Coordinate-wise Power Method

Qi Lei [1] **Kai Zhong** [1] **Inderjit S. Dhillon** [1,2]
[1] Institute for Computational Engineering & Sciences [2] Department of Computer Science
University of Texas at Austin
{leiqi, zhongkai}@ices.utexas.edu, inderjit@cs.utexas.edu

Abstract

In this paper, we propose a coordinate-wise version of the power method from an optimization viewpoint. The vanilla power method simultaneously updates all the coordinates of the iterate, which is essential for its convergence analysis. However, different coordinates converge to the optimal value at different speeds. Our proposed algorithm, which we call coordinate-wise power method, is able to select and update the most *important* k coordinates in $O(kn)$ time at each iteration, where n is the dimension of the matrix and $k \leq n$ is the size of the active set. Inspired by the "greedy" nature of our method, we further propose a greedy coordinate descent algorithm applied on a non-convex objective function specialized for symmetric matrices. We provide convergence analyses for both methods. Experimental results on both synthetic and real data show that our methods achieve up to 23 times speedup over the basic power method. Meanwhile, due to their coordinate-wise nature, our methods are very suitable for the important case when data cannot fit into memory. Finally, we introduce how the coordinate-wise mechanism could be applied to other iterative methods that are used in machine learning.

1 Introduction

Computing the dominant eigenvectors of matrices and graphs is one of the most fundamental tasks in various machine learning problems, including low-rank approximation, principal component analysis, spectral clustering, dimensionality reduction and matrix completion. Several algorithms are known for computing the dominant eigenvectors, such as the power method, Lanczos algorithm [14], randomized SVD [2] and multi-scale method [17]. Among them, the power method is the oldest and simplest one, where a matrix A is multiplied by the normalized iterate $\boldsymbol{x}^{(l)}$ at each iteration, namely,

$$\boldsymbol{x}^{(l+1)} = \text{normalize}(A\boldsymbol{x}^{(l)}).$$

The power method is popular in practice due to its simplicity, small memory footprint and robustness, and particularly suitable for computing the dominant eigenvector of large sparse matrices [14]. It has applied to PageRank [7], sparse PCA [19, 9], private PCA [4] and spectral clustering [18]. However, its convergence rate depends on $|\lambda_2|/|\lambda_1|$, the ratio of magnitude of the top two dominant eigenvalues [14]. Note that when $|\lambda_2| \approx |\lambda_1|$, the power method converges slowly.

In this paper, we propose an improved power method, which we call coordinate-wise power method, to accelerate the vanilla power method. Vanilla power method updates all n coordinates of the iterate simultaneously even if some have already converged to the optimal. This motivates us to develop new algorithms where we select and update a set of important coordinates at each iteration. As updating each coordinate costs only $\frac{1}{n}$ of one power iteration, significant running time can be saved when n is very large. We raise two questions for designing such an algorithm.

The first question: how to select the coordinate? A natural idea is to select the coordinate that will change the most, namely,

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

$$\text{argmax}_i |c_i|, \text{ where } \boldsymbol{c} = \frac{A\boldsymbol{x}}{\boldsymbol{x}^T A\boldsymbol{x}} - \boldsymbol{x}, \tag{1}$$

where $\frac{A\boldsymbol{x}}{\boldsymbol{x}^T A\boldsymbol{x}}$ is a scaled version of the next iterate given by power method, and we will explain this special scaling factor in Section 2. Note that c_i denotes the i-th element of the vector $\boldsymbol{c}$. Instead of choosing only one coordinate to update, we can also choose k coordinates with the largest k changes in $\{|c_i|\}_{i=1}^n$. We will justify this selection criterion by connecting our method with greedy coordinate descent algorithm for minimizing a non-convex function in Section 3. With this selection rule, we are able to show that our method has global convergence guarantees and faster convergence rate compared to vanilla power method if k satisfies certain conditions.

Another key question: how to choose these coordinates without too much overhead? How to efficiently select important elements to update is of great interest in the optimization community. For example, [1] leveraged nearest neighbor search for greedy coordinate selection, while [11] applied partially biased sampling for stochastic gradient descent. To calculate the changes in Eq (1) we need to know all coordinates of the next iterate. This violates our previous intention to calculate a small subset of the new coordinates. We show, by a simple trick, we can use only $O(kn)$ operations to update the most important k coordinates. Experimental results on dense as well as sparse matrices show that our method is up to 8 times faster than vanilla power method.

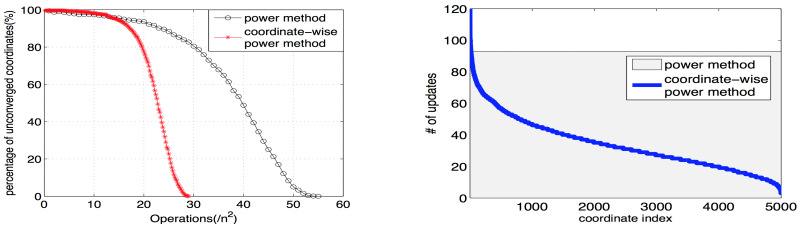
Relation to optimization. Our method reminds us of greedy coordinate descent method. Indeed, we show for symmetric matrices our coordinate-wise power method is similar to greedy coordinate descent for rank-1 matrix approximation, whose variants are widely used in matrix completion [8] and non-negative matrix factorization [6]. Based on this interpretation, we further propose a faster greedy coordinate descent method specialized for symmetric matrices. This method achieves up to 23 times speedup over the basic power method and 3 times speedup over the *Lanczos* method on large real graphs. For this non-convex problem, we also provide convergence guarantees when the initial iterate lies in the neighborhood of the optimal solution.

