on the right side decrease. The second observation is that the second term on the right side is independent of  $k$ . Thus, as  $k \rightarrow \infty$ , the first term on the right side becomes zero while the second term remains, thus

$$\lim_{k \rightarrow \infty} \mathbb{E}\|z^k - z^*\|_2^2 \leq \frac{m\|Kz^* - h\|_\infty^2}{\sigma_{\min}^2(K)}$$

This implies that  $z^k$  from Kaczmarz converges to a fixed ball around  $z^*$  with radius  $\frac{m\|Kz^* - h\|_\infty^2}{\sigma_{\min}^2(K)}$ .

## 2.2 Block Kaczmarz Method

As we can see from the simple Kaczmarz method, we can easily extend this simple idea of projecting  $z$  onto the solution space of a single equation to the solution space of multiple equations at a time. This introduces the idea behind the Block Kaczmarz method. Before we go into the algorithm, let us define some important terms. The number of equations that we decide to project  $z$  onto is referred to as the “block size” which we will refer to as  $b$ . Let  $\tau = \{\tau_1, \tau_2, \dots, \tau_r\}$  be a partition of the row indices of  $K$  where  $r = \lfloor \frac{m}{b} \rfloor$  where  $|\tau_i| = b$  for  $1 \leq i \leq r$ .<sup>3</sup> What this means is that when  $i \neq j$ ,  $\tau_i \cap \tau_j = \emptyset$  and  $\cup_{k=1}^r \tau_k = [1, 2, 3, \dots, m]$ . Furthermore, we will let  $K_\tau$  be the rows of  $K$  indexed by  $\tau$ ,  $h_\tau$  be the values of  $h$  indexed

<sup>2</sup>We can always scale each equation  $t$  of the system to make the matrix  $K$  standardized.

<sup>3</sup>When  $b$  does not perfectly divide  $m$ , the last block will contain  $\leq b$  rows. Without loss of generality, we assume  $b$  divides  $m$ .

by  $\tau$ , and  $K_\tau^\dagger$  denote the Moore-Penrose pseudoinverse. For simplicity sake, we will choose  $\tau_1$  the first  $b$  rows of  $K$  and  $\tau_2$  to be the next  $b$  rows and so on. This will be a valid partition according to the above definition.

---

**Algorithm 1** Block Kaczmarz Method
 

---

**Require:**  $K \in \mathbb{R}^{m,n}$ ,  $h \in \mathbb{R}^m$ ,  $b$  (Block Size),  $z^0 \in \mathbb{R}^n$ ,  $\epsilon \geq 0$ , epochs,  $f$  Control Strategy

- 1:  $r = \lfloor \frac{m}{b} \rfloor$
- 2:  $\tau = \{\tau_1, \tau_2, \dots, \tau_r\} \triangleright \tau_1$  contains the first  $b$  rows of  $K$ ,  $\tau_2$  contains the next  $b$  rows, etc
- 3:  $i = 0$
- 4: **while**  $i \leq \text{epochs}$  and  $\|Kz^i - h\|_2 > \epsilon$  **do**
- 5:     Pick  $\tau = f(i)$
- 6:      $w^i = \mathbf{0}^n$
- 7:      $z^{i+1} = z^i + K_\tau^\dagger(h_\tau - K_\tau z^i)$
- 8:      $i = i + 1$
- 9: **end while**

---

We can see that the expensive part of Block Kaczmarz method occurs on line 7, where we have to compute  $K_\tau^\dagger$ . Always, instead of computing  $K_\tau^\dagger(h_\tau - K_\tau z^i)$  directly as in line 7, it is preferred to solve  $K_\tau w^i = h_\tau - K_\tau z^i$  for  $w^i \in \mathbb{R}^n$ . The reason for this is because computing  $K_\tau^\dagger$  directly and applying it directly to  $h_\tau - K_\tau z^i$  is not as accurate as solving  $K_\tau w^i = h_\tau - K_\tau z^i$  for  $w^i \in \mathbb{R}^n$ . Furthermore, computing  $K_\tau^\dagger$  directly can also result in numerically unstable results. When  $b = 1$  (one row at a time), Block Kaczmarz reduces to the simple Kaczmarz method and when  $b = m$  (all of the rows), you are projecting  $z^i$  onto the solution space of the matrix. In this paper, we assume that  $b$  (the block size) is a fraction of the number of columns; thus we can see here that  $K_\tau w^i = h_\tau - K_\tau z^i$  is an underdetermined system of equations. This is the bottleneck in randomized Block Kaczmarz as one needs to repeatedly solve an underdetermined linear system of equations per iteration. We view any algorithm that solves this underdetermined system of equations as a black-box algorithm.

## Randomized Block Kaczmarz Method

In similar fashion to simple Kaczmarz method, there are multiple random control strategies for choosing the block  $\tau_k$  to project  $z^k$  onto [19]. The one which we will focus on is the one that will randomly permute the rows of  $K$  per iteration and then will choose the blocks in a sequential manner. This process is repeated per iteration.

## Convergence of Randomized Block Kaczmarz

Here we will analyze the convergence rate of Randomized Block Kaczmarz method where one samples a block from  $\tau$  uniformly at random. Before we introduce the convergence rate, it will be helpful to introduce some definitions that were introduced in [19].

### Definitions

A  $(r, \alpha, \beta)$  row paving of the matrix  $K$  is a partition  $\tau = \{\tau_1, \tau_2, \dots, \tau_r\}$  such that

$$\alpha \leq \lambda_{\min}(K_{\tau}K_{\tau}^T) \text{ and } \lambda_{\max}(K_{\tau}K_{\tau}^T) \leq \beta \text{ for all } \tau_k \in \tau$$

Here we can see that

$$\frac{\lambda_{\max}(K_{\tau}K_{\tau}^T)}{\lambda_{\min}(K_{\tau}K_{\tau}^T)} \leq \frac{\beta}{\alpha}$$

Thus we can see  $\frac{\beta}{\alpha}$  is an upper bound on the condition number of every block of the partition  $\tau$  of  $K$ .

### Convergence Rate

The convergence rate [19] is as follows : Given a  $(r, \alpha, \beta)$  row paving  $\tau$ ,

$$\mathbb{E}\|z^k - z^*\|_2^2 \leq \left[1 - \frac{\sigma_{\min}^2(K)}{\beta r}\right]^k \|z^0 - z^*\|_2^2 + \frac{\beta}{\alpha} * \frac{\|Kz^* - h\|_2^2}{\sigma_{\min}^2(K)}$$

There are a few observations that we can make. First, there still seems to be a dependence on  $\sigma_{\min}^2(K)$  and no dependence on  $\sigma_{\max}^2(K)$ . We also still see an exponential rate of convergence as shown by the first term on the right side of the inequality. Furthermore, when the system of equations is consistent, the second term on the right disappears (as  $\|Kz^* - h\|_2^2 = 0$ ) and our convergence rate is only dependent on  $\beta$  and  $\sigma_{\min}^2(K)$ . In general we hope for  $\frac{\beta}{\alpha}$  to be close to 1.

# Chapter 3

## Randomized Linear Algebra

### 3.1 Introduction

In the past, work in numerical linear algebra has been concerned with finding the correct answer with an error close to the machine epsilon,  $\epsilon$ , in a reasonable amount of time. Although there has been success in finding fast algorithms that accomplish this goal, there are still many algorithms for which finding an answer with error  $\epsilon$  takes a long time and requires many computational resources. Often, in many disciplines dealing with big data, it suffices to find a solution that is correct with an error much larger than  $\epsilon$  in a reasonable amount of time. Sometimes, many fewer digits of accuracy suffice in many cases. As a result, there has been an emergence of randomized matrix algorithms, especially in large-scale machine learning and big data analysis. The advantages that randomization presents are that it leads to simpler and faster algorithms. There is still research being done on constructing randomized linear algebra algorithms with theoretical guarantees on the error. In this thesis, the goal is to evaluate our doubly Randomized Block Kaczmarz method and see if it is competitive against traditional iterative solvers for linear systems.

### 3.2 Preliminaries/Notation

Unless noted, we can assume that  $\|\cdot\|$  refers to the  $\ell_2$  norm when the quantity inside is a vector and the operator norm when the quantity inside is a matrix. Furthermore, a Rademacher random variable is defined as a random variable with values  $\{-1, 1\}$  with probability  $\frac{1}{2}$  or  $\{z \in \mathbb{C} : |z| = 1\}$  with uniform probability. Without loss of generality, we will focus on space  $\mathbb{R}$  in this chapter.

