Row selection in Kaczmarz variant methods

A comparison of existing methods and an introduction of a novel method for solving consistent, over-determined systems of linear equations

Shiv Munagala

Contents

1 Introduction

An extensive and important field within modern mathematics is that of *Numerical Linear Algebra*. It explores the use of numerical methods to solve problems arising from linear algebra, and the ability to solve these problems is of fundamental importance to mathematical scientists [1]. Solving systems of linear equations is a question of particular interest within the field of numerical linear algebra, as many real-world problems reduce to solving of linear systems. It is a problem that arises in the contexts of; approximating partial differential equations [2], signal processing [3], [4], computed tomography (CT) scans [5], and machine learning [6], to name only a few applications.

Many of these problems result in very large systems which are difficult, or computationally unreasonable, to solve directly. Some methods to solve linear systems that the reader may be familiar with include: Gaussian elimination, LU decomposition, or QR decomposition. Unfortunately, these methods are not practical on a large scale, as they require order $O(n^3)$ operations for an $n \times n$ matrix [2]. This motivates the need for more efficient numerical methods. Note that we primarily consider *consistent* systems of equations, but we show through experiments that these methods also seem to work for inconsistent systems.

The systems of linear equations that we consider are *over-determined*. In other words, systems of linear equations where we have more equations than we do unknown variables. One of the many contexts which give rise to over-determined systems of linear equations is CT scans [5]. The development of CT scans was awarded the 1979 Nobel Prize in Medicine because of its vital importance in modern medicine [7].

The method implemented in the first medical scanners in 1970 is known as the *Algebraic Reconstruction Technique (ART)* [8], [9]. As it happens, the Algebraic Reconstruction Technique is a rediscovery of an existing method, first introduced by Stefan Kaczmarz in 1937 [10], aptly referred to as the Kaczmarz method in the context of Numerical Linear Algebra [8].

We detail the Kaczmarz method in Section 2.1. While the Kaczmarz method seemingly performs well, we don't have a proof of how it converges on general matrices. This is a problem addressed by Strohmer and Vershynin in their 2009 paper, *"A randomized Kaczmarz algorithm with exponential convergence"* [8]. They propose a variation on the Kaczmarz method, referred to as the *Randomized Kaczmarz method*, which they prove converges with expected exponential rate. Moreover, they show the rate of convergence can't be improved by more than a constant factor. This paper spurred much interest in the literature around the Kaczmarz method [11]–[15]. We discuss this method among other variations in Section 2.

While the variations of the Kaczmarz method have interesting and useful properties that we can prove about them, when performing computational experiments, they often don't perform as well as the standard Kaczmarz method from 1937. To that end, we sought a randomized method that we could prove converges with an expected exponential rate, but computationally is similar to the standard Kaczmarz method. In Section 3, we reveal this method, discuss how it compares computationally from a high-level perspective, and give a proof of expected exponential convergence.

We run numerical experiments using MATLAB in Section 4. The experiments are run on both randomly generated data as well as real-world data. The special case of having an orthogonal matrix is also pointed out in Section 4.3, which gives an intuitive background for the method we introduced in Section 3. Moreover, Section 4.3 serves as a good jumping board for us to compare the methods against each other in Section 5. We conclude the report in Section 6, in which we discuss what has been achieved, what has not, and potential directions for future investigation.

To aid the reader in keeping track of the various methods, we provide a summary of how each method works in Table 2, Appendix A. Material not directly relevant to the discussion in the body of the

report, but which we think is worth mentioning regardless, is given in the appendix and referenced in the body where relevant. The code which we used to run the experiments in Section 4 is provided in Appendix E, this code is heavily commented, which we hope will make it easily reusable.

2 The Kaczmarz Method and its variants

Consider the linear system

$$A\boldsymbol{x} = \boldsymbol{b}, \tag{1}$$

where $A \in \mathbb{R}^{m\times n}$ is a matrix of full column rank (i.e. $\text{rank}(A) = n$), $\boldsymbol{x} \in \mathbb{R}^n$ is an unknown vector which we seek to find, and $\boldsymbol{b} \in \mathbb{R}^m$. Let this be an over-determined system, that is, $m > n$. Then, given A and $\boldsymbol{b}$, we are interested in numerically solving for $\boldsymbol{x}$. We assume that the linear system is consistent, so an exact solution exists. The Kaczmarz method, introduced by Kaczmarz in [10], is a way to numerically solve for $\boldsymbol{x}$.

Let the rows of A be denoted by $\boldsymbol{a}_1^T, \ldots, \boldsymbol{a}_m^T$, and let b_i denote the i-th entry of $\boldsymbol{b}$. Denote the k-th iteration of our numerical solution for $\boldsymbol{x}$ by $\boldsymbol{x}_k$.

2.1 The Kaczmarz Method

In the Kaczmarz method we start with some initial guess $\boldsymbol{x}_0$. Given iteration $\boldsymbol{x}_k$, we orthogonally project it onto the hyperplane $\langle \boldsymbol{a}_i, \boldsymbol{x}\rangle = b_i$ to obtain $\boldsymbol{x}_{k+1}$. The rows of A and entries of $\boldsymbol{b}$ are selected sequentially. By repeatedly performing this orthogonal projection, we converge to $\boldsymbol{x}$ [8], [10]. This can be written as

$$\boldsymbol{x}_{k+1} := \boldsymbol{x}_k + \frac{b_i - \boldsymbol{a}_i^T\boldsymbol{x}_k}{\|\boldsymbol{a}_i\|_2^2}\boldsymbol{a}_i. \tag{2}$$

Here, we use $\|\cdot\|_2$ to denote the l^2-norm. The algorithm for the Kaczmarz method, written in pseudocode for the first $K+1$ iterations, is given in Algorithm 1.

Algorithm 1 The Kaczmarz method

let $\boldsymbol{x}_0$ be some initial approximation
for $k := 0$ **to** K **do**
  $i := (k \bmod m) + 1$
  $\boldsymbol{x}_{k+1} := \boldsymbol{x}_k + \frac{b_i - \boldsymbol{a}_i^T\boldsymbol{x}_k}{\|\boldsymbol{a}_i\|_2^2}\boldsymbol{a}_i$
end for

This procedure is shown graphically in Figure 1 for $n = 2$ dimensions and $m = 3$ equations. The solution hyperplanes in this context correspond to straight lines. We cycle through the rows sequentially, once we reach row $m = 3$, we go back to row 1. As we perform more iterations, we approach the exact solution, $\boldsymbol{x}$.

From this graphical example, it is easy to imagine examples where going through the rows of A sequentially can be far from ideal. Consider the same system as in our graphical example but with more solution hyperplanes, many of which have roughly the same gradient. Then, each iteration leads to only a small change in $\boldsymbol{x}_k$. Depending on the real-world problem we are attempting to solve, having sequential rows being similar is a reasonable concern. A natural way to tackle this is to select the rows of A in some randomized fashion [16]. In [8], Strohmer and Vershynin prove that a randomized version of the Kaczmarz method converges in expectation with exponential rate.

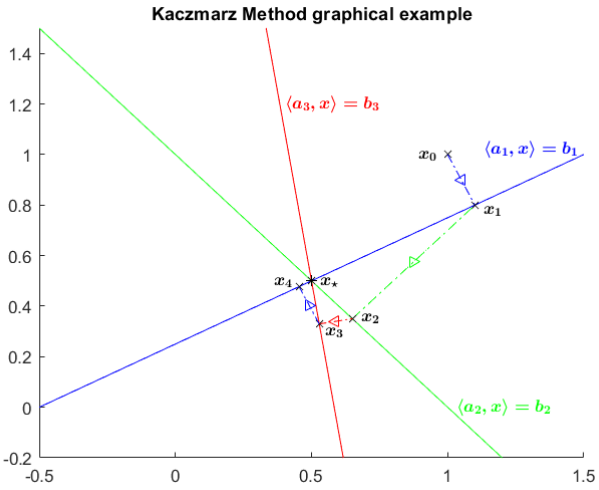


Figure 1: A graphical example of the Kaczmarz method for $m = 3, n = 2$ with $K = 4$ iterations. We use $x_\star$ to denote the exact solution.

2.2 The Randomized Kaczmarz Method

In [8], Strohmer and Vershynin propose selecting the rows of A randomly, with probabilities proportional to the square of the row-norms, $\|\boldsymbol{a}_i\|_2^2$. They prove that this method, henceforth called the Randomized Kaczmarz method, demonstrates expected exponential convergence. Moreover, they give a bound on the rate of convergence; for iteration number k and exact solution $\boldsymbol{x}$ they show:

$$\mathbb{E}\|\boldsymbol{x}_k - \boldsymbol{x}\|_2^2 \leq \left(1 - \kappa(A)^{-2}\right)^k \cdot \|\boldsymbol{x}_0 - \boldsymbol{x}\|_2^2, \tag{3}$$

where $\kappa(A)$ is the *scaled condition number*. The definition of the scaled condition number is somewhat technical and beyond the scope of this report; it plays no notable role for the remainder of the report. We only note that the scaled condition number is a function of the matrix A. Motivated readers may refer to Appendix B to find the definition of the scaled condition number. Moreover, Strohmer and Vershynin also show that their estimate cannot be improved beyond a constant factor, that is, we can only improve the method by reducing the rate of convergence, $1 - \kappa(A)^{-1}$.

In Algorithm 2 we give the algorithm for the Randomized Kaczmarz method for the first K iterations in pseudocode.

Algorithm 2 The Randomized Kaczmarz method

let $\boldsymbol{x}_0$ be some initial approximation
compute $\|\boldsymbol{a}_1\|_2^2, \ldots, \|\boldsymbol{a}_m\|_2^2$
for $k := 0$ **to** K **do**
  choose i from $\{1, \ldots, m\}$ randomly with probability proportional to $\|\boldsymbol{a}_1\|_2^2, \ldots, \|\boldsymbol{a}_m\|_2^2$
  $\boldsymbol{x}_{k+1} := \boldsymbol{x}_k + \frac{b_i - \boldsymbol{a}_i^T \boldsymbol{x}_k}{\|\boldsymbol{a}_i\|_2^2} \boldsymbol{a}_i$
end for

Strohmer and Vershynin also introduce the *Simple Randomized Kaczmarz method* in [8]. In the Simple Randomized Kaczmarz method, we sample the rows of A uniformly at random, with replacement. This method also demonstrates exponential convergence, as is implied by the results in [17]. The algorithm for the Simple Randomized Kaczmarz method for the first $K + 1$ iterations is given in pseudocode in Algorithm 3. The performance of the Simple Randomized Kaczmarz method is

compared to the Randomized Kaczmarz method, among other methods, in Section 4. We make note of the Simple Randomized Kaczmarz method as our novel method in Section 3 is based on it.

Algorithm 3 The Simple Randomized Kaczmarz method

let $\boldsymbol{x}_0$ be some initial approximation
for $k := 0$ **to** K **do**
  choose i from $\{1, \dots, m\}$ randomly with uniform probability
  $\boldsymbol{x}_{k+1} := \boldsymbol{x}_k + \frac{b_i - \boldsymbol{a}_i^T \boldsymbol{x}_k}{\|\boldsymbol{a}_i\|_2^2} \boldsymbol{a}_i$
end for

Comparing the Randomized Kaczmarz method in Algorithm 2 with the standard Kaczmarz method in Algorithm 1, it is clear to see that the Randomized Kaczmarz method has a larger computational overhead from having to compute row-norms before we start the iterations. There is also more computation required per iteration from having to randomly select rows, which is more intensive than modulo calculations.[1] We show in Section 4 that for many matrices this added computation means that the Randomized Kaczmarz method is inferior to the standard Kaczmarz method when considering the computational cost needed to attain a given error.