Extensions. With the coordinate-wise nature, our methods are very suitable to deal with the case when data cannot fit into memory. We can choose a k such that k rows of A can fit in memory, and then fully process those k rows of data before loading the RAM (random access memory) with a new partition of the matrix. This strategy helps balance the data processing and data loading time. The experimental results show our method is 8 times faster than vanilla power method for this case.

The paper is organized as follows. Section 2 introduces coordinate-wise power method for computing the dominant eigenvector. Section 3 interprets our strategy from an optimization perspective and proposes a faster algorithm. Section 4 provides theoretical convergence guarantee for both algorithms. Experimental results on synthetic or real data are shown in Section 5. Finally Section 6 presents the extensions of our methods: dealing with out-of-core cases and generalizing the coordinate-wise mechanism to other iterative methods that are useful for the machine learning community.

2 Coordinate-wise Power Method

The classical power method (PM) iteratively multiplies the iterate $\boldsymbol{x} \in \mathbb{R}^n$ by the matrix $A \in \mathbb{R}^{n \times n}$, which is inefficient since some coordinates may converge faster than others. To illustrate this



(a) The percentage of unconverged coordinates versus the number of operations

(b) Number of updates of each coordinate

Figure 1: **Motivation for the Coordinate-wise Power Method.** Figure 1(a) shows how the percentage of unconverged coordinates decreases with the number of operations. The gradual decrease demonstrates the unevenness of each coordinate as the iterate converges to the dominant eigenvector. In Figure 1(b), the X-axis is the coordinate indices of iterate $\boldsymbol{x}$ sorted by their frequency of updates, which is shown on the Y-axis. The area below each curve approximately equals the total number of operations.The given matrix is synthetic with $|\lambda_2|/|\lambda_1| = 0.5$, and terminating accuracy ϵ is set to be 1e-5.

phenomenon, we conduct an experiment with the power method; we set the stopping criterion as $\|\boldsymbol{x} - \boldsymbol{v}_1\|_\infty < \epsilon$, where ϵ is the threshold for error, and let $\boldsymbol{v}_i$ denote the i-th dominant eigenvector (associated with the eigenvalue of the i-th largest magnitude) of A in this paper. During the iterative process, even if some coordinates meet the stopping criterion, they still have to be updated at every iteration until uniform convergence. In Figure 1(a), we count the number of unconverged coordinates, which we define as $\{i : i \in [n] \big| |x_i - v_{1,i}| > \epsilon\}$, and see it gradually decreases with the iterations, which implies that the power method makes a large number of unnecessary updates. In this paper, for computing the dominant eigenvector, we exhibit a coordinate selection scheme that has the ability to select and update "important" coordinates with little overhead. We call our method *Coordinate-wise Power Method* (CPM). As shown in Figure 1(a) and 1(b), by selecting important entries to update, the number of unconverged coordinates drops much faster, leading to an overall fewer flops.

Algorithm 1 Coordinate-wise Power Method

1: **Input:** Symmetric matrix $A \in \mathbb{R}^{n\times n}$, number of selected coordinates k, and number of iterations, L.

2: Initialize $\boldsymbol{x}^{(0)} \in \mathbb{R}^n$ and set $\boldsymbol{z}^{(0)} = A\boldsymbol{x}^{(0)}$. Set coordinate selecting criterion $\boldsymbol{c}^{(0)} = \boldsymbol{x}^{(0)} - \frac{\boldsymbol{z}^{(0)}}{(\boldsymbol{x}^{(0)})^T\boldsymbol{z}^{(0)}}$.

3: **for** $l = 1$ **to** L **do**

4: Let $\Omega^{(l)}$ be a set containing k coordinates of $\boldsymbol{c}^{(l-1)}$ with the largest magnitude. Execute the following updates:

$$y_j^{(l)} = \begin{cases} \frac{z_j^{(l-1)}}{(\boldsymbol{x}^{(l-1)})^T\boldsymbol{z}^{(l-1)}}, & j \in \Omega^{(l)} \\ x_j^{(l-1)}, & j \notin \Omega^{(l)} \end{cases} \tag{2}$$

$$\boldsymbol{z}^{(l)} = \boldsymbol{z}^{(l-1)} + A(\boldsymbol{y}^{(l)}_{\Omega^{(l)}} - \boldsymbol{x}^{(l-1)}_{\Omega^{(l)}}) \tag{3}$$

$$\boldsymbol{z}^{(l)} = \boldsymbol{z}^{(l)}/\|\boldsymbol{y}^{(l)}\|, \;\; \boldsymbol{x}^{(l)} = \boldsymbol{y}^{(l)}/\|\boldsymbol{y}^{(l)}\|$$

$$\boldsymbol{c}^{(l)} = \boldsymbol{x}^{(l)} - \frac{\boldsymbol{z}^{(l)}}{(\boldsymbol{x}^{(l-1)})^T\boldsymbol{z}^{(l-1)}}$$

5: **Output:** Approximate dominant eigenvector $\boldsymbol{x}^{(L)}$

Algorithm 1 describes our coordinate-wise power method that updates k entries at a time for computing the dominant eigenvector for a symmetric input matrix, while a generalization to asymmetric cases is straightforward. The algorithm starts from an initial vector $\boldsymbol{x}^{(0)}$, and iteratively performs updates $x_i \leftarrow \boldsymbol{a}_i^T\boldsymbol{x}/\boldsymbol{x}^TA\boldsymbol{x}$ with i in a selected set of coordinates $\Omega \subseteq [n]$ defined in step 4, where $\boldsymbol{a}_i$ is the i-th row of A. The set of indices Ω is chosen to maximize the difference between the current coordinate value x_i and the next coordinate value $\boldsymbol{a}_i^T\boldsymbol{x}/\boldsymbol{x}^TA\boldsymbol{x}$. $\boldsymbol{z}^{(l)}$ and $\boldsymbol{c}^{(l)}$ are auxiliary vectors. Maintaining $\boldsymbol{z}^{(l)} \equiv A\boldsymbol{x}^{(l)}$ saves much time, while the magnitude of $\boldsymbol{c}$ represents importance of each coordinate and is used to select Ω.