### 3.3 Subspace Embeddings

In the context of randomized linear algebra, it is common to project an entire matrix onto a much lower dimensional space. There are multiple reasons for this. The first one is that projecting onto a lower dimensional space results in a smaller representation of the data which can then be used in any algorithms down the line. This can be used to decrease the runtime of the algorithm where the matrix is very large, which commonly arises in big data analysis. Furthermore, many matrices tend to be of low rank, containing many redundancies.

These projections are done by applying a subspace embedding  $S$ , a sketching operator, to the input matrix. A sketching operator is simply a random linear operator that is used to apply to a matrix. More specifically, a subspace embedding  $S \in \mathbb{R}^{d,m}$  for  $A \in \mathbb{R}^{m,n}$  is defined as an sketching operator that when applying  $S$  to  $A$ , it condenses the range of  $A$  from the dimension  $m$  to  $d \ll m$  [1]. The resulting matrix is  $SA \in \mathbb{R}^{d,n}$  which is referred to as the sketch of the matrix  $A$ . The main property of interest is that we want to ensure that  $SA$  still contains the same information as  $A$  to a certain level of accuracy. This motivates the following definition.

**Definition 1** ([1]).  $S \in \mathbb{R}^{d,m}$  is a  $\eta$ -embedding for the range( $A$ ) if

$$(1 - \eta)\|Ax\|_2 \leq \|SAx\|_2 \leq (1 + \eta)\|Ax\|_2$$

$\eta$  is referred to as the distortion factor, and ideally we would like it to be close to 0.

#### Gaussian Sketching Operator

The Gaussian sketching operator is one of the most used sketching operators due to its simplicity. A Gaussian sketching operator  $S \in \mathbb{R}^{d,m}$  is constructed as follows. Let  $\tilde{S} \in \mathbb{R}^{d,m}$  be a matrix such that each entry is drawn independently from a  $\mathcal{N}(0, 1)$  distribution. Often, it is common to scale  $\tilde{S}$  such that  $\mathbb{E}[S^T S] = I_m$ , as  $\mathbb{E}[S^T S]$  is the covariance matrix of  $S$ . However, this is not always required. However, we get  $S = \frac{1}{\sqrt{d}}\tilde{S} \in \mathbb{R}^{d,m}$ . One of the disadvantages is that constructing a Gaussian requires one to generate  $md$  i.i.d. samples from a  $\mathcal{N}(0, 1)$  distribution, which is costly in higher dimensions [26]. However, this is much cheaper to construct than an orthonormal sketching operator, as we will see below.

#### Orthonormal Sketching Operator

An orthonormal sketching operator  $S \in \mathbb{R}^{d,m}$  is constructed as follows [18].

1. Construct a Gaussian sketching operator  $\tilde{S} \in \mathbb{R}^{m,d}$
2. Construct the QR factorization of  $\tilde{S} = QR$  where  $Q \in \mathbb{R}^{m,d}$  and  $R \in \mathbb{R}^{d,d}$ .
3.  $S = (Q * \text{sign}(\text{diag}(R)))^T \in \mathbb{R}^{d,m}$  where  $\text{diag}(R) \in \mathbb{R}^d$  contains the diagonal entries of  $R$  in a vector. Furthermore,  $\text{sign}(\text{diag}(R)) \in \mathbb{R}^d$  is a vector where  $\text{sign}(\text{diag}(R))_i = 1$  if  $\text{diag}(R)_i \geq 0$  and -1 otherwise.



One of the key advantages of using an orthonormal sketching operator is that it is numerically stable, thus providing less room for errors. However, one of the main disadvantages is that it is quite expensive to construct, as one needs to not only construct a Gaussian sketching operator but also compute the QR factorization of the resulting sketch operator.

### Sparse Sketching Operator based on Sparse Johnson-Lindenstrauss Transform (SJLT)

While there is not a standardized version of what a sparse sketching operator is, we will go over one particular definition. A sparse sketching operator based on SJLT is constructed as follows. Given 2 parameters,  $v, k \leq d$ , each column  $v_i$  of  $S \in \mathbb{R}^{d,m}$  is constructed by selecting  $k$  out of the  $d$  total coordinates uniformly at random and setting the components to be  $v$  with probability  $\frac{1}{2}$  or  $-v$  with probability  $\frac{1}{2}$  [17]. The rest of the entries in the vector are set to 0. One can use the Hanson-Wright inequality to bound the probability that the distortion factor is within a factor  $\epsilon$  [6].

### Sparse Sign Sketching Operator

A sparse sign sketching operator  $S \in \mathbb{R}^{d,m}$  is constructed as follows. Given a parameter  $2 \leq k \leq d$ , referred to as the “sparsity” of the sketch, we construct  $v_1, v_2, \dots, v_m \in \mathbb{R}^d$  such that each  $v_i \in \mathbb{R}^d$  contains  $k$  i.i.d Rademacher variables in  $k$  uniformly random coordinates [17]. Then  $S$  is constructed as follows

$$S = \sqrt{\frac{m}{k}} \begin{bmatrix} | & | & \dots & | \\ v_1 & v_2 & \dots & v_m \\ | & | & \dots & | \end{bmatrix} \quad (3.1)$$

In practice, it was shown in Tropp, Turtsever, Udell, and Cevher [27], that  $k = \min\{d, 8\}$  works, however there is still much work to be done. It is shown that matrix-vector multiplication with  $S$  takes  $\mathcal{O}(km)$ . One of the main disadvantages of this operator is that it is quite difficult to implement, as one must take advantage of the sparsity structure of  $S$  to achieve good performance. However, there are highly optimized sparse matrix-vector multiplication methods available in widely available libraries ranging from Intel MKL [12] to the Harwell Subroutine library [24].

### Subsampled Randomized Fast Trigonometric Transform (SRTT)

A subsampled randomized fast trigonometric transform sketching operator  $S \in \mathbb{R}^{d,m}$  is constructed as follows.

$$S = \sqrt{\frac{m}{d}} * R * T * F * \Pi$$

where  $R, T, F, \Pi$  are as follows.

- $\Pi \in \mathbb{R}^{m,m}$  - random permutation matrix (optional)
- $F \in \mathbb{R}^{m,m}$  - diagonal matrix of Rademacher variables
- $T \in \mathbb{R}^{m,m}$  - fast trigonometric transform. The choice of trigonometric transform can be DTFT (Discrete Time Fourier Transform) [20], DCT (Discrete Cosine Transform) [20], Walsh Hadamard Transform [25], or the discrete Hartley Transform [11].
- $R \in \mathbb{R}^{d,m}$  - A matrix that randomly samples  $d$  entries from the  $\mathbb{R}^m$  vector.

It can be shown that a matrix-vector multiplication with  $S$  would take  $\mathcal{O}(m \log d)$  [27]. However, while these do perform very well in practice as they are very cheap to store and because they can rapidly be multiplied by matrices to form sketches. However, they are very hard to implement in practice [1], although there are many highly tuned libraries available that implement SRTT that one can use.

### 3.4 Sketch and Precondition

The sketch and precondition framework is one where one obtains a sketch of the matrix (ie, given  $A$ , calculates  $SA$  where  $S$  is a sketching operator) to use for iterative solvers regarding  $A$ . The sketch of the problem data is used to compute a “Preconditioner” [1]. The motivation is that by “preconditioning” the matrix, the condition number of the resulting problem hopefully becomes small, as to accelerate convergence of any iterative solvers used on the problem. The QR factorization is just one of many ways to find appropriate preconditioners that can be used for overdetermined linear systems. Other factorizations of  $SA$  can be used to find appropriate preconditioners [1].

