Reading between the Hyperplanes

James A. Nguyen, Oleg Presnyakov, Adityakrishnan Radhakrishnan

June 2025

Abstract

Following the wake of Randomized Kaczmarz (RK), a row-sampling iterative projection method to solve large-scale systems $Ax = b$, several adaptations to the method have been produced to inspire faster convergence. Focusing on the case of ill-conditioned linear systems– systems on which RK algorithms typically struggle–, we highlight inter-row relationships that can be leveraged to inspire directionally aware projections. In particular, we find that improved convergence rates can sometimes be made by (i) projecting onto pairwise row differences, (ii) sampling from partitioned clusters of nearly orthogonal rows, or (iii) more frequently sampling spectrally-diverse rows.

1 Introduction

Amidst recent developments in randomized numerical linear algebra, randomized projection methods become more applicable as we seek to further improve them. Two prevalent examples of such methods include the Randomized Kaczmarz (RK) and the Sampling Kaczmarz-Motzkin (SKM) method. In particular, we investigate the consistent ill-conditioned case of overdetermined linear systems of the forms $Ax^* = b$ and $Ax \leq b$, where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $m \gg n$. Some of the considerable properties of such algorithms include their efficiency in iterative convergence, sample complexity, and approximation error. Most foundationally, we let x_k be the kth iterate of the algorithm and observe that the convergence of RK is at least linear in expectation:

$$\mathbb{E}(||\varepsilon_k||^2) \leq \left(1 - \frac{\sigma_{\min}^2(A)}{||A||_F^2}\right)^k ||\varepsilon_0||^2, \tag{1}$$

where ε_k is the kth error $x_k - x^*$. Note that the bound depends on the scaled conditioning of the matrix, so it is expected to perform worse on ill-conditioned matrices. Thus, we seek to improve the expected convergence by experimenting with different projection techniques, clustering approaches, and sampling distributions to address spectral instability.

The first problem we tackle is a binary classification problem that we transform into a linear feasibility problem using the Hadamard product. Our solution to this problem is founded upon recent developments in Randomized Kaczmarz methods. Specifically, we utilize an adaptation to the SKM algorithm for solving linear feasibility problems [3]. For the second problem we aimed to construct a coreset for efficient linear system solving for overdetermined matrices. Since the matrix is overdetermined, the feasible solution can be found without utilizing the full information of a matrix. We proposed an algorithm that

partitions the matrix into smaller clusters, which are then used to compute a solution to the original system. Lastly, we take after observations made by Steinerberger that emphasize the directional convergence of the error vector $\varepsilon_k = x_k - x^*$ along the least singular vectors of the system [4]. We suggest sampling techniques that increase the RK sampling frequency of rows with a greater absolute component in the direction of the least singular vector v_n of a system with singular value decomposition $A = U\Sigma V^\intercal$.

2 Pairwise Comparisons

We are interested in leveraging variations of the Randomized Kaczmarz algorithm to approach a binary classification problem. We consider a consistent system of inequalities $Ax^* \leq b$ for some Gaussian matrix A and an unknown solution vector x^* such that $b = sign(Ax^*)$. Using iterative methods, we begin with an initial vector x_0 and seek to converge to some iteration vector x_k that satisfies $sign(Ax^*) = sign(Ax_k)$.

2.1 Problem

We now redefine the binary classification problem as a linear feasibility problem using the Hadamard Product, i.e., the element-wise matrix product. We proceed by (re)defining the following:

- Matrix B to be the $m \times n$ matrix with the vector b as its columns.
- Matrix $A' := -B \odot A$ where $\odot$ represents the Hadamard product.
- Vector $b' = 0$.

We now want to solve the linear feasibility problem $A'x^* \leq b'$.
Recall that we originally wanted to find an x such that

$$\text{sign}(\langle A_i, x\rangle) = \text{sign}(\langle A_i, x^*\rangle) = b_i$$

This is equivalent to solving

$$\begin{aligned} & b_i * \text{sign}(\langle A_i, x\rangle) > 0 \\ \iff \quad & (B_i * A_i)x > 0 \\ \iff \quad & (-B \odot A)x < 0 \\ \iff \quad & A'x < b'. \end{aligned}$$

There are different matrices A that we can use. We first define the following:

- $G := m \times n$ matrix that is generated by an SVD where S is generated by the range of singular values σ_i^2 for $i = 1, ..., n$. U and V are generated by taking QR decompositions of random Gaussian matrices.
- $P := \binom{m}{2} \times n$ Pairwise Comparison matrix where a pairwise comparison (between rows) is defined to be $G_i - G_j$ for $i, j \in [n]$ and $i \neq j$.