In [8] it was shown that for some systems of equations the Randomized Kaczmarz method outperforms the standard Kaczmarz method in terms of the number of iterations needed to reach a certain error. In [18], Censor et al. point out that the Randomized Kaczmarz method is not superior to the standard Kaczmarz method in general, a point corroborated by our numerical experiments in Section 4. In [19], Strohmer and Vershynin clarify that assigning probabilities proportional to the row-norms is *not* optimal and that the significant result from their paper [8] is showing the expected exponential convergence of a Kaczmarz variant method. In Section 4 we show the Randomized Kaczmarz method doesn't outperform other (newer) randomized Kaczmarz variant methods.

From experiments in Section 4, we have reason to believe that the standard Kaczmarz method outperforms the Randomized Kaczmarz method on most matrices when considering the number of iterations and the amount of computation needed. The standard Kaczmarz method, however, has no results to prove exponential convergence, unlike the Randomized Kaczmarz method. This leads us to ask; are there methods that we can prove converge exponentially in expectation, but also outperform the Randomized Kaczmarz method computationally? In Section 2.3 we discuss the Sampling Kaczmarz-Motzkin method, introduced by De Loera et al. in [11], which does precisely this. Then, in Section 3 we provide a novel method that, experimentally, outperforms the Sampling Kaczmarz-Motzkin method.

2.3 The Sampling Kaczmarz-Motzkin Method

In [11], De Loera et al. combine the Simple Randomized Kaczmarz method with the Motzkin method to produce the *Sampling Kaczmarz-Motzkin method*. The details of the Motzkin method, also known as *Greedy Kaczmarz* [20], are not relevant to this report but are given in [11] and [21], should the reader wish to look into it. De Loera et al. show that the Sampling Kaczmarz-Motzkin method also demonstrates expected exponential convergence. From computational experiments in [11] and Section 4 we see that the Sampling Kaczmarz-Motzkin method outperforms the Randomized Kaczmarz method for most matrices when considering both the number of iterations required and the computational cost. Note that we consider the Sampling Kaczmarz-Motzkin method with no *relaxation*[2] for a fair comparison between the methods, this decision is expanded on in Appendix C.

[1] This is later demonstrated in Figure 3.

[2] Relaxation: Multiplying the projection term $\frac{b_i - \boldsymbol{a}_i^T \boldsymbol{x}_k}{\|\boldsymbol{a}_i\|_2^2} \boldsymbol{a}_i$ from Equation (2) by some scalar λ. This can improve the rate of convergence; see Appendix C.

Each iteration of the Sampling Kaczmarz-Motzkin method starts by choosing a subset of the rows of A uniformly at random. The subset should be of size β, an integer which we pick. Then, we choose the row i from that subset which gives the maximum residual, $b_i - \boldsymbol{a}_i^T \boldsymbol{x}_k$. We provide the pseudocode for the Sampling Kaczmarz-Motzkin method for the first $K+1$ iterations in Algorithm 4.

Algorithm 4 The Sampling Kaczmarz-Motzkin method

let $\boldsymbol{x}_0$ be some initial approximation
for $k := 0$ **to** K **do**
  choose a subset τ_{k+1} of size β from $\{1, \dots, m\}$ uniformly at random
  choose i_{k+1} from the β rows such that $i_{k+1} := \arg\max_{i \in \tau_{k+1}} |b_i - \boldsymbol{a}_i^T \boldsymbol{x}_k|$
  $\boldsymbol{x}_{k+1} := \boldsymbol{x}_k + \frac{b_{i_k} - \boldsymbol{a}_{i_k}^T \boldsymbol{x}_k}{\|\boldsymbol{a}_{i_k}\|_2^2} \boldsymbol{a}_{i_k}$
end for

The Sampling Kaczmarz-Motzkin method has no overhead, unlike the Randomized Kaczmarz method. However, at each iteration, we must compute the residual, $|b_i - \boldsymbol{a}_i^T \boldsymbol{x}_k|$, for β rows. This is computationally very intensive. Nonetheless, the improvements from choosing the rows in this way are significant enough such that the Sampling Kaczmarz-Motzkin method still outperforms the Randomized Kaczmarz method in terms of computational cost. In [11], De Loera et al. show that the Sampling Kaczmarz-Motzkin method outperforms the Simple Randomized Kaczmarz method. We replicate this result in Section 4 and show it also outperforms the Randomized Kaczmarz method.

When we compare the Sampling Kaczmarz-Motzkin method to the standard Kaczmarz method, however, the standard Kaczmarz method still seems to outperform the Sampling Kaczmarz-Motzkin method computationally. As with the Randomized Kaczmarz method, the benefit the Sampling Kaczmarz-Motzkin method has over the standard Kaczmarz method is provable exponential convergence.

So we ask, is there a method that is as computationally as efficient as the standard Kaczmarz method, but with provable exponential convergence? This is the problem we tackle in Section 3.

3 The Modified Simple Randomized Kaczmarz Method

In the standard Kaczmarz method, we iterate through the rows of our matrix in order. In the Randomized Kaczmarz method, we chose the rows of our matrix with probability proportional to the square of the l^2-norm of the row. In the Simple Randomized Kaczmarz method, we choose the rows of our matrix uniformly at random.

Expected exponential convergence of the Randomized Kaczmarz method and the Simple Randomized Kaczmarz method are shown in [8] and [17] respectively. In all of the aforementioned Kaczmarz variant methods where we use random sampling, we sample the rows *with* replacement. We introduce the *Modified Simple Randomized Kaczmarz method*, in which we sample the rows uniformly at random *without* replacement. This is given in pseudocode in Algorithm 5 for the first $K+1$ iterations.

We concede that this statement of the algorithm is quite non-specific and there are many ways to implement this, particularly with respect to how we choose rows uniformly at random without replacement. It is, however, easy to understand from the context of the Simple Randomized Kaczmarz method. This is also the natural way to state the algorithm, given the intuition behind it that we provide in 4.3. Furthermore, it is easier to prove the expected exponential convergence when expressed in this way. Nonetheless, we provide a particular implementation using permutations in Algorithm 6 which, in terms of the computational cost, is very efficient.

Algorithm 5 The Modified Simple Randomized Kaczmarz method

let $\boldsymbol{x}_0$ be some initial approximation
for $k := 0$ **to** K **do**
choose i from $\{1, \ldots, m\}$ randomly with uniform probability, without replacement. If all the rows have been chosen, reset to the original set $\{1, \ldots, m\}$.
$\boldsymbol{x}_{k+1} := \boldsymbol{x}_k + \frac{b_{i_k} - \boldsymbol{a}_{i_k}^T \boldsymbol{x}_k}{\|\boldsymbol{a}_{i_k}\|_2^2} \boldsymbol{a}_{i_k}$
end for

With this implementation, we place the code for permutations *before* the code for projections in our `for` loop. In Appendix E.3, we place the code for permutations *after* the code for projections. We do so because we start indexing from 1 in MATLAB. Assuming there is no underlying pattern in the ordering of the rows, these two should perform equivalently in practice.

Algorithm 6 The Modified Simple Randomized Kaczmarz method (permutation implementation)

let $\boldsymbol{x}_0$ be some initial approximation
for $k := 0$ **to** K **do**
if $(k \bmod m) == 0$ **then**
let P be some uniformly random $m \times m$ permutation matrix
$A := PA$
$\boldsymbol{b} := P\boldsymbol{b}$
end if

$i := (k \bmod m) + 1$
$\boldsymbol{x}_{k+1} := \boldsymbol{x}_k + \frac{b_i - \boldsymbol{a}_i^T \boldsymbol{x}_k}{\|\boldsymbol{a}_i\|_2^2} \boldsymbol{a}_i$

end for

Let us define a *cycle* to be a set of iteration such that every row has been sampled once. Notice that one cycle of the Modified Simple Randomized Kaczmarz method is equivalent to the standard Kaczmarz method but with the rows of the matrix A and the rows of $\boldsymbol{b}$ having been permuted. So, one cycle of the Modified Simple Randomized Kaczmarz method can be equivalently implemented by premultiplying our system by some random permutation matrix and applying the standard Kaczmarz method. This is what we are doing in Algorithm 6.

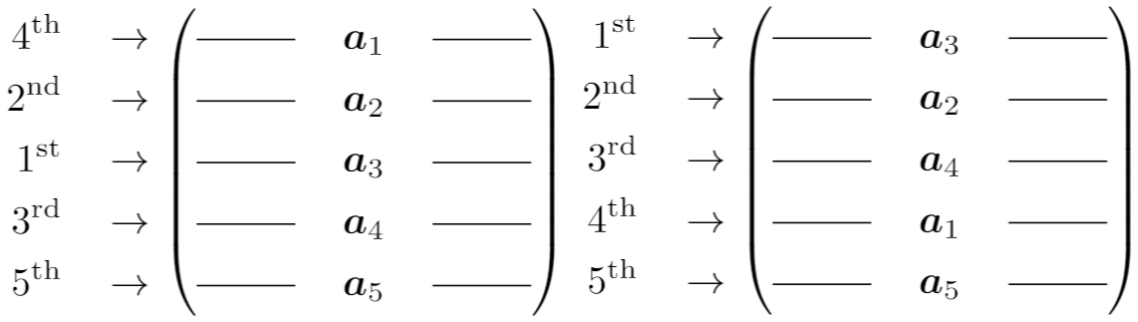
To see the equivalence between these two forms more clearly, we draw the reader's attention to Figure 2. We show an example of a matrix with 5 rows. In Figure 2a we select the rows of A uniformly at random, without replacement, and obtain the order $4, 2, 1, 3, 5$. In the permutation implementation, this would look as follows.

Premultiply the system $A\boldsymbol{x} = \boldsymbol{b}$ by P, where

$$P = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Note that the solution $\boldsymbol{x}$ for this permuted system, $PA\boldsymbol{x} = P\boldsymbol{b}$, is the same.

Consider performing the standard Kaczmarz method for 5 iterations on $PA\boldsymbol{x} = P\boldsymbol{b}$, this is illustrated in Figure 2b. This is equivalent to performing the Modified Simple Randomized Kaczmarz method (permutation implementation) with the specific realisations of random numbers and random matrix mentioned before.



(a) A with random row-selection without replacement (b) Permuted A with sequential row choices

Figure 2: Both of these selection methods results in the same order of rows being selected. For any random row choice, there exists some permutation such that the rows can be chosen in order.

For the proof of expected exponential convergence, we consider the form expressed in Algorithm 5. When implementing the algorithm in code, however, we use the form expressed in Algorithm 6. Permuting the system is, computationally, a very cheap operation. To illustrate this, we use MATLAB Profiler running the Modified Simple Randomized Kaczmarz method on the data from Section 4.1. As can be seen from Figure 3, the most intensive part of the algorithm are projections on `Line 113`, whereas the time for the modulo calculations and permutations (`Line 115-119`) add around 25% of the time `Line 113` does. Compared to the added computation cost from the other methods, this extra 25% is comparatively very small. Part of the efficiency comes from the fact that we are calling the random number generator only once every m iterations, rather than with every single iteration.

```
                 108     %% Modified Simple Randomized Kaczmarz
< 0.001        1 109     AA = A; bb = b; % AA and bb are the permuted versions of A and b
< 0.001        1 110     for k=1:K
  0.006    10000 111         start = tic;
  0.001    10000 112         i = mod(k, m) + 1;
  0.040    10000 113         x_MSRK = x_MSRK + ((bb(i) - AA(i,:)*x_MSRK)/(norm(AA(i,:), 2)^2)) * AA(i,:).';
                 114
  0.002    10000 115         if mod(k, m) == 0
  0.002       10 116             P = randperm(m);
  0.005       10 117             AA = AA(P,:);
< 0.001       10 118             bb = bb(P);
< 0.001       10 119         end
                 120
  0.009    10000 121         T_MSRK_temp(k) = toc(start);
  0.013    10000 122         E_MSRK_temp(k) = norm(x_MSRK - x_exact, 2)^2;
< 0.001    10000 123     end
< 0.001        1 124     E_MSRK = E_MSRK_temp; T_MSRK = T_MSRK_temp;
< 0.001        1 125 end
```

Figure 3: The first column shows the time spent on that line in seconds, the second column shows how many times that line has been run, the third column shows the line number in the file `SingleTrial.m`. See Appendix E for the full code.