We use the Rayleigh Quotient $\boldsymbol{x}^TA\boldsymbol{x}$ ($\boldsymbol{x}$ is normalized) for scaling, different from $\|A\boldsymbol{x}\|$ in the power method. Our intuition is as follows: on one hand, it is well known that Rayleigh Quotient is the best estimate for eigenvalues. On the other hand, the limit point using $\boldsymbol{x}^TA\boldsymbol{x}$ scaling will satisfy $\bar{\boldsymbol{x}} = A\bar{\boldsymbol{x}}/\bar{\boldsymbol{x}}^TA\bar{\boldsymbol{x}}$, which allows both negative or positive dominant eigenvectors, while the scaling $\|A\boldsymbol{x}\|$ is always positive, so its limit point only lies in the eigenvectors associated with positive eigenvalues, which rules out the possibility of converging to the negative dominant eigenvector.

2.1 Coordinate Selection Strategy

An initial understanding for our coordinate selection strategy is that we select coordinates with the largest potential change. With a current iterate $\boldsymbol{x}$ and an arbitrary active set Ω, let $\boldsymbol{y}^\Omega$ be a potential next iterate with only coordinates in Ω updated, namely,

$$(\boldsymbol{y}^\Omega)_i = \begin{cases} \frac{\boldsymbol{a}_i^T\boldsymbol{x}}{\boldsymbol{x}^TA\boldsymbol{x}}, & i \in \Omega \\ x_i, & i \notin \Omega \end{cases}$$

According to our algorithm, we select active set Ω to maximize the iterate change. Therefore:

$$\Omega = \underset{I\subset[n],|I|=k}{\arg\max} \left\{ \left\| (\boldsymbol{x} - \frac{A\boldsymbol{x}}{\boldsymbol{x}^TA\boldsymbol{x}})_I \right\|^2 = \|\boldsymbol{y}^I - \boldsymbol{x}\|^2 \right\} = \underset{I\subset[n],|I|=k}{\arg\min} \left\{ \left\| \frac{A\boldsymbol{x}}{\boldsymbol{x}^TA\boldsymbol{x}} - \boldsymbol{y}^I \right\|^2 \overset{\text{def}}{=} \|\boldsymbol{g}\|^2 \right\}$$

This is to say, with our updating rule, our goal of maximizing iteration gap is equivalent to minimizing the difference between the next iterate $\boldsymbol{y}^{(l+1)}$ and $A\boldsymbol{x}^{(l)}/(\boldsymbol{x}^{(l)})^TA\boldsymbol{x}^{(l)}$, where this difference could be interpreted as noise $\boldsymbol{g}^{(l)}$. A good set Ω ensures a sufficiently small noise $\boldsymbol{g}^{(l)}$, thus achieving a

similar convergence rate in $O(kn)$ time (analyzed later) as the power method does in $O(n^2)$ time. More formal statement for the convergence analysis is given in Section 4.

Another reason for this selection rule is that it incurs little overhead. For each iteration, we maintain a vector $z \equiv Ax$ with kn flops by the updating rule in Eq.(3). And the overhead consists of calculating $\boldsymbol{c}$ and choosing Ω. Both parts cost $O(n)$ operations. Here Ω is chosen by Hoare's quick selection algorithm [5] to find the k^{th} largest entry in $|\boldsymbol{c}|$. Thus the overhead is negligible compared with $O(kn)$. Thus CPM spends as much time on each coordinate as PM does on average, while those updated k coordinates are most important. For sparse matrices, the time complexity is $O(n + \frac{k}{n}\text{nnz}(A))$ for each iteration, where $\text{nnz}(A)$ is the number of nonzero elements in matrix A.

Although the above analysis gives us a good intuition on how our method works, it doesn't directly show that our coordinate selection strategy has any optimal properties. In next section, we give another interpretation of our coordinate-wise power method and establish its connection with the optimization problem for low-rank approximation.

3 Optimization Interpretation

The coordinate descent method [12, 6] was popularized due to its simplicity and good performance. With all but one coordinates fixed, the minimization of the objective function becomes a sequence of subproblems with univariate minimization. When such subproblems are quickly solvable, coordinate descent methods can be efficient. Moreover, in different problem settings, a specific coordinate selecting rule in each iteration makes it possible to further improve the algorithm's efficiency.

The power method reminds us of the rank-one matrix factorization

$$\underset{\boldsymbol{x}\in\mathbb{R}^n, \boldsymbol{y}\in\mathbb{R}^d}{\arg\min} \left\{ f(\boldsymbol{x}, \boldsymbol{y}) = \|A - \boldsymbol{x}\boldsymbol{y}^T\|_F^2 \right\} \tag{4}$$

With alternating minimization, the update for $\boldsymbol{x}$ becomes $\boldsymbol{x} \leftarrow \frac{A\boldsymbol{y}}{\|\boldsymbol{y}\|^2}$ and vice versa for $\boldsymbol{y}$. Therefore for symmetric matrix, alternating minimization is exactly PM apart from the normalization constant.

Meanwhile, the above similarity between PM and alternating minimization extends to the similarity between CPM and greedy coordinate descent. A more detailed interpretation is in Appendix A.5, where we show the equivalence in the following coordinate selecting rules for Eq.(4): **(a)** largest coordinate value change, denoted as $|\delta x_i|$; **(b)** largest partial gradient (Gauss-Southwell rule), $|\nabla_i f(\boldsymbol{x})|$; **(c)** largest function value decrease, $|f(\boldsymbol{x} + \delta x_i \boldsymbol{e}_i) - f(\boldsymbol{x})|$. Therefore, the coordinate selection rule is more formally testified in optimization viewpoint.