# Chapter 4

## Iterative Refinement

### 4.1 Setup

In Randomized Block Kaczmarz method, each iterate includes a subproblem which is to solve an underdetermined system of equations (wide matrix). In this case, there are multiple solutions to the system. In this project, we would like to choose the one with minimum norm. This is written as <sup>1</sup>

$$\min \|x\|_2^2 \text{ subject to } Ax = b$$

We would like to explore the extent to which iterative refinement can improve upon an initial solution for this problem, which is found using a Sketch and Precondition solver from the Parla library [18], a library containing algorithms for randomized linear algebra. Iterative refinement is the process of correcting an initial approximation to the solution of a system of equations  $Ax = b$  where  $A, x, b$  have appropriate dimensions. This is typically done in due part to decrease the relative error between the computed  $x$  (the numerical solution computed by the computer) and  $x^*$  (the true value). The classic algorithm for iterative refinement in the case of square matrices follows this.

---

#### Algorithm 2 Iterative Refinement for Square Matrices

---

**Require:**  $A \in \mathbb{R}^{n,n}, b \in \mathbb{R}^n, x^0 \in \mathbb{R}^n, iterates \geq 1$

**for**  $m = 1, 2, 3, \dots, iterates$  **do**

$$r^m = Ax^m - b$$

▷ this is the residual

Approximately solve  $A\delta^m = r^m$

▷  $\delta^m$  represents the correction

$$x^{m+1} = x^m - \delta^m$$

▷ The correction is added to  $x^m$

**end for**

**return**  $\hat{x} = x^{iterates+1}$

---

<sup>1</sup>For notational convenience, we will refer to  $Kz = h$  as the original overdetermined problem, and  $Ax = b$  as the underdetermined system that we will be solving per iteration.

The intuition behind iterative refinement is that even if  $x^0$  may not be numerically accurate or have high precision, calculating the residual is numerically accurate, and thus we hope that the correction will change the current iterate of the solution in the right direction. As a result, it was initially believed that the residual should be calculated with a higher precision than the correction  $\delta^m$ . However, it was shown that calculating both the residual and correction  $\delta^m$  with the same precision would result in more numerically stable results [3].

If we want to run iterative refinement on an underdetermined system to find the solution with minimum norm, we will use an algorithm that maintains the same precision throughout the algorithm. This algorithm was given in Bjorck's "Numerical Methods for Least Squares" [3].

---

**Algorithm 3** Iterative Refinement via Fixed Precision for Underdetermined Systems

---

**Require:**  $A \in \mathbb{R}^{r,n}$ ,  $b \in \mathbb{R}^r$ ,  $x^0 \in \mathbb{R}^n$ ,  $y^0 \in \mathbb{R}^r$ ,  $iterates \geq 1$

- 1: **for**  $m = 1, 2, 3, \dots, iterates$  **do**
  - 2:      $g^m = b - Ax^m$  ▷ This is the residual
  - 3:     Approximately solve  $AA^T \delta y^m = -g^m$  ▷  $\delta y^m$  represents the correction
  - 4:      $y^{m+1} = y^m + \delta y^m$  ▷ The correction is added to  $y^m$
  - 5:      $x^{m+1} = x^m - A^T \delta y^m$  ▷ This corrects the solution  $x^m$
  - 6: **end for**
  - 7: **return**  $\hat{x} = x^{iterates}$
- 

Oftentimes,  $x^0$ ,  $y^0$  are taken to be 0, 0 respectively. Let  $k()$  refer to the  $\ell_2$  condition number of the quantity inside the parentheses and let  $\epsilon$  be the machine-dependent epsilon and  $c$  be a constant. It is shown in [4], [5], [3] that the rate of improvement of the solution to  $Ax = b$  where  $x^*$  is the optimal value can be formulated as

$$\frac{\|x^s - x^*\|_2}{\|x^{s-1} - x^*\|_2} \leq c\epsilon \min_{D>0} k(AD).$$

We will formulate the problem in line 3 as an augmented system to solve  $\delta y^m$ , which will look like

$$\begin{bmatrix} I & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x \\ \delta y^m \end{bmatrix} = \begin{bmatrix} 0 \\ g_m \end{bmatrix} \quad (4.1)$$

The equations in the augmented system can be reduced to

$$x + A^T \delta y^m = 0 \quad (4.2)$$

$$Ax = g^m. \quad (4.3)$$

When plugging in  $x = -A^T \delta y^m$ , we get  $AA^T \delta y^m = -g^m$ , which is equivalent to the equation on line 3.

## 4.2 Iterative Improvement

While iterative refinement does improve the accuracy of our solution, it comes at a cost of solving multiple systems of equations. This can oftentimes affect the overall performance of our algorithm, making it slower. This is not what we want from a randomized algorithm, as we hope to get faster performance than the deterministic counterparts. Furthermore, when solving linear systems, the condition number of the matrix can strongly impact the problem at hand. Thus, it would be wise to see how well “iterative refinement” improves the initial solution for a variety of matrices differing in condition number. By doing so, we can hope to gain more insight on whether to incorporate iterative refinement via a solver for the augmented system in Equation 4.1 into our modified Kaczmarz algorithm.

### Experiment Setup

If we let  $x^m$  represent the solution found at the  $m^{\text{th}}$  iterate of iterative refinement, the “error improvement ratio” at the  $k^{\text{th}}$  iteration is calculated as follows:

$$\log(\|x^0 - x^*\|_2^2) - \log(\|x^j - x^*\|_2^2)$$

where  $j = \operatorname{argmin}_{0 \leq i \leq k} \|x^i - x^*\|_2^2$  and  $x^0$  is the initial guess to the solution and  $x^*$  is the actual solution. In a numerical sense, this is equivalent to seeing how many extra correct digits of accuracy one gets after running Iterative Refinement after  $j$  iterates of iterative refinement.

In order to explore how well the randomized solver for our augmented system performed iterative refinement on an underdetermined system, we plot the “error improvement ratio” as a function of the row-to-column ratio of our matrix for 5 iterates of iterative refinement. The smaller the row-to-column ratio is, the more “underdetermined” the system is. Furthermore, we decided to have multiple plots, each for different condition numbers.

Let  $p$  represent the row-to-column ratio and let  $r \geq 1$  represent the specified condition number. We tested with a fixed number of columns at 5000. Let  $K'$  be the matrix for which we want to see the effects of the error improvement ratio. Since we want to see the effects of iterative refinement for very “under-determined” systems, as it is often the case in Kaczmarz methods the range of values we considered for the row-to-column ratio ranged from 0.01 to 0.2.

1. Define  $K \in \mathbb{R}^{\lfloor 5000p \rfloor, 5000}$ , where  $K_{i,j} \sim \mathcal{N}(0, 1)$ .
2. Find the SVD of  $K = U\Sigma V^T$ .
3. Construct  $\Sigma'$  such that  $(\Sigma')_{i,i} = (\frac{1}{r})^{\frac{i-1}{\lfloor 5000p \rfloor - 1}}$  for  $1 \leq i \leq \lfloor 5000p \rfloor$ .<sup>2</sup>
4.  $K' = U\Sigma'V^T \in \mathbb{R}^{\lfloor 5000p \rfloor, 5000}$ .

---

<sup>2</sup>This is a geometric sequence ranging from 1 to  $\frac{1}{r}$ .

The reason for Step 3 is we would like the condition number of our matrix to be approximately  $r$  and the condition number of matrix  $K'$  is defined as  $k(K') = \frac{\sigma_1}{\sigma_n} = \frac{1}{\frac{1}{r}} = r$ . Thus, we choose the singular values to follow a geometric sequence from 1 to  $\frac{1}{r}$ .