From experiments in Section 4, we find that the Modified Simple Randomized Kaczmarz method and the standard Kaczmarz method are of comparable computational cost, whereas the Randomized Kaczmarz method, the Simple Randomized Kaczmarz method, and the Sampling Kaczmarz-Motzkin methods are all notably more expensive.

Now that we have an understanding of what the Modified Simple Randomized Kaczmarz method is, we prove its error converges exponentially in expectation. This proof is adapted from the proof of exponential convergence for an optimized randomized scheme in [17] by Dai et al. The proof by Dai et al. for the optimized randomized scheme presumes randomized selection with replacement,

whereas we give a proof for a method without replacement. In writing this report, we could not find a proof of a randomized scheme where the rows are selected without replacement in the literature. So this is a proof of not only a novel method, but for a different class of methods to those with existing proofs.

3.1 Setup

The following setup is very similar to that in section *'II. Optimized RKA'* in [17].

Let

$$\tilde{A}\boldsymbol{x} = \tilde{\boldsymbol{b}},$$

where $\tilde{A} \in \mathbb{R}^{m\times n}, m > n$, is of full column rank, $\boldsymbol{x} \in \mathbb{R}^n$ is the unknown vector, and $\tilde{\boldsymbol{b}} \in \mathbb{R}^m$. Denote the i-th row of $\tilde{A}$ by $\tilde{\boldsymbol{a}}_i^T$.

Restate the problem so that $\tilde{A}$ has normalised rows and $\tilde{\boldsymbol{b}}$ is scaled accordingly, that is,

$$A\boldsymbol{x} = \boldsymbol{b}, \text{ where } A := \operatorname{diag}(\|\tilde{\boldsymbol{a}}_1^T\|_2^{-1}, \ldots, \|\tilde{\boldsymbol{a}}_m^T\|_2^{-1})\tilde{A}, \quad \text{and } \boldsymbol{b} := \operatorname{diag}(\|\tilde{\boldsymbol{a}}_1^T\|_2^{-1}, \ldots, \|\tilde{\boldsymbol{a}}_m^T\|_2^{-1})\tilde{\boldsymbol{b}}.$$

Let the i-th row of A be denoted by $\boldsymbol{a}_i^T$. Let $\boldsymbol{x}_0$ be the initial approximation of $\boldsymbol{x}$. Recall from Equation (2) that in the standard Kaczmarz method, given $\boldsymbol{x}_k$, we compute $\boldsymbol{x}_{k+1}$ as follows:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \frac{b_i - \boldsymbol{a}_i^T\boldsymbol{x}_k}{\|\boldsymbol{a}_i\|_2^2}\boldsymbol{a}_i,$$

where $i = (k \mod m) + 1$.

Consider picking the rows of A randomly with some probability vector $\boldsymbol{p} \in \mathbb{R}^m$, so $p_i \geq 0\ \forall i$, and $\mathbb{1}^T\boldsymbol{p} = 1$, by the properties of probability vectors. Row i is selected with probability p_i.

We make use of a property of the orthogonal projection operation. If we have vectors $\boldsymbol{u}$ and $\boldsymbol{v}$, the component of $\boldsymbol{u}$ in the $\boldsymbol{v}$ direction can be written as $\boldsymbol{u}\sin(\theta)$, where θ is the angle between $\boldsymbol{u}$ and $\boldsymbol{v}$. Therefore, as we define the $\boldsymbol{x}_k$ by orthogonally projecting $\boldsymbol{x}_{k-1}$ onto some hyperplane, we can write:

$$\|\boldsymbol{x}_k - \boldsymbol{x}\|_2^2 = \|\boldsymbol{x}_{k-1} - \boldsymbol{x}\|_2^2 \sin^2(\alpha_i), \tag{4}$$

where α_i denotes the angle between $\boldsymbol{x}_{k-1} - \boldsymbol{x}$ and the hyperplane associated with the selected row, $\boldsymbol{a}_i$. This is merely an extension of the orthogonal projection operation property.

As the row selection is random, we must look at the *expected* error. This is defined similarly to how expectation is usually defined, we take the sum of errors weighed against the probability of that error occurring.

Hence, we have that

$$\mathbb{E}_{\cdot|\boldsymbol{x}_{k-1}}\left[\|\boldsymbol{x}_k - \boldsymbol{x}\|_2^2\right] = \|\boldsymbol{x}_{k-1} - \boldsymbol{x}\|_2^2 \sum_{i=1}^{m} p_i \sin^2(\alpha_i),$$

where $\mathbb{E}_{\cdot|\boldsymbol{x}_{k-1}}$ denotes the expectation given $\boldsymbol{x}_{k-1}$. For the sake of brevity, henceforth we will use $\mathbb{E}$ to denote $\mathbb{E}_{\cdot|\boldsymbol{x}_0}$.

3.2 Proof of convergence

Consider the first m iterations of the Modified Simple Randomized Kaczmarz method. Let the row chosen on the k-th iteration (where $k \leq m$) be denoted by $j_{(k)}$. We will sample the m rows of A without replacement. Hence, the sequence $\left(j_{(k)}\right)_{k=1}^{m}$ is a permutation of $(1, \ldots, m)$. Let $\theta_i^{(k)}$ denote the angle between $\boldsymbol{x}_k - \boldsymbol{x}$ and row $\boldsymbol{a}_i$. Let the probability distribution vector at iteration $k+1$ be denoted by $\boldsymbol{p}^{(k)}$. We chose the row of A uniformly at random (i.e. $p_i^{(0)} = \frac{1}{m},\ \forall i = 1, \ldots, m$). So,

$$\mathbb{E}\left[\|\boldsymbol{x}_1 - \boldsymbol{x}\|_2^2\right] = \|\boldsymbol{x}_0 - \boldsymbol{x}\|_2^2 \sum_{i=1}^{m} \frac{1}{m} \sin^2\left(\theta_i^{(0)}\right).$$

Similarly, for the second iteration we get,

$$\mathbb{E}\left[\|\boldsymbol{x}_2 - \boldsymbol{x}\|_2^2\right] = \mathbb{E}\left[\|\boldsymbol{x}_1 - \boldsymbol{x}\|_2^2\right] \sum_{\substack{i=1 \\ i \neq j_{(1)}}}^{m} \frac{1}{m-1} \sin^2\left(\theta_i^{(1)}\right).$$

Notice that we only sum over $m-1$ terms. This is because we have picked, without replacement, one of the rows of A already (namely $j_{(1)}$), so we can not pick it again. As we are picking uniformly at random between $m-1$ rows now, the probability of any given row being picked is $\frac{1}{m-1}$, i.e.

$$p_i^{(1)} = \begin{cases} \frac{1}{m-1} & \forall i \in \{1, \ldots, m\} \setminus \left\{j_{(1)}\right\} \\ 0 & i = j_{(1)} \end{cases}.$$

Let us define the sequence $(\Omega_l)_{l=0}^{m-1}$ as follows:

$$\begin{aligned}
\Omega_0 &:= \sum_{i=1}^{m} \frac{1}{m} \sin^2\left(\theta_i^{(0)}\right), \\
\Omega_1 &:= \sum_{\substack{i=1 \\ i \neq j_{(1)}}}^{m} \frac{1}{m-1} \sin^2\left(\theta_i^{(1)}\right), \\
&\ldots \\
\Omega_l &:= \sum_{\substack{i=1 \\ i \notin \left\{j_{(1)}, \ldots, j_{(l)}\right\}}}^{m} \frac{1}{m-l} \sin^2\left(\theta_i^{(l)}\right), \\
&\ldots \\
\Omega_{m-1} &:= 1 \cdot \sin^2\left(\theta_{j_{(m)}}^{(m-1)}\right).
\end{aligned}$$

Hence, we can write:

$$\begin{aligned}
\mathbb{E}\left[\|\boldsymbol{x}_k - \boldsymbol{x}\|_2^2\right] &= \mathbb{E}\left[\|\boldsymbol{x}_{k-1} - \boldsymbol{x}\|_2^2\right] \cdot \Omega_{k-1} \\
&= \mathbb{E}\left[\|\boldsymbol{x}_{k-2} - \boldsymbol{x}\|_2^2\right] \cdot \Omega_{k-2} \cdot \Omega_{k-1} \\
&\dots \\
&= \mathbb{E}\left[\|\boldsymbol{x}_0 - \boldsymbol{x}\|_2^2\right] \cdot \prod_{i=0}^{k-1} \Omega_i \\
&= \|\boldsymbol{x}_0 - \boldsymbol{x}\|_2^2 \cdot \prod_{i=0}^{k-1} \Omega_i \\
&= \|\boldsymbol{x}_0 - \boldsymbol{x}\|_2^2 \cdot \Omega^{(k)},
\end{aligned}$$

where $\Omega^{(k)} := \prod_{i=0}^{k-1} \Omega_i$. So,

$$\mathbb{E}\left[\|\boldsymbol{x}_k - \boldsymbol{x}\|_2^2\right] = \|\boldsymbol{x}_0 - \boldsymbol{x}\|_2^2 \cdot \Omega^{(k)}.$$

Claim: If $m > n$ and $k \in \{0, \dots, m-n-1\}$, then $\Omega_k < 1$.
Proof: Assume for contradiction that $m > n$, $k \in \{0, \dots, m-n-1\}$, and $\Omega_k = 1$. Note that we can't have Ω_k be strictly greater than 1, as $0 \leq \sin^2(\theta) \leq 1$ for all values of θ, and by the law of total probability $\mathbb{1}^T \boldsymbol{p} = 1$. So,

$$\begin{aligned}
\Omega_k = 1 &\iff \sin^2\left(\theta_i^{(k)}\right) = 1, \ \forall i : p_i^{(k)} \neq 0, \\
&\iff \cos\left(\theta_i^{(k)}\right) = 0, \ \forall i : p_i^{(k)} \neq 0, \\
&\iff (\boldsymbol{x}_k - \boldsymbol{x}) \perp \boldsymbol{a}_i, \ \forall i : p_i^{(k)} \neq 0.
\end{aligned}$$

At iteration number $k+1$, the vector $\boldsymbol{p}^{(k)}$ has $m-k$ non-zero entries. This is because we have selected k rows, so the probability of them being selected again is 0. So the vector $\boldsymbol{a}_i$ must be orthogonal to $\boldsymbol{x}_k - \boldsymbol{x}$ for $m-k$ distinct values of i. Note that $m-k > n$ for $k \in \{0, \dots, m-n-1\}$. As $\boldsymbol{a}_i \in \mathbb{R}^n$ and $(\boldsymbol{x}_k - \boldsymbol{x}) \in \mathbb{R}^n$, we can't have $\boldsymbol{x}_k - \boldsymbol{x}$ be orthogonal to $\boldsymbol{a}_i$ for $m-k > n$ distinct values of i. ↯. Hence, if $m > n$, then $\Omega_k < 1$ for $k \in \{0, \dots, m-n-1\}$. □

Therefore, for $k \in \{0, \dots, m-n-1\}$ we have that $\Omega_k < 1$, and for $k \in \{m-n, \dots, m-1\}$ we have that $\Omega_k \leq 1$. So, for $k \in \{1, \dots, m\}$, we can write $\mathbb{E}\left[\|\boldsymbol{x}_k - \boldsymbol{x}\|_2^2\right] = \|\boldsymbol{x}_0 - \boldsymbol{x}\|_2^2 \cdot \Omega^{(k)}$, where $\Omega^{(k)} < 1$. Once we reach $k = m$, we can reset k to 0 and repeat the process with probability vector $\boldsymbol{p}$ such that $p_i = \frac{1}{m}, \ \forall i = 1, \dots, m$.