We will compare the performance of SKM on the matrix G and the $(m + \binom{m}{2}) \times n$ matrix $G \cup P$.

2.2 Approach

We proceed by running numerical experiments using an adaptation of the SKM algorithm for solving linear feasibility problems [2] [3]. We also define x_0 to be some random initial iterate, β to be the number of rows sampled for SKM, λ to be a relaxation parameter, and K to be the total number of iterations to run. When we refer to A and b in the algorithm, we mean A' and b' as defined in the previous section. Note that the rows of A are normalized. Finally, z^+ denotes the positive entries of vector z with zeros elsewhere.

Algorithm 1 SKM for linear feasibility

procedure SKM(A, b, x_0, x^*, β, λ, K)
  $k = 1$
  while $k < K$
    Choose a sample of β rows, τ_k, from A.
    $t_k := \underset{i \in \tau_k}{\operatorname{argmax}} \, a_i^T x_{k-1} - b_i$
    $x_k = x_{k-1} - \lambda(a_{t_k}^T x_{k-1} - b_{t_k})^+ a_{t_k}$
    $k = k + 1$
  return x_k
end procedure

2.3 Results

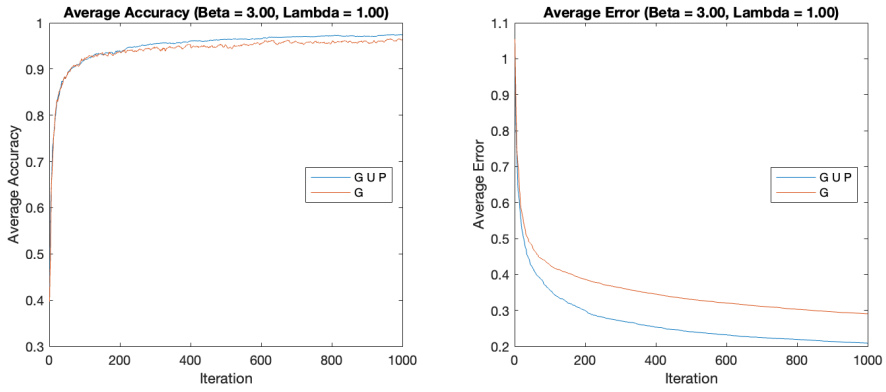

Figure 1: Accuracy vs Iterations (left) and Approximation Error vs. Iterations (right) for Pairwise Comparisons where $m = 240$ and $n = 12$. Experiment averaged over 20 trials.

We have tried to compare the results for different sizes of matrices as well as different types of matrices. Some observations that were made during the test (refer to Figure 1):

- The matrix with pairwise differences performs significantly better than the original matrix in terms of error and performs only slightly better than the original in terms of accuracy.

- The results showed that the algorithm has a good application when the number of rows is significantly larger than the number of columns. This gives our algorithm more room for projection that we can observe in the results. (As the number of rows increases and the number of columns stays stable, we expect the approximation error to improve. Cutting feasible region with more hyperplanes ⇒ making the feasible region smaller)
- We tested the algorithm for different λ (relaxation parameters) and β (the size of a sample) and found out that the best result were when $\lambda \geq 1$. For β it is naturally better to make the sample size bigger, so that instead of projecting just on a random row, we can choose the optimal projection. This supposition also coincides with the result, since with the increase of β the accuracy and error improve.

3 Clustering

A key question we pursued was how to extract a compact coreset from an over-determined linear system. We conjectured that any system of size $m \times n$ contains a coreset of cardinality $O(cn)$, where c is a small constant. By isolating this subset, existing algorithms could solve the reduced linear problem far more rapidly. Our intuition was to partition the matrix A into clusters of rows, enabling each cluster to be processed independently and thus lowering the overall computational cost. The next section explains the underlying idea and the algorithms that realize it.

3.1 Existence of a "Good" Cluster

To obtain a quick proof–of–concept that a small *inner-product coreset* can solve an over-determined linear system almost as accurately as the full data set, we ran the following procedure on a synthetic ill-conditioned matrix.