## Graphs

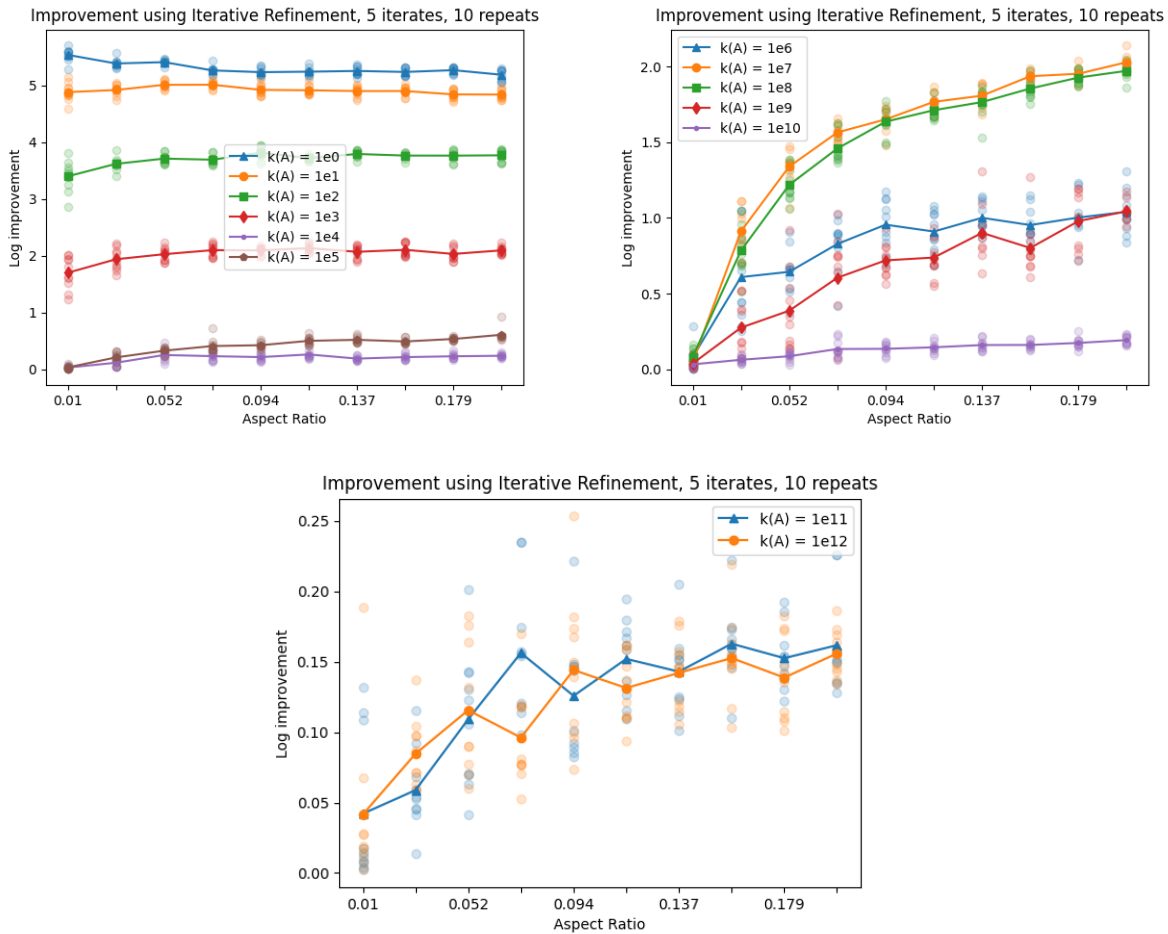


Figure 4.1: These results were generated using 5 iterations of iterative refinement, repeated 10 times for stability. The results of all the runs are shown in the background, with the mean being the dotted line

## Takeaways

There are a couple key takeaways from these two plots. For one, we can see that as the condition number of the matrix increases, the average “error improvement ratio” seems to

decrease. This is in line with what is expected as the condition number of the matrix  $K'$  should give a bound on how “poorly” the solution to a linear system involving  $K'$  will be. However, this trend does not follow throughout as we see that the error improvement ratio seems to be much higher for the matrices with condition number  $10^5$  than that of the  $10^4$  and  $10^3$  as the aspect ratio increases. What is interesting is that from condition number  $10^5$  to  $10^6$ , the average error improvement ratio seems to decrease as the condition number of the matrix increases. However, from condition number  $10^6$  onwards, the error improvement ratio seems to decrease as the condition number of the problem increases, which is what we expect. Future work can include further investigation to understand why the trend does not follow at  $10^5$ . Thus it should make sense that with an ill-conditioned matrix (a matrix with a high condition number), iterative refinement will not improve the accuracy of the solution by too much because the initial solution and the corrections will oftentimes not be calculated accurately as well.

Furthermore, from the first and second graphs we can see that for well-conditioned matrices (matrices with a low condition number), the “error improvement ratio” is quite high, reaching up to 5 or 6 extra digits of accuracy. However, it is quite interesting that as the matrices become more ill-conditioned, the drop-off in extra digits of accuracy gained is quite sharp as noted in the second graph. Nevertheless, here we can see that for matrices with condition number  $10^5$ , the “error improvement ratio” value is quite high, however, for the graph corresponding to condition number  $10^6$  and above, the improvement sharply decreases to close to 1 approximately. For matrices with condition numbers  $10^{11}$ ,  $10^{12}$ , the average “error improvement ratio” is close to 0, indicating close to no significant improvement in accuracy.

It is also quite interesting to see that as the “aspect ratio” of the matrix increases from 0.01 to 0.2, the improvement does not seem to change much for matrices with small condition numbers. However, we can see that for condition numbers  $10^6$  to  $10^{12}$ , as the aspect ratio increases, the error improvement ratio also seems to increase as well. However, it does not span a large range of values as the error improvement ratio seems to be primarily between 0 and 2 for matrices with condition number  $10^6$  to  $10^{10}$ . We also see this as well with matrices with condition number  $10^{11}$  and  $10^{12}$  where the range of error improvement ratio is even smaller, ranging from 0.05 to at most 0.15. This seems to suggest that using Iterative Refinement with the same precision throughout to solve an underdetermined linear system does not seem to be very dependent on how “underdetermined” the system is. This proves to be quite helpful in the context of Randomized Block Kaczmarz, where the choice of block size determines how “under-determined” the linear system is. As we saw in the first chapter, the choice of the block size can massively impact the convergence of Randomized Block Kaczmarz.

## Chapter 5

# Doubly Randomized Block Kaczmarz Method

Randomized Block Kaczmarz method does not seem to demonstrate practical performance. The main expense within Randomized Block Kaczmarz is that within each iteration of the algorithm, one is solving an underdetermined linear system. This can often take a long time per iteration. Here, we can hope to see if a randomized algorithm can give faster results than traditional algorithms. As mentioned in the previous chapter, we can hope to use iterative refinement to improve the accuracy of the answer to our underdetermined linear system that needs to be solved per iteration. In this chapter, we will introduce our new algorithm, called Doubly Randomized Block Kaczmarz method, which will be similar to Randomized Block Kaczmarz method while incorporating a solver for our augmented system alongside iterative refinement to solve each underdetermined linear system.

### 5.1 Preliminaries

To review, the question that we want to solve is the following:  $z^* = \operatorname{argmin}_{z \in \mathbb{R}^n} \|Kz - h\|_2^2$  where  $K \in \mathbb{R}^{m,n}$ ,  $h \in \mathbb{R}^m$ ,  $m \geq n$ . Next, the underdetermined system that we need to solve is  $K_\tau w^i = h_\tau - K_\tau z^i$  where  $K_\tau$  is the subset of rows of the original matrix  $K$  with rows indexed by  $\tau$  and  $h_\tau$  is the response vector to the original problem, but only containing elements indexed by  $\tau$ . Furthermore, let  $z^i$  be the value of  $z$  found after processing the  $i^{\text{th}}$  block in our own randomized Block Kaczmarz algorithm.

### 5.2 Algorithm

Our algorithm will be a modification of Randomized Block Kaczmarz method. The main difference is that rather than solve each “underdetermined” linear system in an exact manner, we will attempt to use a saddle point solver alongside iterative refinement. The saddle point solver that we employed is an instance of sketch and precondition. How this works



is we apply  $S$ , an arbitrary sketching operator which can be chosen by the user, to  $K_\tau^T$  to get  $SK_\tau^T$ . Then, we compute a right preconditioner for  $K_\tau^T$ , which is equivalent to a left preconditioner for  $K_\tau$ . This is done by computing the SVD of  $SK_\tau^T = U\Sigma V^T$  and setting the right preconditioner for  $K_\tau^T$  to be  $M$  where the  $i^{\text{th}}$  row of  $M$  is equal to the  $i^{\text{th}}$  row of  $V$  divided by its corresponding singular value ( $\sigma_i$ ). The “ambient dimension” of  $S$  is chosen by setting it to be a fraction of the number of rows of  $K_\tau$ , known as the sampling factor. This parameter is chosen by the user. Once this is done, preconditioned conjugate gradient algorithm [7] is called on the resulting system to get our initial answer. Afterwards, we run iterative refinement using the saddle point system solver, as to improve the accuracy of the initial solution found. The number of iterations during which we run an iterative refinement will be called *itref*. Furthermore,  $\epsilon_2 \geq 0$  will represent the termination criteria of saddle point solver + iterative refinement. Furthermore, if the  $\ell_2$  norm of the solution returned by the saddle\_solver is below  $\epsilon_3$ , we terminate. For simplicity’s sake, we will refer to this entire process of solving the underdetermined system as a black box method called `saddle_solver( $K_\tau, h_\tau - K_\tau z^i, S, \text{sampling\_factor}, \text{itref}, \epsilon_2$ )`.