Let $\Omega := \Omega^{(m)}$. Note that, if we start with our initial approximation on a hyperplane, $\Omega < 1$ is a constant for a given matrix. This is because projections in different orders will result in the same coefficient, Ω, at the end. This is a consequence of the angles between the hyperplanes being fixed. When we repeat the process using the same hyperplanes as before, with iterations from $\{m, \dots, 2m-1\}$, the expected factor by which the error decreases is once again Ω. If we repeat this process, we get expected exponential convergence in κ, where $k = \kappa m$ and κ is a positive integer. As $\Omega < 1$ is a constant, after $k \in \mathbb{N}_{>0}$ iterations, the expected factor of reduction in the error is $\Omega^{k/m}$.

3.3 Discussion

The Modified Simple Randomized Kaczmarz method converges exponentially when looking at cycles. As it happens, this is not as tight as the bounds that exist for the Randomized Kaczmarz method or the Sampling Kaczmarz-Motzkin method, as we find out in Section 5. However, from experiments, the Simple Randomized Kaczmarz method is more computationally efficient than both of

these. This is because we don't need the random selection of rows on every iteration, which can be computationally expensive, nor do we need to compute the row norms of A at the start of the algorithm, so there is less overhead.

From experiments, the Modified Simple Randomized Kaczmarz method is basically as computationally efficient as the standard Kaczmarz method, and both are far cheaper than the Randomized Kaczmarz method while offering essentially the same per-iteration improvement as we observe in the Randomized Kaczmarz method. These advantages also extend to the Sampling Kaczmarz-Motzkin method, the Modified Simple Randomized Kaczmarz method outperforms it in terms of the computational cost, but not the number of iterations.

Intuition suggests that the Modified Simple Randomized Kaczmarz method should be more robust than the standard Kaczmarz method, as we change the order in which we sample the rows every cycle, so a particularly unfortunate ordering of rows shouldn't affect the Modified Simple Randomized Kaczmarz method. However, as discussed in [19] with regards to the Randomized Kaczmarz method, the main advantage that the Modified Simple Randomized Kaczmarz method would pose over the standard Kaczmarz method is that we can prove it convergence exponential in expectation. The advantage of the Modified Simple Randomized Kaczmarz method over the Randomized Kaczmarz method and the Sampling Kaczmarz-Motzkin method would come from the computational costs. However, this is not something we prove, rather it is something we observe from the experiments.

4 Numerical experiments using MATLAB

We will perform experiments to compare our model against existing methods using random and real-world data, similarly to De Loera et al. in [11]. The specifications of the setup we used for the experiments are detailed in Table 3 in Appendix F. The code used to run the experiments is given in Appendix E. Given the setup of our problem, we focus primarily on consistent, over-determined systems of linear equations, however, we also demonstrate that these results also seem to work in the case of inconsistent systems.

In the random case, we randomly sample the entries in the matrix A. In the real-world data case, we use existing an existing data set to set A. In both cases, We randomly generate the vector we are trying to find, $\boldsymbol{x}$. Using these, we create the vector $\boldsymbol{b} := A\boldsymbol{x}$. We then run the methods, using A and $\boldsymbol{b}$ as inputs, and compare the errors with respect to $\boldsymbol{x}$. We keep track of the error at each iteration and the time taken to compute each iteration.

Note that the following comparisons are not an exhaustive catalogue of the systems we have considered. We excluded similarly performing setups for the sake of brevity. Appendix E.1 contains the vestigial code should a motivated reader wish to experiment further. To that end, we only consider $\boldsymbol{x}$ where each entry is generated by the standard normal distribution.

In our graphs we often see the error plateau to some limiting value. In the case of consistent systems, this is a result of how MATLAB deals with small values. We are bounded by how small the numbers MATLAB can accurately perform computation are. This number is known as *machine epsilon* and is approximately 2.22×10^{-16} for the MATLAB `'double'` datatype [22]. If we take the value of $\boldsymbol{x}_k$ once it has plateaued and calculate the residual by $\|\boldsymbol{b} - A\boldsymbol{x}_k\|$, we obtain values roughly on the 10^{-15} order of magnitude. So the plateau is as a result of the residual $b_i - \boldsymbol{a}_i^T\boldsymbol{x}_k$ from Equation (2) being within the realm of machine epsilon for MATLAB. With the systems we use, this usually corresponds to errors in $\boldsymbol{x}$ of the order 10^{-29}. Of course, all of these algorithms would converge to 0 with more iterations, but for computing smaller errors, we need a computational setup with a smaller machine epsilon. In the case of inconsistent systems, we often reach a limiting value quicker, this is expanded upon by Wang et al. in [13].

From [11], we know that (when not using relaxation) the optimal β value for the Sampling Kaczmarz-Motzkin method is often small, within 10% of the number of rows m. In our numerical experiments, we approximate the optimal value for β by trial and error using a binary search method. This method may be prone to error and the value for β we find may not be optimal. Nonetheless, it should perform similarly to the optimal β. This is a shortcoming of the Sampling Kaczmarz-Motzkin method, as we don't have an explicit way to find the optimal β. We discuss the difficulties in choosing β further in Appendix D.

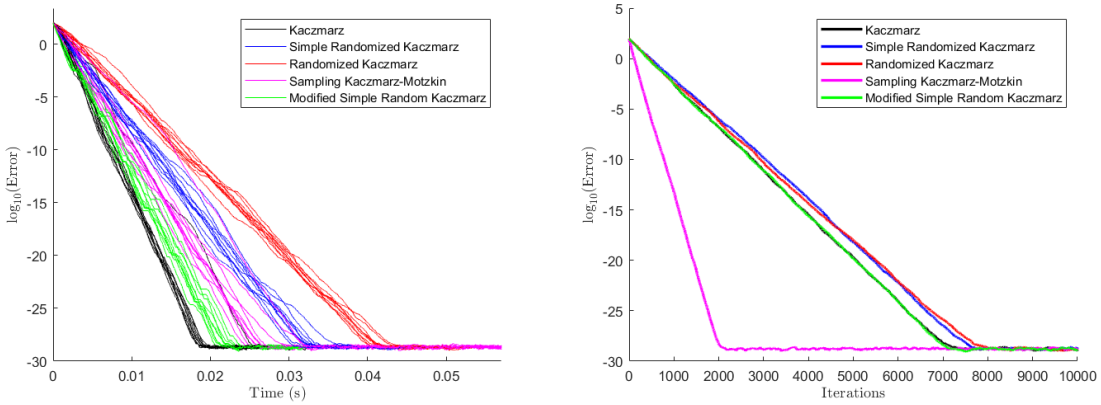
We first run the experiments for randomly generated data in Section 4.1, then real-world data in Section 4.2, and finally in Section 4.3 we consider the special case of when A is orthogonal. Sections 4.1 and 4.2 will give us an idea of how these methods perform in general. We then use the curious behaviour demonstrated in Section 4.3 to give some intuition into why these methods perform as they do. This will give us the requisite information to compare the methods more generally in Section 5.

4.1 Matrix A sampled from $\mathcal{N}(0,1)$

Consider the matrix A of dimensions 1000×100, where each entry of A is chosen by the standard normal distribution, $\mathcal{N}(0,1)$. We choose our exact solution $\boldsymbol{x}$ to have dimensions 100×1 and have each of its entries also chosen by the standard normal distribution. We define the 1000×1 vector $\boldsymbol{b}$ by $\boldsymbol{b} := A\boldsymbol{x}$.

We compute the results for 10 realisation using the seeds $1, ..., 10$ for our random number generator, this is so our results can be easily replicated. We plot the log error against the computation time required to get to that error. Note that the computation time may vary depending on background processes the computer may be executing, so we may see some outlier realisations. As the methods compare similarly when considering the log error against the number of iterations, we only plot one realisation, for visual clarity in our graph. The code we use is included in Appendix E and is thoroughly commented, should the reader wish to try this with a different number of realisations.

Running 10 realisations of each method for 10000 iterations, we obtain Figure 4. Note that we only plot the first realisation Figure 4b. In both the consistent and inconsistent case we find that $\beta = 6$ works best for the Sampling Kaczmarz-Method.

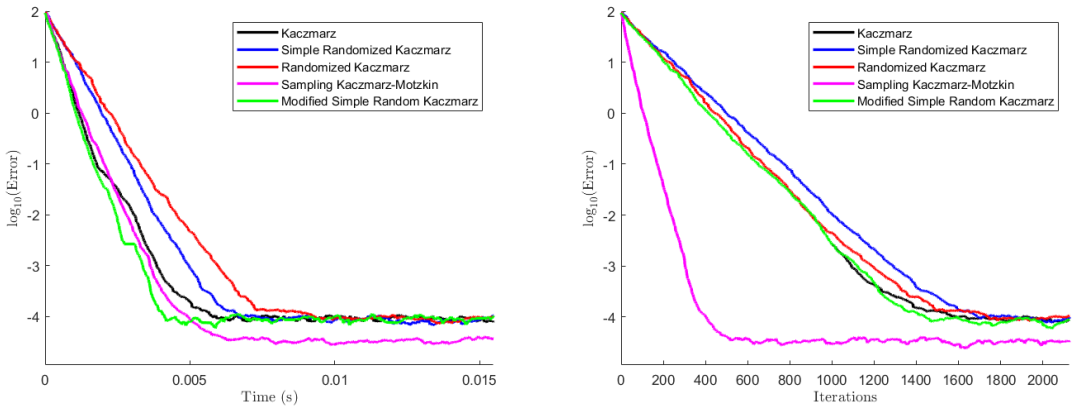


(a) Log error against time, 10 realisations

(b) Log error against iterations, 1 realisation

Figure 4: Consistent system with matrix $A \in \mathbb{R}^{1000\times100}$ sampled from standard normal, vector $\boldsymbol{x} \in \mathbb{R}^{100\times1}$ sampled from standard normal.

Many of the problems we are interested in solving give rise to inconsistent systems. So far, we have only discussed these methods in the context of consistent systems. What happens if the system is inconsistent? This is shown in Figure 5. We can obtain an inconsistent system from our consistent system by adding a small, normally distributed noise term to $\boldsymbol{b}$. For each entry in $\boldsymbol{b}$ we add a random realisation of $\mathcal{N}(0, 0.01)$. Because of the random noise, the value to which the methods converge varies [12], so for clarity in the graph we only plot one realisation. As this system converges much faster, we only consider the first 3000 iterations.



(a) Log error against time, 1 realisations

(b) Log error against iterations, 1 realisation

Figure 5: Inconsistent system with matrix $A \in \mathbb{R}^{1000\times 100}$ sampled from standard normal, vector $\boldsymbol{x} \in \mathbb{R}^{100\times 1}$ sampled from standard normal, with noise term $\mathcal{N}(0, 0.01)$ on $\boldsymbol{b}$. We use $\beta = 6$ for the Sampling Kaczmarz-Motzkin method.