1. **Data generation.** For most of our tests we generated *ill-conditioned matrices*, which usually constitute the worst-case scenario for iterative algorithms. Consequently, insights gained on such matrices tend to generalize well to arbitrary random ones. To build each ill-conditioned matrix we employed an SVD-based procedure: two random orthogonal factors were sampled, and the singular values were manually assigned so that the ratio between the largest and the smallest was on the order of 10^7.
2. **Row scoring.** For every row a_i compute the absolute inner product $s_i = \left|\langle a_i,\, x^\star \rangle\right|$. Large scores indicate rows whose directions are highly aligned with $x^\star$.
3. **Coreset extraction.** Sort the scores and keep the $k = c\,n$ rows with the *smallest* absolute inner products (we use the heuristic factor $c = 2$). The retained rows form a sub-matrix B with the matching right-hand side b_B.
4. **Least-squares solves.**
   - Full system: $x_A = \arg\min_x \|A\,x - b\|_2$.
   - Coreset system: $x_B = \arg\min_x \|B\,x - b_B\|_2$.

   Both minimisers are obtained with `numpy.linalg.lstsq`.

5. **Evaluation.** Measure the relative error $\|x_B - x_A\|_2/\|x_A\|_2$ and compare residual norms $\|Ax_B - b\|_2$ and $\|Bx_B - b_B\|_2$.

3.2 Algorithms

In this section we will talk about all the algorithms we've been trying to utilize in order to achieve the best clustering. At the end of the section we will provide plots with all clustering methods to compare its result on a real data.

3.2.1 ϵ-cover net Algorithm

The first algorithm is based on an ϵ-cover net idea. For visualization, imagine each row as a vector in an n-dimensional vector space that connects the origin and the point defined by coefficients of a matrix. Then we are trying to understand how each row acts on the other vectors in our space with respect to an inner product. It is not hard to imagine that if points are close to each other, then they act similarly. As a result, we decided to use partition based on how far from each other points are. For details, refer to Algorithm **??**.

- Assign a_1 to cluster S_1.
- For $a_k, k > 1$, assign to the first cluster $(S_1, S_2, ...)$ where $\langle a_k, \frac{1}{p}\sum_{i=1}^{p} s_i\rangle < \varepsilon$, where p is the size of the chosen cluster and s_i are the rows in said cluster.
- If such a cluster does not exist, assign a_k to a new cluster.

After clustering, to start off the algorithm we project iterates onto one random row from each cluster. Then pick the "best" cluster, S^*, where the criterion for "best" is being the most orthogonal cluster to the current iterate (which is our best approximation for x^*), i.e., we let S^* be cluster S_i where

$$i := \underset{i}{\operatorname{argmin}} |\langle x_k, \frac{1}{p}\sum_{i=1}^{p} s_i\rangle|.$$

The algorithm's performance hinges on the chosen coverage radius ϵ. If ϵ is set too small relative to the spacing of the vectors, clusters become sparse and uninformative; if too large, nearly all rows collapse into a single cluster. Hence, our next objective is to identify an ϵ that yields a well-balanced matrix partition. In addition, because we track the mean projection in every cluster, we can dynamically reassign rows to different clusters once the current partition stops producing error reductions in further iterations.

3.2.2 Online cluster updating

Another idea is essentially taken from a well-known K-means algorithm. Since we know that the Karzmarz algorithm will eventually converge to a solution, we were trying to find an optimization that will be updated during the iterations. Each fixed number of iterations we were trying to reduce the matrix A that we were working with. Particularly, for an $m \times n$ matrix:

- Start with a cluster of the $2n$ best rows from the first active set which is the entire matrix of m rows.

- Every $(2^k * 100)$-th iteration $(k = 0, 1, ...)$, reduce the size of the active set by a factor of 2 $(\frac{m}{2}, \frac{m}{4}, ...)$ and stop at $4n$ rows.
- For each active set, we recluster and choose the $2n$ best rows based on the current iterate.

The "best" row is defined as the one most orthogonal to the current iterate:

$$i \ := \ \underset{i \in [m]}{\arg\min} \left(a_i^{\mathsf{T}} x_{k-1} \ - \ b_i \right).$$

With each iteration the rows we discard contribute less and less: their projections cease to reveal new information about the solution subspace. Although the exact pattern of elimination depends on the particular right-hand side, the sequence of discarded sets still carries useful structure. We can harness this history to pre-form clusters whenever the same matrix must be solved against multiple right-hand sides.

3.3 Results

We include two main ways to test the quality of partition. The first one is the condition testing. We hoped that our cluster would find a subset with a better conditioning than the random sample of the same size. Intuitively, partitioning

4 Singular Vector Analysis

4.1 Background

The latest avenue of our exploration emphasizes a direction-aware approach to RK algorithms. Whilst previous approaches focus on measures of orthogonality, we redirect that focus to analyzing the singular directions present in our system. Our analysis is inspired by recent developments revealing that the error vector converges along the system's smallest singular vector [4]. Steinerberger's analysis of the convergence of the error vector suggests that RK – under traditional sampling techniques – struggles to iterate in the directions least represented by the data.