## Pseudocode

---

### Algorithm 4 Doubly Randomized Block Kaczmarz Method (DRBK)

---

**Require:**  $K \in \mathbb{R}^{m,n}$ ,  $h \in \mathbb{R}^m$ ,  $b$  (Block Size),  $x^0 \in \mathbb{R}^m$ ,  $\epsilon, \epsilon_2, \epsilon_3 \geq 0$ , epochs, sampling factor  $v$ , sketch type  $S$ , *it* (number of iterations of iterative refinement)

```

1:  $r = \lfloor \frac{m}{b} \rfloor$ 
2:  $i = 0$ 
3: while  $i \leq \text{epochs}$  and  $\|Kz^i - h\|_2 / \|h\|_2 > \epsilon$  do
4:   Randomly permute the order of all of the rows of  $K$  and permute the entries of  $h$  correspondingly. Call these  $\tilde{K}$  and  $\tilde{h}$  respectively.
5:   for  $p = 1 \dots r$  do
6:     Let  $\tilde{K}_p$  contain the  $(p-1)b$  to  $\min(m, pb)$  rows of  $\tilde{K}$ 
7:     Let  $\tilde{h}_p$  contain the  $(p-1)b$  to  $\min(m, pb)$  entries of  $\tilde{h}$ 
8:      $w^i = \text{saddle\_solver}(\tilde{K}_p, \tilde{h}_p - \tilde{K}_p z^i, S, v, \text{it}, \epsilon_2)$ 
9:     if  $\|w^i\|_2 \leq \epsilon_3$  then
10:      return  $z^i$ 
11:     end if
12:      $z^{i+1} = z^i + w^i$ 
13:      $i = i + 1$ 
14:   end for
15: end while
16: return  $z^i$ 

```

---

### 5.3 Numerical Experiments

Here we will show plots displaying the convergence of our doubly Randomized Block Kaczmarz method. We plot pairs  $(i, \|Kz^i - h\|_2/\|h\|_2)$  where  $i$  represents the iteration count and  $z^i$  is the solution found by our algorithm at the  $i^{\text{th}}$  iteration. For our experiments below, we chose  $S$  to be a sparse sketching operator based on the Sparse Johnson-Lindenstrauss Transform and the sampling factor to be 3. Furthermore, we chose the block size to be  $b = \lfloor 0.2 * n \rfloor$  where  $n$  is the number of columns. The number of iterations we run of iterative refinement is 2 and  $\epsilon_1$  is  $10^{-8}$ . Furthermore,  $\epsilon_2, \epsilon_3$  are both chosen to be  $10^{-14}$ . We allowed a maximum of 60 epochs to be run. If we recall from Algorithm 4, each epoch goes through  $\lfloor \frac{m}{b} \rfloor$  iterations. In the plot below, each dot represents the progress at every 10 epochs.

#### Convergence Plots

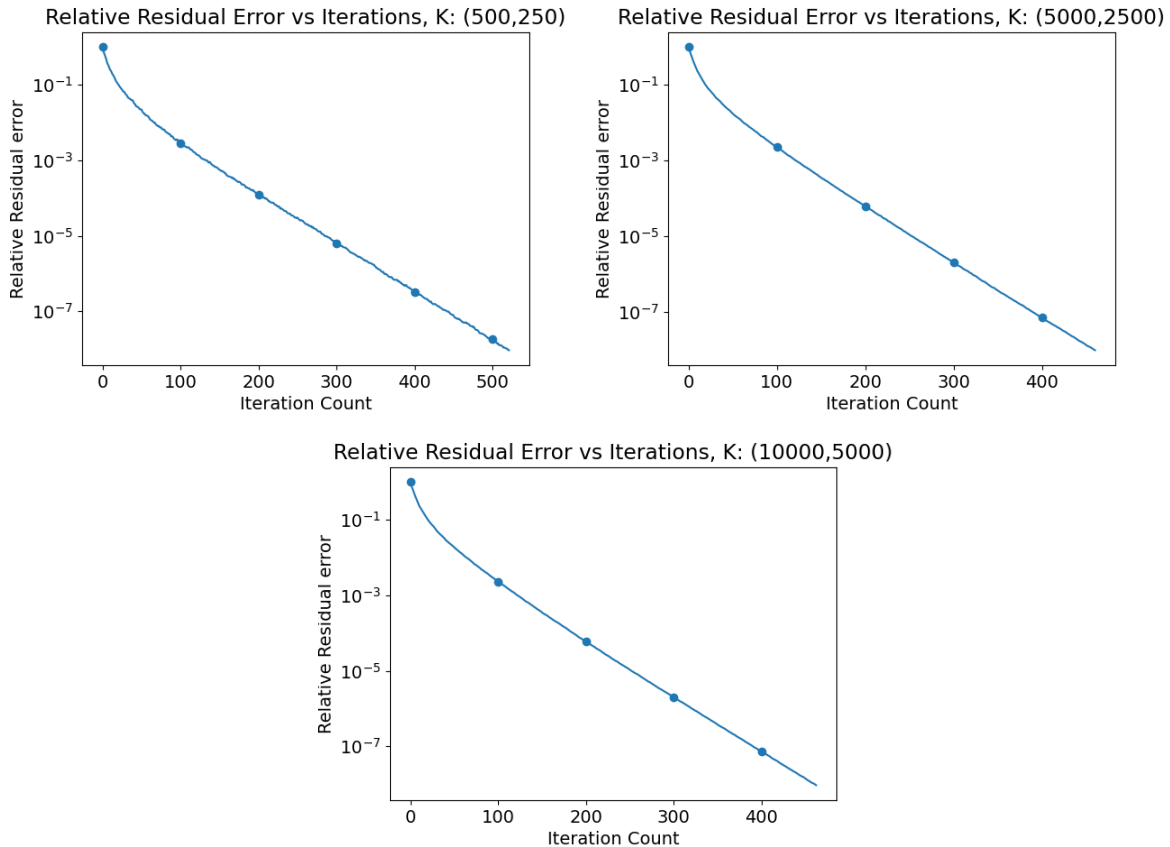


Figure 5.1: Each of the 3 matrices was generated with iid  $\mathcal{N}(0, 1)$  entries, as well as the response vector  $h$  and then each row of the matrix is standardized.

## Analysis

An observation that we see is that DRBK reaches the same error, even as the size of the matrix grows larger. It seems to be the case that DRBK seems to follow the same convergence rate as well. This may be a result of each of the matrices being generated in a similar fashion. Further work may warrant testing this out for different types of matrices as well. Nevertheless, we see that our algorithm does perform considerably well in terms of minimizing the relative residual error.

# Chapter 6

## Tuning Experiments

Oftentimes, an algorithm will have a set of input parameters that have varying purposes. Some of the input parameters serve as ways to provide data for the problem at hand, while others can serve to affect the performance of the algorithm on the problem. In general, “performance” refers to the runtime of the application, but any quantitative value can be used (e.g., minimizing memory usage, minimizing words moved from fast memory to slow memory). These latter parameters are called tuning parameters, and choosing the correct values for these tuning parameters that result in optimal performance is the subject of considerable research. One can mathematically formulate this problem as follows. Let  $\mathbf{t}$  be a vector of tuning parameters, and let  $\mathbf{T}$  represent the space of all possible tuning parameter configurations. Furthermore, let  $f(x; t)$  represent the performance metric of the algorithm with the tuning parameters  $t$  and the problem data  $x$ . Then the problem reduces to

$$\text{For a specific } x, \text{ find } t^* \text{ such that } t^* = \min_{t \in \mathbf{T}} f(x; t)$$

How does one find the optimal tuning parameters for a particular algorithm for optimal performance? In practice, the naive way is to try all possible configurations of the tuning parameter space; however, that becomes harder to do when the number of tuning parameters grows large. Furthermore, the choice of computer architecture on which the program is being executed can also massively impact performance. Considerable research has been done on the process of “tuning parameter selection,” and how one can find the optimal set of tuning parameters for a particular hardware in a reasonable amount of time. There are many sophisticated “auto-tuners”<sup>1</sup> that have been developed and used in practice.