Of course, when using these methods with real-world applications there may be underlying patterns that affect how these methods perform against each other. In Section 4.2 we test these methods using real-world data.

4.2 Real-world data

Here we recreate one of the real-world experiments performed by De Loera et al. in [11]. The problem arises in the context of machine learning. We attempt to classify data points given some attributes, one method by which we can do this is using a Support-vector Machine (SVM) model. The precise method by which SVM works is not relevant for this report, we only point out that solving an over-determined system of linear equations is part of the process.

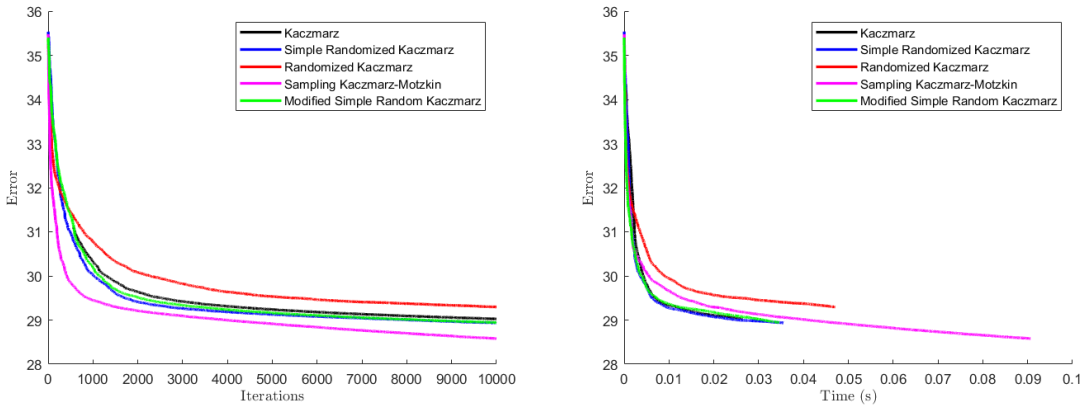
The data we use is from the *'Breast Cancer Wisconsin (Diagnostic) Data Set'*[3] from the UCI Machine Learning Repository [23]. We use this data set to set the matrix A, which will have dimensions 569×30. Here, the 569 rows correspond to different patients and the 32 columns correspond to different attributes that were measured. This data set has no missing values and the attributes are all real-valued.

We use this data set to create a linear system, as in [11]. De Loera et al. generate $\boldsymbol{b}$ as `b = ... 10^(-6)*ones(m, 1);`, i.e. a vector of size $m \times 1$ where each entry is 10^{-6}. They then run their experiments, keeping track of the error $\|\boldsymbol{b} - A\boldsymbol{x}_k\|_2^2$. However, we first generate a $\boldsymbol{x}$, as we did in

[3]We only experiment on this data set and not the *'Credit Card Default Data Set'* from [11] because that data set leads to convergence in far fewer iterations than there are rows. So it is not useful for us to distinguish between the methods.

4.1, and then define $\boldsymbol{b} := A\boldsymbol{x}$. We compute the error as $\|\boldsymbol{x} - \boldsymbol{x}_k\|_2^2$ for the sake of consistency within this report.

The results after performing 10000 iterations with random seed 1 is shown in Figure 6. Note that, while the Sampling Kaczmarz-Motzkin method is faster in terms of the number of iterations, it does not outperform the Modified Simple Randomized Kaczmarz method when looking at the computational time. Part of the reason for this is that the Sampling Kaczmarz-Motzkin method is much more computationally expensive per iteration, as we discussed in 2.3.



(a) Error against number of iterations

(b) Error against computation time

Figure 6: Real-world experiment with data from the Wisconsin (Diagnostic) Breast Cancer data set [23], A has dimensions of 569×30. We use $\beta = 10$ as that seems to be roughly optimal from our experiments, this differs from the value given in [11], see Appendix D for further discussion.

So why is it that non-greedy methods that sample the rows without replacement perform as well as they do? While we don't have a rigorous justification, in Section 4.3 we give a specific example where sampling each row at least once is a natural choice. We suggest that this reasoning extends to non-special cases as well, albeit to a lesser extent.

4.3 Special case of A orthogonal

Take the case when the matrix A is square and orthogonal. Of course, with this setup, we can compute the exact value of $\boldsymbol{x}$ as $A^T\boldsymbol{b}$, which is computationally a very cheap operation. However, we take the time to explore this setup as it demonstrates behaviour that, at first glance, may seem very unusual, but with deeper consideration gives us an intuitive insight into the pros and cons of the different methods. This special case was noticed by Yuji Nakatsukasa for the Randomized Kaczmarz Method [24].

Consider the case when A is a square, orthogonal matrix. To generate this matrix, we take the Q-component of a QR-decomposition of an $m \times m$ matrix where each entry has been sampled from the standard normal distribution. We will refer to this matrix as having *'no noise'*. This can be generated in MATLAB as follows.

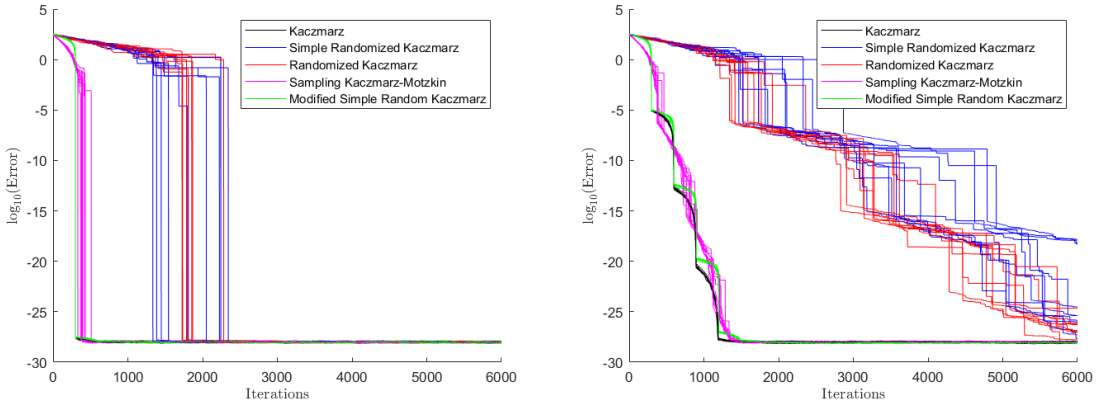
```
1  [A, ] = qr(randn(m));
```

Using this matrix A with parameters $m = 300$, $n = 300$, $K = 6000$ iterations, and $r = 10$ realisations, we obtain Figure 7a. Consider adding another randomly generated matrix, with entries from the

standard normal, which has been scaled by a factor of 10^{-5}. This resultant matrix, which we will refer to as *'A with noise'*, is *almost* orthogonal. This can be generated in MATLAB as follows.

```
[A, ] = qr(randn(m)); A = A + randn(m) * 1e-5;
```

Using this matrix A with noise, with the same parameters as before, we obtain Figure 7b. Note that the standard Kaczmarz method and the Modified Simple Randomized Kaczmarz method perform almost identically for reasons we will explain later.



(a) Orthogonal A with no noise

(b) Orthogonal A with noise of order 10^{-5}

Figure 7: Consistent system with $A \in \mathbb{R}^{300\times 300}$ orthogonal and vector $\boldsymbol{x} \in \mathbb{R}^{300\times 1}$ sampled from standard normal. We use $\beta = 12$ for Sampling Kaczmarz-Motzkin again.[4]

We observe a '*sudden convergence*' to 0 when A is orthogonal, or orthogonal with some noise term. When we have a noise term, we observe periodic occurrences of the sudden convergence. What are these phenomena, and why do the different methods vary in the number of iterations required to display this sudden convergence? We answer these questions for the system with no noise first, then the system with noise is a simple extension.

Fundamentally, all we are observing with the sudden convergence in Figure 7a is the algorithm finding the exact value of $\boldsymbol{x}$. As MATLAB can only perform computations to a precision of machine epsilon, once the exact solution for $\boldsymbol{x}$ is found, we still record an error in the order of 10^{-29} in $\boldsymbol{x}$.

To see why we find the exact value, we ask the reader to consider Figure 1 once again. As the rows of A are orthogonal, the hyperplanes $\langle \boldsymbol{a}_i, \boldsymbol{x}\rangle = b_i$ which we project onto are also orthogonal. Note that if we project onto hyperplane i, then the component of our error in the direction of the i-th hyperplane is now 0. In terms of Equation (4), after we have projected onto every single hyperplane, whichever hyperplane i we choose next, the angle between our approximation and hyperplane i will be $\alpha_i = 0$. So, $\sin^2(\alpha_i)$ will be 0, therefore, the error will be 0. So sudden convergence happens when every hyperplane has been visited. The sudden drop is a result of going from some non-zero error to 0.

The idea of needing to project onto every hyperplane at least once gives us insight into why the different methods differ in the number of iterations needed to converge suddenly. Both the standard Kaczmarz and the Modified Simple Randomized Kaczmarz method obtain the minimum necessary iterations for convergence, which is simply the number of orthogonal rows A has because in each of

[4]The optimal value, when considering the number of iterations, in this case would just be $\beta = 300$. With this value of β the method performs near identically to the standard Kaczmarz and the Modified Simple Randomized Kaczmarz methods. Note that we are not interested in the computation time here, just the sudden convergence.

these methods we can't project onto a hyperplane we have already selected until we have selected every other hyperplane first. If we had chosen $\beta = 300$ (which is optimal), the Sampling Kaczmarz-Motzkin method would also converge suddenly in 300 iterations, as it greedily selects the row with the largest residual, which will always be a row we haven't selected yet.

With the other randomized variants of the Kaczmarz method, because we select rows randomly with replacement, the number of iterations can vary. There is no stipulation to select a hyperplane we 'haven't yet selected' or the one with the largest residual. So, they take more iterations as they keep projecting onto hyperplanes they have already projected onto. If we project onto a hyperplane i that has been selected before, the angle between our approximation and hyperplane i will be $\alpha_i = \pi/2$. So, from Equation (4), the error will reduce by a factor of $\sin^2(\alpha_i) = 1$, i.e. it doesn't change. This corresponds to segments with a flat, horizontal line in Figure 7.

The probability that the Simple Randomized Kaczmarz converges suddenly in k iterations is equivalent to asking, *'What is the probability that one of the rows hasn't been selected in k iterations?'* For $k < m$, where m is the number of rows, this is 0 as not all the rows have been selected yet. For $k \geq m$ it is

$$\left(\frac{m-1}{m}\right)^k \cdot m.$$

Note that $\left(\frac{m-1}{m}\right)^k$ corresponds to the probability that some particular row hasn't been selected in k iterations, we then multiply this by the number of rows m to obtain the expression above. A similar procedure can be done to obtain the probability for the Randomized Kaczmarz Method, but we would have to take the different probability masses for each row into account.

This procedure is occurring repeatedly in Figure 7b. As we have a noise term in the order of 10^{-5}, we don't reach an error of exactly 0. Instead, after the first sudden convergence, the error $\|\boldsymbol{b} - A\boldsymbol{x}_k\|_2$ gets to within 10^{-6}. This corresponds to an error of magnitude 10^{-5} or 10^{-6} for $\|\boldsymbol{x} - \boldsymbol{x}_k\|_2$. Instead of dropping to a 0 error we drop to 10^{-6}, with the next sudden convergence we drop to an error of order 10^{-12} in $\|\boldsymbol{b} - A\boldsymbol{x}_k\|_2$, then 10^{-18}, and so on ad infinitum. Note that the error $\|\boldsymbol{b} - A\boldsymbol{x}_k\|_2$ will be different to the error $\|\boldsymbol{x} - \boldsymbol{x}_k\|_2$, which is what we plot.