We see that Figure 2 confirms Steinerberger's results from [4] and suggest the component of the error vector ε_k in the direction of the largest singular vectors converge to 0 most quickly.

4.2 Approach

To compensate for the under-representation of small singular vectors in the trajectory of our iterate x_k, we seek to adapt our sampling distribution to select with higher probability rows that have a greater component in the direction of the least singular vector. Let a_i be the ith row of $A \in \mathbb{R}^{m \times n}$. Then, a_i is some linear combination of the n singular vectors:

$$a_i = \sum_{j=1}^{n} c_j^{(i)} \vec{v_j} = c_1^{(i)} \vec{v_1} + \cdots c_n^{(i)} \vec{v_n}, \qquad c_j^{(i)} \in \mathbb{R}, \ i = 1, \ldots m. \tag{2}$$

where each $\vec{v}_j$ is the jth singular vector of A. In the context of our problem, it should not be assumed that knowledge of the singular vectors be known; otherwise, the solution could

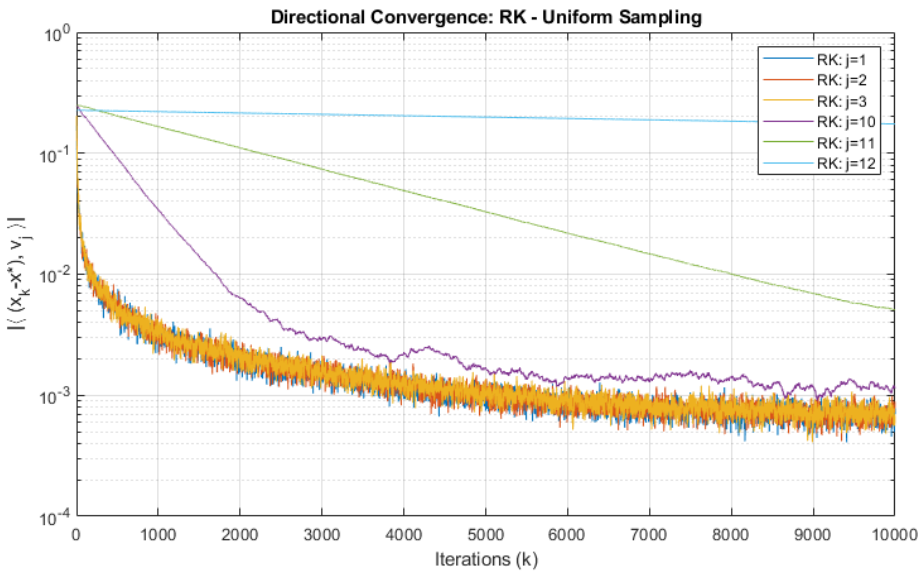

Figure 2: $A \in \mathbb{R}^{240\times 12}$ is constructed as described in Section 4.3. The figure demonstrates the convergence of $|\langle x_k - x^*, v_j\rangle|$ for each singular vector v_j $(j = 1, \ldots, n)$ at each iteration k. It demonstrates the ordinal spectrum of convergence by decreasing singular value.

be reconstructed and the problem would be trivial. However, we analyze the convergence using this knowledge to inspire a new algorithm. Considering that $A = U\Sigma V^\intercal$, we obtain each $c_n^{(i)}$ by:

$$c_n = [c_n^{(1)}, \ldots, c_n^{(m)}]^\intercal = AV_{:n} \tag{3}$$

With these nth spectral coefficients c_n, we construct spectral sampling weights ω_i for each row of the system:

$$\omega_i := \frac{|c_n^{(i)}|}{\sum_{l=1}^{m} |c_n^{(l)}|}, \qquad i = 1, \ldots, m. \tag{4}$$

By substituting the row norm-based sampling distribution by the one determined by the spectral weights from (4), we run a series of numerical experiments, expecting to improve upon the approximation error horizon encountered by the RK algorithm.