3.1 Symmetric Greedy Coordinate Descent (SGCD)

We propose an even faster algorithm based on greedy coordinate descent. This method is designed for symmetric matrices and additionally requires to know the sign of the most dominant eigenvalue. We also prove its convergence to the global optimum with a sufficiently close initial point.

A natural alternative objective function specifically for the symmetric case would be

$$\underset{\boldsymbol{x}\in\mathbb{R}^n}{\arg\min} \left\{ f(\boldsymbol{x}) = \|A - \boldsymbol{x}\boldsymbol{x}^T\|_F^2 \right\}. \tag{5}$$

Notice that the stationary points of $f(\boldsymbol{x})$, which require $\nabla f(\boldsymbol{x}) = 4(\|\boldsymbol{x}\|^2\boldsymbol{x} - A\boldsymbol{x}) = 0$, are obtained at eigenvectors: $\boldsymbol{x}_i^* = \sqrt{\lambda_i}\boldsymbol{v}_i$, if the eigenvalue λ_i is positive. The global minimum for Eq. (5) is the eigenvector corresponding to the largest positive eigenvalue, not the one with the largest magnitude. For most applications like PageRank we know λ_1 is positive, but if we want to calculate the negative eigenvalue with the largest magnitude, just optimize on $f = \|A + \boldsymbol{x}\boldsymbol{x}^T\|_F^2$ instead.

Now we introduce Algorithm 2 that optimizes Eq. (5). With coordinate descent, we update the i-th coordinate by $x_i^{(l+1)} \leftarrow \arg\min_\alpha f(\boldsymbol{x}^{(l)} + (\alpha - x_i^{(l)})\boldsymbol{e}_i)$, which requires the partial derivative of $f(\boldsymbol{x})$ in i-th coordinate to be zero, i.e.,

$$\nabla_i f(\boldsymbol{x}) = 4(x_i\|\boldsymbol{x}\|_2^2 - \boldsymbol{a}_i^T\boldsymbol{x}) = 0. \tag{6}$$

$$\iff \quad x_i^3 + px_i + q = 0, \text{ where } p = \|\boldsymbol{x}\|^2 - x_i^2 - a_{ii}, \text{ and } q = -\boldsymbol{a}_i^T\boldsymbol{x} + a_{ii}x_i \tag{7}$$

Similar to CPM, the most time consuming part comes from maintaining $\boldsymbol{z}$ ($\equiv A\boldsymbol{x}$), as the calculation for selecting the criterion $\boldsymbol{c}$ and the coefficient q requires it. Therefore the overall time complexity for one iteration is the same as CPM.

Notice that $\boldsymbol{c}$ from Eq.(6) is the partial gradient of f, so we are using the Gauss-Southwell rule to choose the active set. And it is actually the only effective and computationally cheap selection rule among previously analyzed rules **(a), (b)** or **(c)**. For calculating the iterate change $|\delta x_i|$, one needs to obtain roots for n equations. Likewise, the function decrease $|\Delta f_i|$ requires even more work.

Remark: for an unbiased initializer, $\boldsymbol{x}^{(0)}$ should be scaled by a constant α such that

$$\alpha = \arg\min_{a \geq 0} \|A - (a\boldsymbol{x}^{(0)})(a\boldsymbol{x}^{(0)})^T\|_F = \sqrt{\frac{(\boldsymbol{x}^{(0)})^T A \boldsymbol{x}^{(0)}}{\|\boldsymbol{x}^{(0)}\|^4}}$$

Algorithm 2 Symmetric greedy coordinate descent (SGCD)

1: **Input:** Symmetric matrix $A \in \mathbb{R}^{n \times n}$, number of selected coordinate, k, and number of iterations, L.
2: Initialize $\boldsymbol{x}^{(0)} \in \mathbb{R}^n$ and set $\boldsymbol{z}^{(0)} = A\boldsymbol{x}^{(0)}$. Set coordinate selecting criterion $\boldsymbol{c}^{(0)} = \boldsymbol{x}^{(0)} - \frac{\boldsymbol{z}^{(0)}}{\|\boldsymbol{x}^{(0)}\|^2}$.
3: **for** $l = 0$ **to** $L - 1$ **do**
4: Let $\Omega^{(l)}$ be a set containing k coordinates of $\boldsymbol{c}^{(l)}$ with the largest magnitude. Execute the following updates:

$$\begin{aligned} x_j^{(l+1)} &= \begin{cases} \arg\min_\alpha f\left(\boldsymbol{x}^{(l)} + (\alpha - x_j^{(l)})\boldsymbol{e}_j\right), & \text{if } j \in \Omega^{(l)}, \\ x_j^{(l)}, & \text{if } j \notin \Omega^{(l)}. \end{cases} \\ \boldsymbol{z}^{(l+1)} &= \boldsymbol{z}^{(l)} + A(\boldsymbol{x}_{\Omega^{(l)}}^{(l+1)} - \boldsymbol{x}_{\Omega^{(l)}}^{(l)}) \\ \boldsymbol{c}^{(l+1)} &= \boldsymbol{x}^{(l+1)} - \frac{\boldsymbol{z}^{(l+1)}}{\|\boldsymbol{x}^{(l+1)}\|^2} \end{aligned}$$

5: **Output:** vector $x^{(L)}$

4 Convergence Analysis

In the previous section, we propose coordinate-wise power method (CPM) and symmetric greedy coordinate descent (SGCD) on a non-convex function for computing the dominant eigenvector. However, it remains an open problem to prove convergence of coordinate descent methods for general non-convex functions. In this section, we show that both CPM and SGCD converge to the dominant eigenvector under some assumptions.