This behaviour illustrates why we might want to select rows without replacement, as in certain cases projecting onto hyperplanes we have already selected may not give us any more information, so we should refrain from selecting them until we have tried the others. This is the key idea behind our Modified Simple Randomized Kaczmarz method. We also note that the Simple Randomized Kaczmarz Method is robust against patterns in the rows (e.g. the first $m/2$ rows being the same), unlike the standard Kaczmarz method (which would iterate through the first $m/2$ rows in order). So, it may be the case that with certain matrices the Modified Simple Randomized Kaczmarz method attains some error bound slightly quicker than the standard Kaczmarz method. The maximum number of iterations the Modified Simple Randomized Kaczmarz method can be quicker by is, at most, m. This is because it needs to select every row at least once before the next cycle, as with the standard Kaczmarz method. This brings us to Section 5 where we discuss how the methods compare more generally.

5 Comparing the methods

From Section 4.3 we have some insight into why it may be advantageous to not repeat a row selection till we have tried every other row. Is this also the case for non-orthogonal A? While we haven't proven it to be the case, experiments from Sections 4.1 and 4.2 certainly seem to suggest so. Motivated readers may wish to run the code in Appendix E to corroborate this. In general, with the classes of linear systems explored in Appendix E.1 and the real-world data we tried, the standard Kaczmarz method and the Modified Simple Randomized Kaczmarz method perform very similarly,

and they often outperform the Simple Randomized Kaczmarz and Randomized Kaczmarz methods in terms of the number of iterations. This could be suggestive of the idea that row-selection without replacement is superior to row-selection with replacement, though this is not something we have proved.

It is worth noting, however, that trying to minimize the number of iterations can be a red herring. Our main goal is to solve the system as efficiently as possible, to that end, we suggest that computational time is a better metric to use in comparing the methods against each other. This is highlighted by the Sampling Kaczmarz-Motzkin method, it outperforms every other method by a great deal in terms of the number of iterations needed, however, each iteration is much more computationally intensive. So when looking at computation time, it doesn't do as well as the standard Kaczmarz method or the Modified Simple Randomized Kaczmarz method. The Sampling Kaczmarz-Motzkin method does, however, outperform the Simple Randomized Kaczmarz method and the Randomized Kaczmarz method when we choose a reasonably good value for β. Though, we don't have an efficient way to find the optimal value for β. Furthermore, we suggest that, depending on the implementation of the method in code, the optimal value for β will also depend on the nature of the hardware and software on which the method is run, which further complicates the matter of choosing a good value for β. We expand on this idea in Appendix D.

The standard Kaczmarz method can perform just as well as, and often slightly better than, our Modified Simple Randomized Kaczmarz method when looking at computation time. The standard Kaczmarz method, however, does not have a proof for its exponential convergence, which is something every other method we have stated does have. In [19], Strohmer and Vershynin cite this provable expected exponential convergence as a benefit of the Randomized Kaczmarz method over the standard Kaczmarz method. The paper [19] was written in response to [18] by Censor et al., who point to the fact that the standard Kaczmarz method, in general, outperforms the Randomized Kaczmarz method in terms of the number of iterations, as a reason for the inferiority of the Randomized Kaczmarz method.

The Sampling Kaczmarz-Motzkin method goes some way in addressing these issues, by both demonstrating provable expected exponential convergence, while also outperforming the Randomized Kaczmarz method [11]. However, it still falls short of the standard Kaczmarz method.[5]

This is how our Modified Simple Randomized Kaczmarz method fits into the family of existing methods. It performs very similarly to the standard Kaczmarz method with only a very small extra computation cost per iteration, but also provably converges exponentially in expectation.

In our proof of convergence for the Modified Simple Randomized Kaczmarz method in Section 3.2, we show that the factor by which the error decreases after cycles of m iterations is just some constant $\Omega < 1$. However, we don't have a tighter bound on this constant, which is something that exists for the Randomized Kaczmarz method and the Sampling Kaczmarz-Motzkin method. These bounds are shown in Table 1.

So, while the Modified Simple Randomized Kaczmarz method doesn't unilaterally outperform the existing methods in every way, it still stands as a method with various benefits over the existing methods. We conclude with a discussion on what exactly this report has achieved, why it is important, and where we can go from here.

[5] See Section 4.1 for comparison to the standard Kaczmarz method

[6] See Appendix B for definitions of these terms. The specific bounds are not relevant to our discussion, we only point out the fact that they exist.

Method	Upper bound after m iterations
Randomized Kaczmarz [8]	$(1-\kappa(A)^{-2})^m$
Sampling Kaczmarz-Motzkin [25]	$(1-\|A^{-1}\|_2^{-2}\cdot m^{-1})^m$
Modified Simple Randomized Kaczmarz	some constant $\Omega < 1$

Table 1: Upper bound for the expected reduction in error after m iterations for each method. Here, $\kappa(A)$ refers to the scaled condition number of A, and $\|A^{-1}\|_2$ is the spectral norm of the left inverse of A.[6]

6 Conclusion

In this report, we considered various numerical methods to approximate the solution to a large, consistent, over-determined, system of linear equations, $A\boldsymbol{x} = \boldsymbol{b}$. A well-known way to do this is using the Kaczmarz method, and there exist many different variations upon the Kaczmarz method [8], [11], [14], [20]. The primary way in which the methods we discuss differ is in the way in which they select the rows of A. Not only do we compare the different methods and their various strengths and weaknesses, we think about the idea of row-selection more generally.

First, we looked at existing methods in Section 2. With the Kaczmarz method, we don't have a proof that the error reduces exponentially for general matrices. We consider variations of the Kaczmarz method, namely the Randomized Kaczmarz method and Simple Randomized Kaczmarz method from Strohmer and Vershynin [8], and the Sampling Kaczmarz-Motzkin method from De Loera et al. [11]. Under these methods, we can prove that the error reduces exponentially in expectation for general matrices. However, these methods are computationally more expensive than the standard Kaczmarz method.

To go some way in remedying this issue, we proposed the Modified Simple Randomized Kaczmarz method. We proved in Section 3 that with this method the error reduces exponentially in expectation, and showed via numerical experiments that often it is often more computationally efficient than the methods of Strohmer and Vershynin or De Loera et al.

As far as we can tell, the proof we provide in Section 3.2 differs from proofs of other methods in the literature in an important way. Of the methods we have seen discussed in the literature, they all sample from the rows of A *with* replacement, whereas in our method we sample *without* replacement. The special case of having the matrix A being orthogonal, which we considered in Section 4.3, quite naturally motivates this form of row-selection.

The other experiments that we conducted in Section 4 show that much of the time our method performs just as well as the Kaczmarz method, and occasionally better. We credit this to the highly efficient implementation in Algorithm 6. We also suggest that the Modified Simple Randomized Kaczmarz method can be more robust than the standard Kaczmarz method, as it won't be affected by unfavourable initial row orderings.

However, our method has some shortcomings that we discuss in Section 6. Both the Randomized Kaczmarz method and the Sampling Kaczmarz-Motzkin method have tighter bounds on convergence than we were able to give. Though, we think the experiments in Section 4 indicate that the Modified Simple Randomized Kaczmarz method could be provably better than the Randomized Kaczmarz or Sampling Kaczmarz-Motzkin methods.

It is also worth noting that with the Randomized Kaczmarz method, there is much more literature around it, proving the usability of it, or one of its variants, in a range of contexts. We particularly point out the following papers which; provide the exact mean squared error for inconsistent systems [12], show convergence even with a noise term [26], convergence to the minimum Euclidean norm in the least-squares case [27] for both over- and under-determined systems [15], [28]. We also point out that in the original paper by Strohmer and Vershynin, they give a proof for complex linear systems (i.e. taking values in $\mathbb{C}$), whereas we considered a real linear system (i.e. taking values in $\mathbb{R}$), though this is a relatively simple extension we ignore for simplicity, as we were emulating the proof in [17].

We believe that many of these properties can be extended to the Modified Simple Randomized Kaczmarz method. In particular, we point to Figure 5 to show the method does work in the inconsistent case as well. In [19], Strohmer and Vershynin invited the reader to come up with a method that is provably better than the Randomized Kaczmarz method. As we were limited by time, we were unable to prove that the Modified Simple Randomized Kaczmarz is provably better than existing methods, nonetheless, we hope that report provides enough evidence to say that it is a likely candidate.

Going beyond the Modified Simple Randomized Kaczmarz method, we emphasise the use of computation time when comparing methods, rather than the number of iterations. It may need more careful consideration, but is a better metric to measure errors against than the number of iterations, which can be counterproductive to the goal of finding solutions efficiently. We also hope that this report does enough to suggest that random sampling of rows *without* replacement merits further investigation. To parrot Strohmer and Vershynin in [19], 'it is absolutely possible that somebody, maybe the reader of this note, will come up with a version of the randomized Kaczmarz method that is provably better than the one we proposed.'

7 References

[1] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*, ser. Other Titles in Applied Mathematics. SIAM, 1997, ch. Preface, II, IV, ISBN: 9780898713619.

[2] A. Greenbaum, *Iterative methods for solving linear systems*. SIAM, 1997, ch. 1, ISBN: 9780898713961.

[3] J. J. Benedetto and P. J. Ferreira, *Modern Sampling Theory: Mathematics and Applications*, ser. Applied and Numerical Harmonic Analysis. Birkhäuser Boston, 2012, ISBN: 9781461201434.

[4] H. G. Feichtinger, C. Cenker, M. Mayer, H. Steier and T. Strohmer, 'New variants of the POCS method using affine subspaces of finite codimension with applications to irregular sampling,' in *Visual Communications and Image Processing'92*, International Society for Optics and Photonics, vol. 1818, 1992, pp. 299–310.

[5] F. Natterer, *The Mathematics of Computerized Tomography*, ser. Classics in Applied Mathematics. SIAM, 2001, ch. I, V, ISBN: 9780898714937.

[6] M. P. Deisenroth, A. A. Faisal and C. S. Ong, *Mathematics for Machine Learning*. Cambridge University Press, 2020, ch. 2, 9.

[7] NobelPrize.org. 'The nobel prize in physiology or medicine 1979.' (2021), [Online]. Available: `https://www.nobelprize.org/prizes/medicine/1979/summary/` (visited on 12/03/2021).

[8] T. Strohmer and R. Vershynin, 'A randomized Kaczmarz algorithm with exponential convergence,' *Journal of Fourier Analysis and Applications*, vol. 15, no. 2, pp. 262–278, 2009.

[9] G. N. Hounsfield, 'Computerized transverse axial scanning (tomography): Part 1. Description of system,' *The British journal of radiology*, vol. 46, no. 552, pp. 1016–1022, 1973.

[10] S. Kaczmarz, 'Angenaherte Auflosung von Systemen linearer Gleichungen,' *Bulletin International de l'Académie Polonaise des Sciences et des Lettres*, vol. 35, pp. 335–337, 1937.

[11] J. A. De Loera, J. Haddock and D. Needell, 'A sampling Kaczmarz–Motzkin algorithm for linear feasibility,' *SIAM Journal on Scientific Computing*, vol. 39, no. 5, S66–S87, 2017.