### 6.1 Using Auto-tuners

We will discuss six auto-tuners in particular: hybridM [16], skoptDummy [21], skoptForest [22], skoptGP [23], SMAC [15], TPE [2]. skoptDummy performs a random search within

---

<sup>1</sup>An auto-tuner is software that attempts to find the “optimal” tuning parameter configuration using an algorithm.

the bounds of the parameter values through uniform sampling. SkoptForest, skoptGP, TPE, SMAC, and hybridM attempt to model the function  $f(x; t)$  by a simpler model function. We call these surrogate models for  $f$ . The choice of surrogate models is a topic of research and oftentimes is what differentiates one auto-tuner from another. The next sample is chosen via maximizing an acquisition function, which is typically an inexpensive function and much cheaper than  $f(x; t)$  [10]. In our case, the acquisition function we choose is the expected improvement function. This can be mathematically written as  $\mathbb{E}[u(x; t)] = \mathbb{E}_t[\max(0, f_{min} - f(x; t))]$  where  $f_{min}$  is the minimum value found by  $f$  so far [10]. From here, the smaller  $f(x; t)$  is than  $f_{min}$ , the larger the reward, which is what we want. After finding the point that maximizes this acquisition function, the surrogate model is updated accordingly.

In the Doubly Randomized Block Kaczmarz algorithm, there are many tuning parameters in the algorithm. Such tuning parameters include the block size, the choice of sketching operator, and the oversampling factor (which decides the ambient dimension  $d$  of the sketch). Thus, it would be prudent to use these tuners to see if such a configuration of tuning parameters exists for a particular problem such that the runtime of our algorithm is minimized. If so, we may hope to gather more insight into the particular values or ranges that the tuning parameters take on. Thus, we will run an experiment that will consist of trying out a multitude of different auto-tuners and comparing results. First, a setup of the experiment and a short description of how the auto-tuners work will be introduced. Then we will go into the results of the tuning experiment and analyze the results of our experiment.

## 6.2 Numerical Experiments

### Tuning Variables

- $K \in \mathbb{R}^{m,n}, h \in \mathbb{R}^m, z^0 \in \mathbb{R}^m$
- $\epsilon$  (termination criteria for DRBK<sup>2</sup>),  $r$  (number of iterations of iterative refinement),  $\epsilon_2$  (termination criteria of iterative refinement),  $\epsilon_3$  (termination criteria regarding the norm of the solution to the underdetermined system) and epochs.
- Block size  $b$ , sampling factor  $v$ , sketch type  $S$ .

As we can see,  $K, h, z^0$  are all parameters that define the input problem for our algorithm. Furthermore, the second set of parameters dictates how long the algorithm can run for at most and the accuracy desired by the solution. We get to the third set of parameters, which are our tuning parameters for this experiment. These parameters are interesting because for one we know that the choice of the block size can strongly impact the performance of Kaczmarz methods and thus it would be interesting to see how this changes when using a randomized solver within DRBK. Furthermore, the sampling factor and sketch type are

---

<sup>2</sup>DRBK refers to Doubly Randomized Block Kaczmarz method as referred to in Chapter 5.

quite interesting, because these are tuning parameters that are used within the solver used for the augmented system.

## Results

### Experiment Setup

We plot the cumulative minimum per tuner over 100 samples (chosen according to the tuner) averaged over 5 repeats. Each sample corresponds to a different tuning configuration, and all 5 repeats per sample use the same tuning configuration. The repetitions are done to account for the randomness in the tuning process. By doing so, we hope to see how well each tuner does in finding a tuning configuration that results in optimal runtime. Let  $f(x; t)$  be the runtime of Doubly Randomized Block Kaczmarz with the tuning configuration  $t$  on problem instance  $x$ . For a single tuner  $s$ , we have:  $\{(X_1, y_1), (X_2, y_2), (X_3, y_3), (X_4, y_4), (X_5, y_5)\}$  where  $X_i$  is a list containing 100 samples on the  $i^{\text{th}}$  repeat and  $y_i$  is a list such that  $(y_i)_j = f(x; (X_i)_j)$ . This will be our setup for Figure 6.1 and Table 6.1. In Figure 6.1, for a single tuner  $s$ , we compute  $y_{i,best} \in \mathbb{R}^{100}$  such that  $(y_{i,best})_j = \min_{1 \leq k \leq j} (y_i)_k$ . This can be seen as computing a “running-minimum” of the runtimes in the  $i^{\text{th}}$  repeat. We then plot the point-wise average:  $\overline{y_{best}} = \frac{1}{5} \sum_{j=1}^5 y_{j,best}$ . We compute this for every tuner  $s$ . For our experiment, the values of the non-tuning parameters are listed below:

- $K \in \mathbb{R}^{5000,1000}$ ,  $K_{i,j} \sim \mathcal{N}(0, 1)$ ,  $h \in \mathbb{R}^{5000}$ ,  $h_i \sim \mathcal{N}(0, 1)$ ,  $z^0 = 0^{1000}$
- $\epsilon = 10^{-8}$ ,  $r = 5$ ,  $\epsilon_2 = 10^{-14}$ ,  $\epsilon_3 = 10^{-14}$ , epochs=10

Furthermore, the range of possible values that the auto-tuners will consider for the tuning parameters will be listed below

- Block Size: [100, 300]
- Sampling Factor: (1,5)
- Sketch Type: Orthonormal Sketching Operator, Gaussian Sketching Operator, Sparse Sketching Operator based on SJLT, SRTT Sketching Operator (DCT-II)

In Table 6.1, for a single tuner, we calculate  $y_{min,i} = \operatorname{argmin}_{1 \leq k \leq 100} (y_i)_k$ . We calculate this for  $i = 1, 2, 3, 4, 5$ . We then calculate  $\widetilde{y_{min}} = \operatorname{argmedian}_{1 \leq i \leq 5} \{y_{min,i}\}$ .<sup>3</sup>  $\gamma = \widetilde{y_{min}}$  can be seen as the index of the batch corresponding to the median of the minimum runtimes out of the 5 batches. We then calculate  $X^* = (X_\gamma)_{y_{min,\gamma}}$ , which is the corresponding tuning configuration. In Table 6.1, we show  $X^*$  as well as the runtime for that particular tuning configuration:  $f(x; X^*)$  where  $f(x; t)$  is the runtime of DRBK on tuning configuration  $t$  on problem  $x$ . We also show the total time it took for each tuner to converge to  $X^*$  within

<sup>3</sup>The median is chosen by sorting all 5 elements in ascending order and picking the third element .

batch  $\gamma$  and how many samples the tuner went through. The reason we do this is to get an idea of what tuning parameters each tuner picked so as to minimize the runtime.

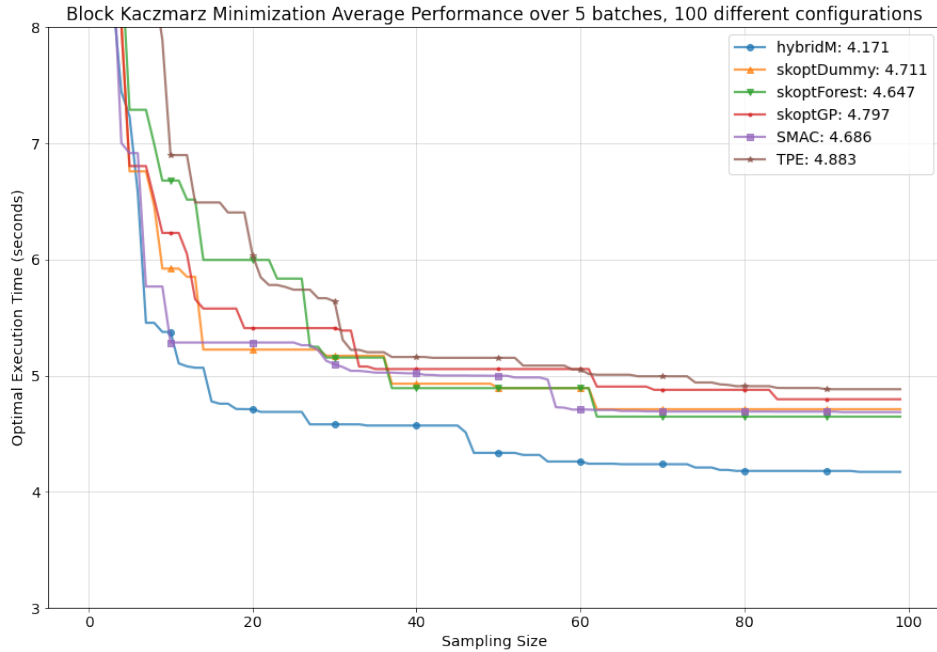


Figure 6.1: A total of 6 auto-tuners were tested. A total of 100 samples were chosen for each auto-tuner repeated 5 times. Here, the cumulative minimum taken for each of the 5 repeats, and then we plot the point-wise mean to get a single curve.