4.1 Convergence of Coordinate-wise Power Method

Consider a positive semidefinite matrix A, and let $\boldsymbol{v}_1$ be its leading eigenvector. For any sequence $(\boldsymbol{x}^{(0)}, \boldsymbol{x}^{(1)}, \cdots)$ generated by Algorithm 1, let $\theta^{(l)}$ to be the angle between vector $\boldsymbol{x}^{(l)}$ and $\boldsymbol{v}_1$, and $\phi^{(l)}(k) \stackrel{\text{def}}{=} \min_{|\Omega|=k} \sqrt{\sum_{i \notin \Omega} (c_i^{(l)})^2}/\|\boldsymbol{c}^{(l)}\|_2 = \|\boldsymbol{g}^{(l)}\|/\|\boldsymbol{c}^{(l)}\|$. The following lemma illustrates convergence of the tangent of $\theta^{(l)}$.

Lemma 4.1. *Suppose k is large enough such that*

$$\phi^{(l)}(k) < \frac{\lambda_1 - \lambda_2}{(1 + \tan\theta^{(l)})\lambda_1}. \tag{8}$$

Then

$$\tan\theta^{(l+1)} \leq \tan\theta^{(l)}\left(\frac{\lambda_2}{\lambda_1} + \frac{\phi^{(l)}(k))}{\cos\theta^{(l)}} < \tan\theta^{(l)}\right) \tag{9}$$

With the aid of Lemma 4.1, we show the following iteration complexity:

Theorem 4.2. *For any sequence $(\boldsymbol{x}^{(0)}, \boldsymbol{x}^{(1)}, \cdots)$ generated by Algorithm 1 with k satisfying $\phi^{(l)}(k) < \frac{\lambda_1 - \lambda_2}{2\lambda_1(1+\tan\theta^{(l)})}$, if $\boldsymbol{x}^{(0)}$ is not orthogonal to $\boldsymbol{v}_1$, then after $T = O(\frac{\lambda_1}{\lambda_1 - \lambda_2}\log(\frac{\tan\theta^{(0)}}{\varepsilon}))$ iterations we have $\tan\theta^{(T)} \leq \varepsilon$.*

The iteration complexity shown is the same as the power method, but since it requires less operations ($O(k\text{nnz}(A)/n)$ instead of $O(\text{nnz}(A))$ per iteration, we have

Corollary 4.2.1. *If the requirements in Theorem 4.2 apply and additionally k satisfies:*

$$k < n\log((\lambda_1 + \lambda_2)/(2\lambda_1))/\log(\lambda_2/\lambda_1), \tag{10}$$

CPM has a better convergence rate than PM in terms of the number of equivalent passes over the coordinates.

The RHS of (10) ranges from $0.06n$ to $0.5n$ when $\frac{\lambda_2}{\lambda_1}$ goes from 10^{-5} to $1-10^{-5}$. Meanwhile, experiments show that the performance of our algorithms isn't too sensitive to the choice of k. Figure 6 in Appendix A.6 illustrates that a sufficiently large range of k guarantees good performances. Thus we use a prescribed $k = \frac{n}{20}$ throughout our experiments in this paper, which saves the burden of tuning parameters and is a theoretically and experimentally favorable choice.

Part of the proof is inspired by the noisy power method [3] in that we consider the unchanged part $\boldsymbol{g}$ as noise. For the sake of a neat proof we require our target matrix to be positive semidefinite, although experimentally a generalization to regular matrices is also valid for our algorithm. Details can be found in Appendix A.1 and A.3.

4.2 Local Convergence for Optimization on $\|A - \boldsymbol{x}\boldsymbol{x}^T\|_F^2$

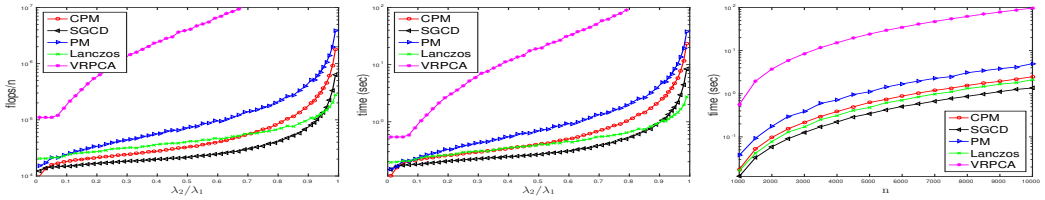
As the objective in Problem (5) is non-convex, it is hard to show global convergence. Clearly, with exact coordinate descent, Algorithm 2 will converge to some stationary point. In the following, we show that Algorithm 2 converges to the global minimum with a starting point sufficiently close to it.

Theorem 4.3. *(Local Linear Convergence) For any sequence of iterates $(\boldsymbol{x}^{(0)}, \boldsymbol{x}^{(1)}, \cdots)$ generated by Algorithm 2, assume the starting point $\boldsymbol{x}^{(0)}$ is in a ball centered by $\sqrt{\lambda_1}\boldsymbol{v}_1$ with radius $r = O(\frac{\lambda_1-\lambda_2}{\sqrt{\lambda_1}})$, or formally, $\boldsymbol{x}^{(0)} \in B_r(\sqrt{\lambda_1}\boldsymbol{v}_1)$, then $(\boldsymbol{x}^0, \boldsymbol{x}^1, \cdots)$ converges to the optima linearly.*

Specifically, when $k = 1$, then after $T = \frac{14\lambda_1 - 2\lambda_2 + 4\max_i |a_{ii}|}{\mu} \log \frac{f(\boldsymbol{x}^{(0)}) - f^*}{\varepsilon}$ iterations, we have $f(\boldsymbol{x}^{(T)}) - f^* \leq \varepsilon$, where $f^* = f(\sqrt{\lambda_1}\boldsymbol{v}_1)$ is the global minimum of the objective function f, and $\mu = \inf_{x,y \in B_r(\sqrt{\lambda_1}\boldsymbol{v}_1)} \frac{\|\nabla f(x) - \nabla f(y)\|_\infty}{\|x-y\|_1} \in [\frac{3(\lambda_1-\lambda_2)}{n}, 3(\lambda_1 - \lambda_2)]$.