[12] A. Agaskar, C. Wang and Y. M. Lu, 'Randomized Kaczmarz algorithms: Exact MSE analysis and optimal sampling probabilities,' in *2014 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, IEEE, 2014, pp. 389–393.

[13] C. Wang, A. Agaskar and Y. M. Lu, 'Randomized Kaczmarz algorithm for inconsistent linear systems: An exact MSE analysis,' in *2015 International Conference on Sampling Theory and Applications (SampTA)*, IEEE, 2015, pp. 498–502.

[14] I. Necoara, 'Faster randomized block Kaczmarz algorithms,' *SIAM Journal on Matrix Analysis and Applications*, vol. 40, no. 4, pp. 1425–1452, 2019.

[15] Z.-Z. Bai and W.-T. Wu, 'On convergence rate of the randomized Kaczmarz method,' *Linear Algebra and its Applications*, vol. 553, pp. 252–269, 2018.

[16] G. T. Herman and L. B. Meyer, 'Algebraic reconstruction techniques can be made computationally efficient (positron emission tomography application),' *IEEE transactions on medical imaging*, vol. 12, no. 3, pp. 600–609, 1993.

[17] L. Dai, M. Soltanalian and K. Pelckmans, 'On the randomized Kaczmarz algorithm,' *IEEE Signal Processing Letters*, vol. 21, no. 3, pp. 330–333, 2013.

[18] Y. Censor, G. T. Herman and M. Jiang, 'A note on the behavior of the randomized Kaczmarz algorithm of Strohmer and Vershynin,' *Journal of Fourier Analysis and Applications*, vol. 15, no. 4, pp. 431–436, 2009.

[19] T. Strohmer and R. Vershynin, 'Comments on the randomized Kaczmarz method,' *Journal of Fourier Analysis and Applications*, vol. 15, no. 4, pp. 437–440, 2009.

[20] J. Nutini, B. Sepehry, I. Laradji, M. Schmidt, H. Koepke and A. Virani, 'Convergence rates for greedy kaczmarz algorithms, and faster randomized kaczmarz rules using the orthogonality graph,' in *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence*, AUAI Press, 2016, pp. 547–556.

[21] T. S. Motzkin and I. J. Schoenberg, 'The relaxation method for linear inequalities,' *Canadian Journal of Mathematics*, vol. 6, pp. 393–404, 1954.

[22] MATLAB, *version 9.5.0 (R2018b)*. Natick, Massachusetts: The MathWorks Inc., 2018, ch. eps, Built-in function.

[23] D. Dua and C. Graff, *UCI Machine Learning Repository*, 2017. [Online]. Available: `http://archive.ics.uci.edu/ml`.

[24] Y. Nakatsukasa, Private Communications, Yuji demonstrated the sudden convergence phenomena for orthogonal matrices in the Randomized Kaczmarz method in our communications, 2020.

[25] J. Haddock and A. Ma, 'Greed Works: An Improved Analysis of Sampling Kaczmarz–Motzkin,' *SIAM Journal on Mathematics of Data Science*, vol. 3, no. 1, pp. 342–368, 2021.

[26] D. Needell, 'Randomized Kaczmarz solver for noisy linear systems,' *BIT Numerical Mathematics*, vol. 50, no. 2, pp. 395–403, 2010.

[27] A. Zouzias and N. M. Freris, 'Randomized extended Kaczmarz for solving least squares,' *SIAM Journal on Matrix Analysis and Applications*, vol. 34, no. 2, pp. 773–793, 2013.

[28] A. Ma, D. Needell and A. Ramdas, 'Convergence properties of the randomized extended gauss–seidel and Kaczmarz methods,' *SIAM Journal on Matrix Analysis and Applications*, vol. 36, no. 4, pp. 1590–1604, 2015.

[29] J. W. Demmel, 'The probability that a numerical analysis problem is difficult,' *Mathematics of Computation*, vol. 50, no. 182, pp. 449–480, 1988.

[30] G. H. Gloub and C. F. Van Loan, 'Matrix computations,' *Johns Hopkins Universtiy Press, 3rd edtion*, 1996.

[31] R. A. Horn and C. R. Johnson, 'Norms for vectors and matrices,' *Matrix analysis*, pp. 313–386, 1990.

[32] G. B. Arfken and H. J. Weber, *Mathematical methods for physicists*. American Association of Physics Teachers, 1999, pp. 233–234.

[33] G. T. Herman, A. Lent and P. H. Lutz, 'Relaxation methods for image reconstruction,' *Communications of the ACM*, vol. 21, no. 2, pp. 152–158, 1978.

[34] S. Agmon, 'The relaxation method for linear inequalities,' *Canadian Journal of Mathematics*, vol. 6, pp. 382–392, 1954.

A Methods summary

Here we provide a quick reference guide to aid the reader. We summarise the row selection rule for each method.

Method name	**Row selection rule**	**Rows replaced?**
The (standard) Kaczmarz method	Cycle through the rows in the given order. Once we reach the end, we start again. (See Algorithm 1)	N/A
The Randomized Kaczmarz method	Pick rows randomly with probability proportional to their row norms. (See Algorithm 2)	Yes
The Simple Randomized Kaczmarz method	Pick rows randomly with uniform probability. (See Algorithm 3)	Yes
The Sampling Kaczmarz-Motzkin method	Pick a subset of the rows of size β randomly with uniform probability. From this subset, select the row which has the greatest residual. (See Algorithm 4)	Yes
The Modified Simple Randomized Kaczmarz Method	Pick rows randomly with uniform probability. Once every row has been selected, we start again. (See Algorithm 5.)	No

Table 2: Quick reference guide of how each method selects the rows

B The Scaled Condition Number

In [8], Strohmer and Vershynin make use of the *scaled condition number,* $\kappa(A)$, to give an upper bound for the expected rate of convergence for the Randomized Kaczmarz method. The scaled condition number for a matrix A, as defined by Demmel in [29], is

$$\kappa(A) := \|A\|_F \|A^{-1}\|_2.$$

Here, A^{-1} denotes the left-inverse of A, that is, $A^{-1}A = I_n$, The left inverse is assumed to exist as we assume A is of full column rank (i.e. $\text{rank}(A) = n$) and $m > n$. We use $\|\cdot\|_F$ and $\|\cdot\|_2$ to denote the Frobenius norm and the Spectral norm respectively. For a real matrix, the Frobenius norm is simply the square root of the sum of the squares of the elements [30]. For matrix $A = (a_{i,j})_{i=1,\dots,m,\ j=1,\dots,n}$ this can be written as

$$\|A\|_F := \left(\sum_{i=1}^{m} \sum_{j=1}^{n} a_{i,j}^2 \right).$$

We adapt the definition of the spectral norm in [31] for real matrices. Let A^T denote the transpose of A, and let α be the maximum eigenvalue of $A^T A$. Then,

$$\|A\|_2 := \alpha^{\frac{1}{2}}.$$

So, for real matrix A, the scaled condition number is $\|A\|_F\|A^{-1}\|_2$, where $\|\cdot\|_F$, $\|\cdot\|_2$, and A^{-1} are as defined above. Note that the scaled condition number can be arbitrarily large, regardless of the size of the matrix. So for matrices with a large scaled condition number, the rate of convergence will mainly be governed by the size of the scaled condition number and not the size of the matrix. Matrices with large condition numbers, e.g. $\kappa(A) > 10^{10}$, are known as *ill-conditioned* [32]. Even with ill-conditioned matrices, we expect exponential convergence, as can be seen from the bounds in Table 1.

C Relaxation

It has been noted by Strohmer and Vershynin in [8] that the Randomized Kaczmarz method can be improved by multiplying the $\frac{b_i - \boldsymbol{a}_i^T \boldsymbol{x}_k}{\|\boldsymbol{a}_i\|_2^2}\boldsymbol{a}_i$ term from Equation (2) by some scalar λ. This is known as *relaxation*; if $0 < \lambda < 1$ then we call this *under-relaxation* and if $1 < \lambda < 2$ we call this *over-relaxation* [21]. Note that we need λ to be in this range for consistent systems to ensure convergence [8], [33].

This is a well-known phenomenon in the literature that many methods make use of [21], [33], [34], including the Sampling Kaczmarz-Motzkin method. In [11], De Loera et al. also compare how the Sampling Kaczmarz-Motzkin method performs with different relaxation parameters λ, and it is rarely the case that $\lambda = 1$ (i.e. no relaxation) is optimal.

From cursory experiments, we found that our Modified Simple Randomized Kaczmarz method can also be improved through a carefully chosen relaxation parameter. However, we haven't proved it to be the case, nor do we have general guidelines for picking an optimal value of λ.

The standard Kaczmarz and the Randomized Kaczmarz methods don't make use of a relaxation parameter, so we only compare them against the Sampling Kaczmarz-Motzkin method with no relaxation parameter. This also saves us from having to find an optimal value for λ for each method. As most of the methods seem to benefit similarly through relaxation, we believe that ignoring relaxation is a fair choice.

D Optimal β with the Sampling Kaczmarz-Motzkin method

We don't have an efficient way to find the optimal value (in terms of computational cost) for β in the Sampling Kaczmarz-Motzkin method [11]. While larger values of β are better when considering the number of iterations (with $\beta = m$ being optimal), this comes with a much larger computation cost per iteration. De Loera et al. mention that the optimal value for β is rarely 1 or m, which correspond to the Simple Randomized Kaczmarz method and Motzkin method respectively.

The way the method is implemented and the hardware on which the method is run can also play a role in this. In Section 4.2 we see that the optimal value for β that we found ($\beta \approx 10$) differs from the one in [11] ($\beta \approx 150$). One reason for this is how the methods are implemented, specifically in regards to how the row with the largest residual is chosen at each iteration. We can see in Appendix E.2 that we find the row with the largest residual as follows.

```
1 tau_k = randi(m, 1, beta);      % Choose beta rows of A uniformly at random
2
3 % Compute residual for each of the previously selected rows
4 resids = zeros(1, beta);
5 for iter=1:beta
```

```
    resids(iter) = abs(A(tau_k(iter),:)*x_SKM - b(tau_k(iter)));
end

% Choose row of A from the subset with maximum residual
[ , t] = max(resids);              % t is the index of max value in resids
i = tau_k(t);                      % Index of max value in A
```

In [11] it is found as follows.

```
function [l,a] = maxviolatedofsample_constraint(A,b,x,beta)

%l : amount of violation
%a : row of A (hyperplane normal) that defines the most violated constraint inside
    %the sample of size beta

m = size(A,1);  %number of inequalities

perm=randsample(m,beta);

[l,ind]=max(A(perm(1:size(perm,1)),:)*x-b(perm(1:size(perm,1))));
a=A(perm(ind),:);

l = max(l,0);

end
```

These implementations could result in different optimal values for β. We believe that using what we found to be a near-optimal value of β using our implementation is most fair for the comparisons. This is because when we tried the method in [11], it performed worse. While we could replicate the figures they produced, we couldn't replicate their implementation's superiority over the Randomized Kaczmarz method, whereas we could within our code. Hence we think using our implementation is better. These differences could be as a result of the different hardware on which we ran the tests, we give the technical specifications of our hardware and software in Appendix F.

Notice that given a subset of rows τ_k, we can calculate the residual for each of the rows independently, i.e. they are not dependent on one another. So we can compute the residuals for rows in our sample in parallel. Having more capacity to perform these calculations in parallel will lead to less computation time per iteration as well as a greater optimal value for β on that system. The implementation we used does not take advantage of this ability to compute residuals in parallel.

With certain implementations of the Sampling Kaczmarz-Motzkin method, the optimal value for β is not only dependent on the linear system but also the parallel processing capabilities of the hardware and software that it is run on.