Model	Block Size	Sketch	Sampling Factor	Attained Minimum Runtime (s)	Total Time to Converge (s)	Samples
hybridM	212	SRTT	5	4.001	153.867	12
skoptDummy	253	SRTT	4.894	4.619	77.579	6
skoptForest	253	SRTT	4.894	4.566	77.485	6
skoptGP	240	SRTT	5	4.296	1057.454	70
SMAC	250	SRTT	4.15	4.466	669.020	63
TPE	257	SRTT	4.89	4.615	1003.011	68

Table 6.1: The tuning configuration as well as the attained minimum runtime over 5 batches per auto-tuner.

## Analysis

From Figure 6.1, we can see that the hybridM model finds the tuning configuration resulting in the smallest minimum runtime. Furthermore, from what we can see, it seems that by around sample 70, all of the auto-tuners besides skoptGP, TPE, and hybridM stop finding better tuning configurations. This may indicate that, for larger matrices, one might want to limit the sampling size to 60-80 to take less time. However, we should take these results with a grain of salt as the runtimes reported may differ on a different computer architecture.

Furthermore, from analyzing Table 6.1, we can see that the SRTT based on DCT II seems to be the best sketch operator for all 6 tuners. This may suggest that for future problems we can reasonably use SRTT. However, more research should be done into whether this holds for larger problems, such as matrices with dimensions  $(2 * 10^5, 1 * 10^5)$ . Furthermore, another observation is that while skoptGP has the second minimum runtime, it took the most time (1057.454 seconds) to find that value. This is relevant as one might prefer an auto-tuner that finds their minimum runtime in a quicker amount of time.



# Chapter 7

## Related Work

There have been many variants and extensions of Block Kaczmarz as well as a simple Kaczmarz method, that vary in application and usage. For instance, one can incorporate randomness into choosing the blocks, as we covered in Chapter 1. Here we will go over extensions of Kaczmarz. By doing so, we can hope to analyze how well Kaczmarz performs in practice and tricks that people have used to make it work for them.

### 7.1 Clustered Kaczmarz Method

A variant of Kaczmarz that was used was introduced in “Accelerating Random Kaczmarz Algorithm Based on Clustering Information.” This paper improves upon “Acceleration of Randomized Kaczmarz Method via the Johnson-Lindenstrauss Lemma” [14] [9] in that it uses the algorithm mentioned as a benchmark. The new algorithm is as follows: rather than looking through all the rows  $K_i$  of  $K$  or  $f(K_i)$  where  $f$  is the projection operator as noted in [9] and selecting the one that maximizes  $\|z^{k+1} - z^k\|_2$ , the rows of the matrix  $K$  are instead clustered into  $p$  clusters via K-means clustering. Then the algorithm chooses a representative vector  $K_{c_l}$  for each cluster  $c_l$  and proceed. The intuition is that since the goal is to maximize the distance  $\|z^{k+1} - z^k\|_2$ , we want to consider hyperplanes that are furthest in distance to our current iterate, which likely lie in the cluster furthest from the current iterate rather than looking at other clusters. Other than that, the algorithm proceeds in the same way as RBK-JL. This paper also improves on [9] in that it extends this for Block Kaczmarz method. Here, the blocks are chosen by randomly picking one row from each cluster  $c$  and aggregating it into one block  $K_{\tau_i}$  and repeating the process for however many blocks are needed, which is up to the user. The corresponding values in  $h$  are given by  $h_{\tau_i}$ . After this is done, the Randomized Block Kaczmarz is run with this partition of blocks. As we have seen in Chapter 1, the convergence rate of Randomized Block Kaczmarz is highly dependent on the row paving  $\frac{\beta}{\alpha}$  which essentially acts as a “condition number” for the blocks. The smaller this number, the faster the convergence. The reason why this is important is that the paper provides a proof which states that the constructing the blocks via extracting rows

from each cluster results in each block being well-conditioned with a small spectral norm [14]. However, the theoretical analysis conducted in the paper is done under the assumption that the data are somewhat “Gaussian.” Numerical experiments show that this modified Kaczmarz with clustering outperforms RBK-JL while also being more robust to noise that is added to  $K$ .

## 7.2 Applications of Block Kaczmarz

As we analyze the randomized Block Kaczmarz method and the practicality that it may prove, it is important to look into current applications of Block Kaczmarz method and how people are using the method for their particular project or study. The reason why this is important is because solving tall-and-skinny systems of equations comes up in many fields of study, ranging from physics to chemistry to machine learning to computational statistics. This will prove to be important because by looking into the practical applications of Block Kaczmarz to solve tall systems of equations, it may help us recognize some inherent advantages of Block Kaczmarz and whether or not it can be used to solve big systems in a practical time period. We might be able to gather more insight into different row sampling schemes using the Block Kaczmarz method.

For example, Block Kaczmarz methods can be used to help solve the matrix low-rank factorization problem. This is shown in “On Application of Block Kaczmarz Methods in Matrix Factorization” [8]. This problem arises quite often in many machine learning applications such as recommendation systems, collaborative filtering, and topic modeling. The low-rank matrix factorization problem is defined below.

$$\text{Given a data matrix } X \in \mathbb{R}^{m,n}, \text{ find } A, S = \underset{A,S}{\operatorname{argmin}} \|X - AS\|_F^2$$

where  $A \in \mathbb{R}^{m,k}$  and  $S \in \mathbb{R}^{k,n}$ . An explicit solution to the problem can be found using the SVD of  $X$ . A common approach to computationally solving matrix factorization problems is to utilize some notion of an “alternating scheme” where different least squares problems are solved each iteration. Because we have two matrices under which we are optimizing over, this works by “updating” one matrix while holding the other one constant and then alternating with the other matrix. However, solving a system of equations each time becomes more costly as the size of the matrix increases. Furthermore, oftentimes in matrix factorization problems, there is some level of sparsity that can be exploited to achieve faster convergence rates.

How the alternating scheme works with Randomized Block Kaczmarz is that it samples a column index  $i \in [n]$ . Then it solves  $As = X_{:,i}$  for  $s \in \mathbb{R}^k$  using Randomized Block Kaczmarz. Then the  $i^{\text{th}}$  column of  $S$  is replaced by  $s$ . Then, we sample a row index  $j \in [m]$  and then solve  $S^T a = X_{j,:}^T$  using Randomized Block Kaczmarz. Then the  $j^{\text{th}}$  row of  $A$  is replaced by  $a$ . This is repeated for how many iterations need be. Initially, the original algorithm attempts to directly solve for  $a, s$ . In [8], multiple experiments were run, comparing using alternating

least squares with randomized Block Kaczmarz to the current method of alternating least squares and the results showed that for big matrices where the true rank  $k$  is known,  $X$ , randomized Block Kaczmarz method outperforms the previous alternating least squares in wall clock time, however, requiring more iterations to achieve the same relative error. This relative error is defined as:

$$\frac{\|X - AS\|_F^2}{\|X\|_F^2}$$

Another key observation that was made was that although Randomized Block Kaczmarz required more iterations to converge to a solution, it required much less memory since only a block needed to be loaded into memory each iteration. This is interesting because it presents the question of whether Block Kaczmarz methods can be used in scenarios when the data matrix is too big to fit into memory.