We prove this by showing that the objective (5) is strongly convex and coordinate-wise Lipschitz continuous in a neighborhood of the optimum. The proof is given in Appendix A.4.

Remark: For real-life graphs, the diagonal values $a_{ii} = 0$, and the coefficient in the iteration complexity could be simplified as $\frac{14\lambda_1 - 2\lambda_2}{\mu}$ when $k = 1$.



(a) Convergence flops vs $\frac{\lambda_2}{\lambda_1}$ (b) Convergence time vs $\frac{\lambda_2}{\lambda_1}$ (c) Convergence time vs dimension

Figure 2: **Matrix properties affecting performance.** Figure 2(a), 2(b) show the performance of five methods with $\frac{\lambda_2}{\lambda_1}$ ranging from 0.01 to 0.99 and fixed matrix size $n = 5000$. In Figure 2(a) the measurement is FLOPs while in Figure 2(b) Y-axis is CPU time. Figure 2(c) shows how the convergence time varies with the dimension when fixing $\frac{\lambda_2}{\lambda_1} = 2/3$. In all figures Y-axis is in log scale for better observation. Results are averaged over from 20 runs.

5 Experiments

In this section, we compared our algorithms with PM, Lanczos method [14], and VRPCA [16] on dense as well as sparse dataset. All the experiments were executed on Intel(R) Xeon(R) E5430 machine with 16G RAM and Linux OS. We implement all the five algorithms in C++ with Eigen library.

5.1 Comparison on Dense and Simulated Dataset

We compare PM with our CPM and SGCD methods to show how coordinate-wise mechanism improves the original method. Further, we compared with a state-of-the-art algorithm *Lanczos* method. Besides, we also include a recent proposed stochastic SVD algorithm, *VRPCA*, that enjoys exponential convergence rate and shows similar insight in viewing the data in a separable way.

With dense and synthetic matrices, we are able to test the condition that our methods are preferable, and how the properties of the matrix, like λ_2/λ_1 or the dimension, affect the performance. For each algorithm, we start from the same random vector, and set stopping condition to be $\cos\theta \geq 1 - \epsilon, \epsilon = 10^{-6}$, where θ is the angle between the current iterate and the dominant eigenvector.

First we compare the performances with number of FLOPs (Floating Point Operations), which could better illustrate how greediness affects the algorithm's efficiency. From Figure 2(a) we can see our method shows much better performance than PM, especially when $\lambda_2/\lambda_1 \to 1$, where CPM and SGCD respectively achieve more than 2 and 3 times faster than PM. Figure 2(b) shows running time using five methods under different eigenvalue ratios λ_2/λ_1. We can see that only in some extreme cases when PM converges in less than 0.1 second, PM is comparable to our methods. In Figure 2(c) the testing factor is the dimension, which shows the performance is independent of the size of n. Meanwhile, in most cases, SGCD is better than Lanczos method. And although VRPCA has better convergence rate, it requires at least $10n^2$ operations for one data pass. Therefore in real applications, it is not even comparable to PM.

5.2 Comparison on Sparse and Real Dataset

Table 1: Six datasets and the performance of three methods on them.

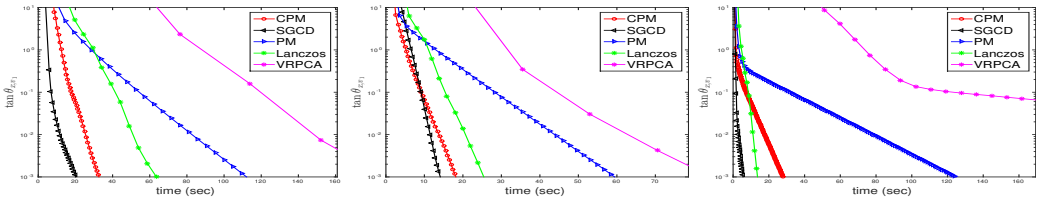
Dataset	n	nnz(A)	nnz/n	$\frac{\lambda_2}{\lambda_1}$	Time (sec)				
					PM	CPM	SGCD	Lanczos	VRPCA
com-Orkut	3.07M	234M	76.3	0.71	109.6	31.5	**19.3**	63.6	189.7
soc-LiveJournal	4.85M	86M	17.8	0.78	58.5	17.9	**13.7**	25.8	88.1
soc-Pokec	1.63M	44M	27.3	0.95	118	26.5	**5.2**	14.2	596.2
web-Stanford	282K	3.99M	14.1	0.95	8.15	1.05	**0.54**	0.69	7.55
ego-Gplus	108K	30.5M	283	0.51	0.99	**0.57**	0.61	1.01	5.06
ego-Twitter	81.3K	2.68M	33	0.65	0.31	0.15	**0.11**	0.19	0.98

To test the scalability of our methods, we further test and compare our methods on large and sparse datasets. We use the following real datasets:

1) com-Orkut: Orkut online social network
2) soc-LiveJournal: On-line community for maintaining journals, individual and group blogs
3) soc-Pokec: Pokec, most popular on-line social network in Slovakia
4) web-Stanford: Pages from Stanford University (stanford.edu) and hyperlinks between them
5) ego-Gplus (Google+): Social circles from Google+
6) ego-Twitter: Social circles from Twitter

The statistics of the datasets are summarized in Table 1, which includes the essential properties of the datasets that affect the performances and the average CPU time for reaching $\cos\theta_{\boldsymbol{x},\boldsymbol{v}_1} \geq 1 - 10^{-6}$. Figure 3 shows $\tan\theta_{\boldsymbol{x},\boldsymbol{v}_1}$ against the CPU time for the four methods with multiple datasets.