E MATLAB Code

We include the code used to conduct the numerical experiments here. The code is commented to aid future reuse. In the file `Data.m` the user may set up classes of linear systems different to those discussed in this report.

The `Data.m` file generates the linear system $A\boldsymbol{x} = \boldsymbol{b}$. The function `Data` is used in `SingleTrial.m`, which executes each of the methods, storing the error and time taken at each iteration. In `main.m` we repeat `SingleTrail.m` for `r` realisations with seeds $1, ..., r$, then we plot the results.

E.1 Data.m

In this file we generate the system of linear equations, `A*x_exact = b`. The **`Data.m`** function takes the size of the matrix A as $m \times n$ and a random seed as inputs. We leave residual code that was used in testing but wasn't written about, should the reader wish to experiment further.

```
% Generate the system
% Comment / uncomment section for the relevant setup.

function [A, b, x_exact] = Data(m, n, seed)
    rng(seed)                                   % Set random seed

%%% Choose the method for generating random matrix A
    A = normrnd(0, 1, [m, n]);                  % Sampled from N(0, 1) distribution
%     A = gallery('randsvd', [m, n], 1);          % Well-conditioned matrix
%     A = rand(m, n);                             % Sampled from Unif(0, 1) ...
    distribution
%     [A, ] = qr(randn(m));                       % Orthogonal A
%     [A, ] = qr(randn(m)); A = A + randn(m) * 1e-5;   % Orthogonal A with noise

    % Matrix with normally distributed rows, with variance increasing
    % proportial to row number
%     A = zeros(m, n);
%     for i=1:m
%        A(i,:) = normrnd(0, i/(n^0.5), [1, n]);
%     end

%%% Choose the exact value of x
x_exact = normrnd(0, 1, [n, 1]);            % Sampled from N(0, 1) distribution
% x_exact = rand(n, 1);                       % Sampled from Unif(0, 1) distribution

%%% Choose to make the system consistent or inconsistent with some small
%%% added noise term that is normally distributed with a small variance
    b = A*x_exact;                              % Consistent system
%     b = A*x_exact + normrnd(0, 0.01, [n, 1])';    % Added normal noise term
end
```

E.2 SingleTrial.m

This function runs each method once using the given parameters and outputs the error and time taken at each iteration, for each method. The inputs are `m`, `n`, `K`, `seed`, where $m \times n$ is the dimensions of our matrix A, K is the number of iterations to run, and `seed` is the seed that the random number generator uses.

The outputs are `E_K`, `E_SRK`, `E_RK`, `E_SKM`, `E_MSRK`, `T_K`, `T_SRK`, `T_RK`, `T_SKM`, `T_MSRK`. Each of these outputs is a list of length K. The variables with the prefix "`E_`" store the error at each iteration, and the variables with prefix "`T_`" store the time taken at each iteration. The suffix represents the specific method.

Key: Kaczmarz (`K`), Simple Randomized Kaczmarz (`SRK`), Randomized Kaczmarz (`RK`), Sampling Kaczmarz-Motzkin (`SKM`), Modified Simple Randomized Kaczmarz (`MSRK`).

```
function [E_K, E_SRK, E_RK, E_SKM, E_MSRK, T_K, T_SRK, T_RK, T_SKM, T_MSRK]...
    = SingleTrial(m, n, K, seed)
    %% Setup
    [A, b, x_exact] = Data(m, n, seed);

    % Initial guesses (i.e. x_0)
```

```
    x_K = zeros(n, 1); x_SRK = zeros(n, 1);
    x_RK = zeros(n, 1); x_MSRK = zeros(n, 1);
    x_SKM = zeros(n, 1);

    % Initialise list of errors at each iteration
    E_K_temp = zeros(1, K); E_SRK_temp = zeros(1, K);
    E_RK_temp = zeros(1, K); E_MSRK_temp = zeros(1, K);
    E_SKM_temp = zeros(1, K);

    % Initialise list of times at each iteration
    T_K_temp = zeros(1, K); T_SRK_temp = zeros(1, K);
    T_RK_temp = zeros(1, K); T_MSRK_temp = zeros(1, K);
    T_SKM_temp = zeros(1, K);

    %% Standard Kaczmarz
    for k=1:K
        start = tic;
        i = mod(k, m) + 1;              % Pick the row
        x_K = x_K + ((b(i) - A(i,:)*x_K)/(norm(A(i,:), 2)^2)) * A(i,:).';

        T_K_temp(k) = toc(start);
        E_K_temp(k) = norm(x_K - x_exact, 2)^2;
    end
    E_K = E_K_temp; T_K = T_K_temp;


    %% Simple Randomized Kaczmarz
    for k=1:K
        start = tic;
        i = randi(m);                   % Pick a row uniformly at random
        x_SRK = x_SRK + ((b(i) - A(i,:)*x_SRK)/(norm(A(i,:), 2)^2)) * A(i,:).';

        T_SRK_temp(k) = toc(start);
        E_SRK_temp(k) = norm(x_SRK - x_exact, 2)^2;
    end
    E_SRK = E_SRK_temp; T_SRK = T_SRK_temp;


    %% Randomized Kaczmarz
    % Setting array of probabilities
    p = zeros(1, m);
    A_F2 = norm(A, 'fro')^2;            % Square of the Frobenius norm of A, used ...
        to normalise p
    for j=1:m                           % Compute probability for each row
       p(j) = (norm(A(j,:), 2)^2)/A_F2;
    end
    cp = [0, cumsum(p)];                % Cumulative sum of probabilities

    % Run iterations
    for k=1:K
        start = tic;

        i = find(rand>cp, 1, 'last');   % Row index for this random iteration
        x_RK = x_RK + ((b(i) - A(i,:)*x_RK)/(norm(A(i,:), 2)^2)) * A(i,:).';

        T_RK_temp(k) = toc(start);
        E_RK_temp(k) = norm(x_RK - x_exact, 2)^2;
    end
    E_RK = E_RK_temp; T_RK = T_RK_temp;


    %% Sampling Kaczmarz-Motzkin
    beta = 12;
    for k=1:K
        start = tic;
        tau_k = randi(m, 1, beta);      % Choose beta rows of A uniformly at random

        % Compute residual for each of the previously selected rows
```

```
74          resids = zeros(1, beta);
75          for iter=1:beta
76              resids(iter) = abs(A(tau_k(iter),:)*x_SKM - b(tau_k(iter)));
77          end
78
79          % Choose row of A from the subset with maximum residual
80          [ , t] = max(resids);               % t is the index of max value in resids
81          i = tau_k(t);                       % Index of max value in A
82
83          x_SKM = x_SKM + ((b(i) - A(i,:)*x_SKM)/(norm(A(i,:),2)^2)) * A(i,:).';
84
85          T_SKM_temp(k) = toc(start);
86          E_SKM_temp(k) = norm(x_SKM - x_exact, 2)^2;
87      end
88      E_SKM = E_SKM_temp; T_SKM = T_SKM_temp;
89
90
91      %% Modified Simple Randomized Kaczmarz
92      AA = A; bb = b; % AA and bb are the permuted versions of A and b
93      for k=1:K
94          start = tic;
95          i = mod(k, m) + 1;
96          x_MSRK = x_MSRK + ((bb(i) - AA(i,:)*x_MSRK)/(norm(AA(i,:), 2)^2)) * AA(i,:).';
97
98          if mod(k, m) == 0
99              P = randperm(m);
100             AA = AA(P,:);
101             bb = bb(P);
102         end
103
104         T_MSRK_temp(k) = toc(start);
105         E_MSRK_temp(k) = norm(x_MSRK - x_exact, 2)^2;
106     end
107     E_MSRK = E_MSRK_temp; T_MSRK = T_MSRK_temp;
108 end
```

E.3 `main.m`

We run each of the methods for 10 realisations with different systems and different random seeds. We set up our linear systems in the file `Data.m`. We make use of the function `SingleTrial` which runs each of the methods once and outputs, for each method, the list of errors and times for each iteration. Finally, we plot the results.

```
1  %% Setup
2  % Set parameters
3  r = 10;              % Number of realisations
4  m = 1000; n = 100;   % Dimensions of our matrix A
5  K = 10000;           % Number of iterations
6
7  % Set up lists of errors and times for each method. The dimensions are r*K,
8  % where r is the number of realisations and K is the number of iterations
9  % that we are computing.
10
11 % Initialise list of least-square errors
12 E_K = zeros(r, K); E_SRK = zeros(r, K);
13 E_RK = zeros(r, K); E_MSRK = zeros(r, K);
14 E_SKM = zeros(r, K);
15 % Initialise list of times
16 T_K = zeros(r, K); T_SRK = zeros(r, K);
17 T_RK = zeros(r, K); T_MSRK = zeros(r, K);
18 T_SKM = zeros(r, K);
19
```

```
%% Execution
% Run r realisations with seeds 1,...,r and store the outputs
for i=1:r
    [E_K(i,:), E_SRK(i,:), E_RK(i,:), E_SKM(i,:), E_MSRK(i,:),...
        T_K(i,:), T_SRK(i,:), T_RK(i,:), T_SKM(i,:), T_MSRK(i,:)] = ...
        SingleTrial(m, n, K, i);
end

%% Plots
% Plot the outputs for r realisations. As the behaviour of convergence is
% exponential, we plot the logarithm of the errors.
lw = 1; % Set line width

figure('Name', 'Error at each iteration, 10 realisations')
hold on;
for i=1:r
    plot(1:K, log10(E_K(i,:)), 'k-', 'LineWidth', lw);
    plot(1:K, log10(E_SRK(i,:)), 'b-', 'LineWidth', lw);
    plot(1:K, log10(E_RK(i,:)), 'r-', 'LineWidth', lw);
    plot(1:K, log10(E_SKM(i,:)), 'm-', 'LineWidth', lw)
    plot(1:K, log10(E_MSRK(i,:)), 'g-.', 'LineWidth', lw);
end
xlabel('Iterations','interpreter','latex');
ylabel('$log_{10}($Error$)$','interpreter','latex');
legend({'Kaczmarz', 'Simple Randomized Kaczmarz', 'Randomized Kaczmarz',...
    'Sampling Kaczmarz-Motzkin', 'Modified Simple Randomized Kaczmarz'})


figure('Name', 'Error against computation time, 10 realisations')
hold on;
for i=1:r
    plot(cumsum(T_K(i,:)), log10(E_K(i,:)), 'k-', 'LineWidth', lw);
    plot(cumsum(T_SRK(i,:)), log10(E_SRK(i,:)), 'b-', 'LineWidth', lw);
    plot(cumsum(T_RK(i,:)), log10(E_RK(i,:)), 'r-', 'LineWidth', lw);
    plot(cumsum(T_SKM(i,:)), log10(E_SKM(i,:)), 'm-', 'LineWidth',lw);
    plot(cumsum(T_MSRK(i,:)), log10(E_MSRK(i,:)), 'g-.', 'LineWidth', lw);
end
xlabel('Time','interpreter','latex');
ylabel('$log_{10}($Error$)$','interpreter','latex');
legend({'Kaczmarz', 'Simple Randomized Kaczmarz', 'Randomized Kaczmarz',...
    'Sampling Kaczmarz-Motzkin', 'Modified Simple Randomized Kaczmarz'})
```

F Technical specifications

Processor	Intel Core i5-6200U at 2.4GHz base frequency
RAM	8GB DDR4 at 2133MHz
Operating System	Windows 10 Home, Build 19041.804
MATLAB Version	9.5.0.1033004 (R2018b) Update 2

Table 3: Specifications of the setup used to run the numerical experiments