Another scenario where we see a similar variant of Kaczmarz methods being used in practice is in tomography. The method used is called ‘‘Algebraic Reconstruction Technique’’ which is an iterative solver to solve  $Kz = h$ . The algorithm is defined as

$$z^{k+1} = z^k + \lambda_k \frac{h_t - K_t z^k}{\|K_t\|_2^2} K_t^T, k \geq 0$$

where  $K_t$  is the  $t^{\text{th}}$  row of  $K$ ,  $h_t$  is the  $t^{\text{th}}$  value of  $h$ , and  $z^k$  is the answer found at iteration  $k$ . As we can see, this is the same as Kaczmarz method other than the  $\lambda_k$  parameter, which is used as an ‘‘relaxation parameter’’ to control the rate of convergence. This is used in the medical field for ‘‘image reconstruction.’’

These practical applications of Block Kaczmarz give us insight into how we might be better able to analyze certain tuning parameters, such as the block size. For instance, in [8], it was shown that a larger block size resulted in a trade-off of faster convergence with increasing runtime. From its usage in tomography, we can look into this  $\lambda_k$  relaxation parameter as a potential tuning parameter in the future. By tuning the Randomized Block Kaczmarz and looking into some of the tuning parameters, we hope to see more usage of Block Kaczmarz in practice.

# Chapter 8

## Conclusion

As we can see, the beauty of Kaczmarz lies in its simplicity. From the simple algorithm of projecting onto a solution space of a block of equations, one can derive the block Kaczmarz algorithm and then derive the randomized block Kaczmarz algorithm and then more variants. Furthermore, by only having to solve an under-determined linear system of equations per iteration, one simply needs to load in one block of the matrix into memory at a time, which may be advantageous in the case where the entire matrix is too big to fit into memory. In this thesis, we went over Kaczmarz methods and the basics of randomized linear algebra to understand the motivation behind our new algorithm, Doubly Randomized Block Kaczmarz method. This primarily focused on incorporating a randomized solver to solve the under-determined linear system that needed to be solved per iteration. Furthermore, to improve the accuracy of our solution found by the randomized solver, we went over using Iterative Refinement, a common technique for improving the solution to a linear system of equations. Afterwards, we ran a tuning experiment to see how practically effective our Doubly Randomized Block Kaczmarz method can be.

Through this thesis, we aimed to evaluate how efficient and practical Kaczmarz methods can be in finding a solution for an over-determined linear system of equations. Future work may focus on choosing a different termination criteria within our Doubly Randomized Block Kaczmarz algorithm and running it through a bunch of auto-tuners and seeing if the tuning parameters change. We hope that our work serves as a reference in the development of randomized algorithms for solving linear systems of equations through Kaczmarz.

# Bibliography

- [1] BALLISTIC. “Prospectus for a Randomized BLAS and LAPACK”. Preliminary version, not for circulation. Nov. 2021.
- [2] James Bergstra, Dan Yamins, and David D. Cox. “Making a Science of Model Search”. In: *CoRR* abs/1209.5111 (2012). arXiv: 1209.5111. URL: <http://arxiv.org/abs/1209.5111>.
- [3] Ake Bjorck. “Iterative Refinement”. In: *Numerical Methods for Least Squares*. SIAM, 1996, pp. 120–124.
- [4] Ake Bjorck. “Iterative Refinement of Linear Least Squares Solution I”. In: *BIT* 7 (1967), pp. 257–258.
- [5] Ake Bjorck. “Iterative Refinement of Linear Least Squares Solution II”. In: *BIT* 8 (1968), pp. 8–30.
- [6] Michael Cohen, T S Jayram, and Jelani Nelson. *Simple analyses of the sparse Johnson-Lindenstrauss transform*. Jan. 2018. URL: <https://drops.dagstuhl.de/opus/volltexte/2018/8305/>.
- [7] James W. Demmel. *Applied Numerical Linear Algebra*. Siam, 1997.
- [8] Jamie Haddock Edwin Chau. “ON APPLICATION OF BLOCK KACZMARZ METHODS IN MATRIX FACTORIZATION”. In: *arXiv preprint arXiv:2010.10635* (2020). arXiv: 2010.10635.
- [9] Yonina C. Eldar and Deanna Needell. *Acceleration of Randomized Kaczmarz Method via the Johnson-Lindenstrauss Lemma*. 2010. DOI: 10.48550/ARXIV.1008.4397. URL: <https://arxiv.org/abs/1008.4397>.
- [10] Roman Garnett. *Bayesian optimization*. 2015. URL: [https://www.cse.wustl.edu/~garnett/cse515t/spring\\_2015/files/lecture\\_notes/12.pdf](https://www.cse.wustl.edu/~garnett/cse515t/spring_2015/files/lecture_notes/12.pdf).
- [11] R.V.L. Hartley. “A More Symmetrical Fourier Analysis Applied to Transmission Problems”. In: *Proceedings of the IRE* 30.3 (1942), pp. 144–150. DOI: 10.1109/JRPROC.1942.234333.
- [12] *Intel Math Kernel Library. Reference Manual*. Santa Clara, USA. ISBN 630813-054US. Intel Corporation, 2009.

- [13] Stefan Kaczmarz. “Angenäherte Auflösung von Systemen linearer Gleichungen”. In: *Bulletin International de l’Académie Polonaise des Sciences et des Lettres. Classe des Sciences Mathématiques et Naturelles. Série A, Sciences Mathématiques* 35 (1937), pp. 355–357.
- [14] Yujun Li, Kaichun Mo, and Haishan Ye. *Accelerating Random Kaczmarz Algorithm Based on Clustering Information*. 2015. DOI: 10.48550/ARXIV.1511.05362. URL: <https://arxiv.org/abs/1511.05362>.
- [15] Marius Lindauer et al. *SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization*. 2021. arXiv: 2109.09831 [cs.LG].
- [16] Hengrui Luo et al. “Hybrid Models for Mixed Variables in Bayesian Optimization”. In: (2022+).
- [17] Per-Gunnar Martinsson and Joel Tropp. *Randomized Numerical Linear Algebra: Foundations and Algorithms*. 2020. DOI: 10.48550/ARXIV.2002.01387. URL: <https://arxiv.org/abs/2002.01387>.
- [18] Riley Murray. *BALLISTICLA/Parla: Python algorithms for randomized linear algebra*. Aug. 2021. URL: <https://github.com/BallisticLA/parla>.
- [19] Deanna Needell and Joel A. Tropp. “Paved with good intentions: Analysis of a randomized block Kaczmarz method”. In: *Linear Algebra and its Applications* 441 (Jan. 2014), pp. 199–221. DOI: 10.1016/j.laa.2012.12.022. URL: <https://doi.org/10.1016%2Fj.laa.2012.12.022>.
- [20] Alan V Oppenheim, Ronald W Schafer, and John R Buck. *Discrete-Time Signal Processing*. Prentice Hall, 1999.
- [21] *Skopt.dummy\_minimize*. URL: [https://scikit-optimize.github.io/stable/modules/generated/skopt.dummy\\_minimize.html](https://scikit-optimize.github.io/stable/modules/generated/skopt.dummy_minimize.html).
- [22] *Skopt.forest\_minimize*. URL: [https://scikit-optimize.github.io/stable/modules/generated/skopt.forest\\_minimize.html](https://scikit-optimize.github.io/stable/modules/generated/skopt.forest_minimize.html).
- [23] *Skopt.gp\_minimize*. URL: [https://scikit-optimize.github.io/stable/modules/generated/skopt.gp\\_minimize.html](https://scikit-optimize.github.io/stable/modules/generated/skopt.gp_minimize.html).
- [24] *The HSL Mathematical Software Library*. URL: <https://www.hsl.rl.ac.uk/>.
- [25] Joel A Tropp. “IMPROVED ANALYSIS OF THE SUBSAMPLED RANDOMIZED HADAMARD TRANSFORM”. In: *Advances in Adaptive Data Analysis* 3.1 (), pp. 115–126.
- [26] Joel A Tropp. *Randomized Algorithms for Matrix Computations*. 2020. URL: <https://authors.library.caltech.edu/108783/1/Tro20-Randomized-Matrix-Computations-LN.pdf>.
- [27] Joel A. Tropp et al. “Streaming Low-Rank Matrix Approximation with an Application to Scientific Simulation”. In: *CoRR* abs/1902.08651 (2019). arXiv: 1902.08651. URL: <http://arxiv.org/abs/1902.08651>.