From the statistics in Table 1 we can see that in all the cases, either CPM or SGCD performs the best. CPM is roughly 2-8 times faster than PM, while SGCD reaches up to 23 times and 3 times faster than PM and Lanczos method respectively. Our methods show their privilege in the soc-Pokec(3(c)) and web-Stanford(3(d)), the most ill-conditioned cases ($\lambda_2/\lambda_1 \approx 0.95$), achieving 15 or 23 times of speedup on PM with SGCD. Meanwhile, when the condition number of the datasets is not too small (see 3(a),3(b),3(e),3(f)), both CPM and SGCD outperform PM as well as Lanczos method. And



(a) Performance on com-Orkut (b) Performance on LiveJournal (c) Performance on soc-Pokec

(d) Performance on web-Stanford (e) Performance on Google+ (f) Performance on ego-Twitter

Figure 3: **Time comparison for sparse dataset.** X-axis shows the CPU time while Y-axis is log scaled $\tan\theta$ between $\boldsymbol{x}$ and $\boldsymbol{v}_1$. The empirical performance shows all three methods have linear convergence.

similar to the reasoning in the dense case, although VRPCA requires less iterations for convergence, the overall CPU time is much longer than others in practice.

In summary of performances on both dense and sparse datasets, SGCD is the fastest among others.

6 Other Application and Extensions

6.1 Comparison on Out-of-core Real Dataset

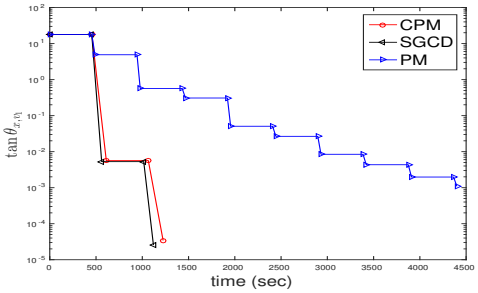


Figure 4: **A pseudograph for time comparison of out-of-core dataset from Twitter.** Each "staircase" illustrates the performance of one data pass. The flat part indicates the stage of loading data, while the downward part shows the phase of processing data. As we only updated auxiliary vectors instead of the iterate every time we load part of the matrix, we could not test performances until a whole data pass. Therefore for the sake of clear observation, we group together the loading phase and the processing phase in each data pass.

An important application for coordinate-wise power method is the case when data can not fit into memory. Existing methods can't be easily applied to out-of-core dataset. Most existing methods don't indicate how we can update part of the coordinates multiple times and fully reuse part of the matrix corresponding to those active coordinates. Therefore the data loading and data processing time are highly unbalanced. A naive way of using PM would be repetitively loading part of the matrix from the disk and calculating that part of matrix-vector multiplication. But from Figure 4 we can see reading from the disk costs much more time than the process of computation, therefore we will waste a lot of time if we cannot fully use the data before dumping it. For CPM, as we showed in Theorem 4.1 that updating only k coordinates of iterate $\boldsymbol{x}$ may still enhance the target direction, we could do matrix vector multiplication multiple times after one single loading. As with SGCD, optimization on part of x for several times will also decrease the function value.

We did experiments on the dataset from Twitter [10] using out-of-core version of the three algorithms shown in Algorithm 3 in Appendix A.7. The data, which contains 41.7 million user profiles and 1.47 billion social relations, is originally 25.6 GB and then separated into 5 files. In Figure 4, we can see that after data pass, our methods can already reach rather high precision, which compresses hours of processing time to 8 minutes.

6.2 Extension to other linear algebraic methods

With the interpretation in optimization, we could apply a coordinate-wise mechanism to PM and get good performance. Meanwhile, for some other iterative methods in linear algebra, if the connection to optimization is valid, or if the update is separable for each coordinate, the coordinate-wise mechanism may also be applicable, like Jacobi method.

For diagonal dominant matrices, Jacobi iteration [15] is a classical method for solving linear system $Ax = b$ with linear convergence rate. The iteration procedure is:

Initialize: $A \to D + R$, where D =Diag(A), and $R = A - D$.
Iterations: $x^+ \leftarrow D^{-1}(b - Rx)$.

This method is similar to the vanilla power method, which includes a matrix vector multiplication $-Rx$ with an extra translation b and a normalization step D^{-1}. Therefore, a potential similar realization of greedy coordinate-wise mechanism is also applicable here. See Appendix A.8 for more experiments and analyses, where we also specify its relation to Gauss-Seidel iteration [15].

7 Conclusion

In summary, we propose a new coordinate-wise power method and greedy coordinate descent method for computing the most dominant eigenvector of a matrix. This problem is critical to many applications in machine learning. Our methods have convergence guarantees and achieve up to 23 times of speedup on both real and synthetic data, as compared to the vanilla power method.

Acknowledgements

This research was supported by NSF grants CCF-1320746, IIS-1546452 and CCF-1564000.

References

[1] Inderjit S Dhillon, Pradeep K Ravikumar, and Ambuj Tewari. Nearest neighbor based greedy coordinate descent. In *Advances in Neural Information Processing Systems*, pages 2160–2168, 2011.

[2] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

[3] Moritz Hardt and Eric Price. The noisy power method: A meta algorithm with applications. In *Advances in Neural Information Processing Systems*, pages 2861–2869, 2014.

[4] Moritz Hardt and Aaron Roth. Beyond worst-case analysis in private singular vector computation. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 331–340. ACM, 2013.

[5] Charles AR Hoare. Algorithm 65: find. *Communications of the ACM*, 4(7):321–322, 1961.

[6] Cho-Jui Hsieh and Inderjit S Dhillon. Fast coordinate descent methods with variable selection for non-negative matrix factorization. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1064–1072. ACM, 2011.