4.3 Numerical Experimentation

For all numerical experiments in this section, we refer to the following matrix construction: We produce $A \in \mathbb{R}^{240\times 12}$ by taking the QR-decompositions of the Gaussian matrices $\hat{U} \in \mathbb{R}^{240\times 240}$ and $\hat{V} \in \mathbb{R}^{12\times 12}$ to obtain orthogonal matrices $U \in \mathbb{R}^{240\times 240}$ and $V \in \mathbb{R}^{12\times 12}$. We then construct a matrix $\Sigma \in \mathbb{R}^{240\times 12}$ with entries $\Sigma_{jj} = e^{n-j+1}$, $(j = 1, \ldots, n)$ and 0 everywhere else such that we obtain $A = U\Sigma V^T$. To complete the system, we let $x^* = [0, 0, \ldots, 1]^\intercal$ be the length-n solution vector to $Ax^* = b$.

To quantify the progress of the iterate x_k in the direction of each singular vector, we follow the dot product of the error vector ε_k with each singular vector. We define the jth singular error at each iteration as:

$$\varepsilon_k^{(j)} := |\langle x - x^*, v_j \rangle|, \qquad j = 1, \ldots, n \tag{5}$$

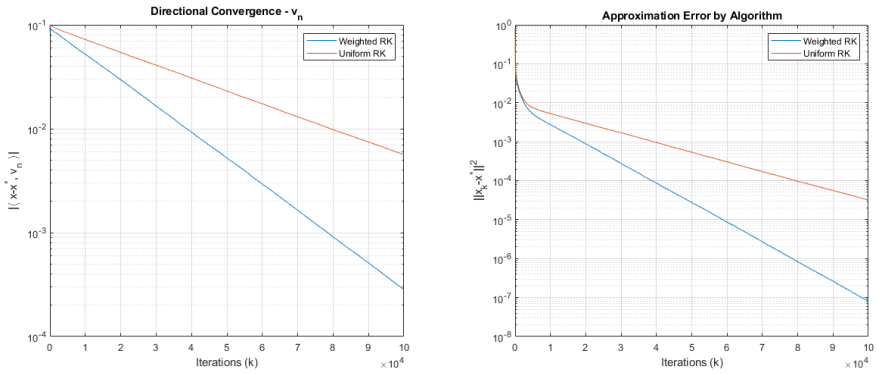

Figure 3: A matrix $A \in \mathbb{R}^{240\times 12}$ is constructed as described in Section 4.3. Plot data averages results over 50 different random matrices. The left plot describes the directional convergence $|\langle x_k - x^*, v_n \rangle|$ for each algorithm. The right plot demonstrates the improved convergence rate for weighted sampling.

We notice in Figure 3 that the modified sampling method inspired by the directional weighting scheme improves the convergence in the direction of the least singular vector. It does not improve the directional convergence in any of the other singular vectors, yet this is expected as we determine our sampling weights using only the coefficient c_n. This method effectively targets the restricting factor of (1) by sampling each vector with a probability proportional to the magnitude of its component in the direction of v_j. Figure 3 also reveals that this weighted sampling distribution, on average, improves the convergence rate of the algorithm. Further plotting would reveal that x_k converges to x^* within machine error ($\epsilon_{machine} = 1 \times 10^{-15}$) in less than half the number of iterations when using weighted sampling compared to uniform.

Although this method requires knowledge of the singular value decomposition of A, the trend it unveils should be used to inspire methods that improve existing RK algorithms for ill-conditioned systems. We propose some solutions that leverage these properties in Section 5.

5 Future Work

We remind the reader that the weighted sampling method proposed in Section 4 requires the knowledge of the SVD of the matrix beforehand. Since the process of obtaining the SVD is, itself, expensive and would render the problem trivial, we suggest approximating the singular vectors v_j of A iteratively. We suggest two possible approaches: (i) iteratively

adapt sampling weights as the singular vectors are approximated, or (ii) perform Average Block Kaczmarz (ABK) [1] with β many rows and adjust the projection weight of each row proportionally to its component $c_n^{(i)}$ relative the other rows'.

References

[1] K. Du, W.-T. Si, and X.-H. Sun, "Randomized extended average block kaczmarz for solving least squares," *SIAM Journal on Scientific Computing*, vol. 42, no. 6, pp. A3541–A3559, 2020. [Online]. Available: https://doi.org/10.1137/20M1312629

[2] J. Haddock and A. Ma, "Greed works: An improved analysis of sampling kaczmarz-motzkin," 2020. [Online]. Available: https://arxiv.org/abs/1912.03544

[3] J. D. Loera, J. Haddock, and D. Needell, "A sampling kaczmarz-motzkin algorithm for linear feasibility," 2019. [Online]. Available: https://arxiv.org/abs/1605.01418

[4] S. Steinerberger, "Randomized kaczmarz converges along small singular vectors," 2021. [Online]. Available: https://arxiv.org/abs/2006.16978