[7] Ilse Ipsen and Rebecca M Wills. Analysis and computation of google's pagerank. In *7th IMACS international symposium on iterative methods in scientific computing, Fields Institute, Toronto, Canada*, volume 5, 2005.

[8] Prateek Jain, Praneeth Netrapalli, and Sujay Sanghavi. Low-rank matrix completion using alternating minimization. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 665–674. ACM, 2013.

[9] Michel Journée, Yurii Nesterov, Peter Richtárik, and Rodolphe Sepulchre. Generalized power method for sparse principal component analysis. *The Journal of Machine Learning Research*, 11:517–553, 2010.

[10] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is Twitter, a social network or a news media? *Proceedings of the 19th international conference on World wide web*, pages 591–600, 2010.

[11] Deanna Needell, Rachel Ward, and Nati Srebro. Stochastic gradient descent, weighted sampling, and the randomized Kaczmarz algorithm. In *Advances in Neural Information Processing Systems*, pages 1017–1025, 2014.

[12] Yu Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

[13] Julie Nutini, Mark Schmidt, Issam H Laradji, Michael Friedlander, and Hoyt Koepke. Coordinate descent converges faster with the Gauss-Southwell rule than random selection. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1632–1641, 2015.

[14] Beresford N Parlett. *The Symmetric Eigenvalue Problem*, volume 20. SIAM, 1998.

[15] Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[16] Ohad Shamir. A stochastic PCA and SVD algorithm with an exponential convergence rate. In *Proc. of the 32st Int. Conf. Machine Learning (ICML 2015)*, pages 144–152, 2015.

[17] Si Si, Donghyuk Shin, Inderjit S Dhillon, and Beresford N Parlett. Multi-scale spectral decomposition of massive graphs. In *Advances in Neural Information Processing Systems*, pages 2798–2806, 2014.

[18] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[19] Xiao-Tong Yuan and Tong Zhang. Truncated power method for sparse eigenvalue problems. *The Journal of Machine Learning Research*, 14(1):899–925, 2013.

A Appendix

A.1 Proof of Lemma 4.1

We consider the difference between $\boldsymbol{y}^{(l+1)}$ and $\frac{A\boldsymbol{x}^{(l)}}{(\boldsymbol{x}^{(l)})^T A\boldsymbol{x}^{(l)}}$ as noise, denoted by $\boldsymbol{g}^{(l)}$. To prove the results, we need to use Lemma A.1:

Lemma A.1. *For any unit norm $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{c} \stackrel{def}{=} \boldsymbol{x} - \frac{A\boldsymbol{x}}{\boldsymbol{x}^T A\boldsymbol{x}}$ satisfies $\boldsymbol{x}^T A\boldsymbol{x}\|\boldsymbol{c}\| \leq \sin\theta\lambda_1$, where θ is the angle between $\boldsymbol{v}_1$ and $\boldsymbol{x}$.*

Proof. Write $\boldsymbol{x} = \cos\theta\boldsymbol{v}_1 + \sin\theta\boldsymbol{u}$, where $\boldsymbol{u} \perp \boldsymbol{v}_1$. Then

$$\begin{aligned} & \|A\boldsymbol{x} - (\boldsymbol{x}^T A\boldsymbol{x})\boldsymbol{x}\|^2 \\ = \; & \|\cos\theta\lambda_1\boldsymbol{v}_1 + \sin\theta A\boldsymbol{u} - (\cos^2\theta\lambda_1 + \sin^2\theta\boldsymbol{u}^T A\boldsymbol{u})(\cos\theta\boldsymbol{v}_1 + \sin\theta\boldsymbol{u})\|^2 \\ = \; & \|\cos\theta\sin^2\theta(\lambda_1 - \boldsymbol{u}^T A\boldsymbol{u})\boldsymbol{v}_1 + \sin\theta(\cos^2\theta((\boldsymbol{u}^T A\boldsymbol{u})\boldsymbol{u} - \lambda_1\boldsymbol{u}) + (A\boldsymbol{u} - (\boldsymbol{u}^T A\boldsymbol{u})\boldsymbol{u}))\|^2 \end{aligned}$$

Notice $\boldsymbol{u}$, $A\boldsymbol{u} - (\boldsymbol{u}^T A\boldsymbol{u})\boldsymbol{u}$ and $\boldsymbol{v}_1$ are orthogonal to each other. Therefore,

$$\begin{aligned} & \|A\boldsymbol{x} - (\boldsymbol{x}^T A\boldsymbol{x})\boldsymbol{x}\|^2 \\ = \; & \cos^2\theta\sin^2\theta(\lambda_1 - \boldsymbol{u}^T A\boldsymbol{u})^2 + \sin^2\theta\|A\boldsymbol{u} - (\boldsymbol{u}^T A\boldsymbol{u})\boldsymbol{u}\|^2 \\ \leq \; & \sin^2\theta(\lambda_1 - \boldsymbol{u}^T A\boldsymbol{u})^2 + \sin^2\theta\|A\boldsymbol{u}\|^2 \\ \leq \; & (\lambda_1\sin\theta)^2 \end{aligned}$$

The last step makes use of the fact that $\lambda_1 A - A^T A$ is positive semidefinite, so that $\lambda_1\boldsymbol{u}^T A\boldsymbol{u} \geq \boldsymbol{u}^T A^T A\boldsymbol{u} = \|A\boldsymbol{u}\|^2$ for any $\boldsymbol{u}$. □

Now we have the following corollary.

Corollary A.1.1. *For $\boldsymbol{g}^{(l)}$, $\boldsymbol{x}^{(l)}$, $\phi^{(l)}(k)$ defined for Algorithm 1, $(\boldsymbol{x}^{(l)})^T A\boldsymbol{x}^{(l)}\|\boldsymbol{g}^{(l)}\| \leq \sin\theta^{(l)}\lambda_1\phi^{(l)}(k)$.*

This result is crucial to the following proof of Lemma 4.1.

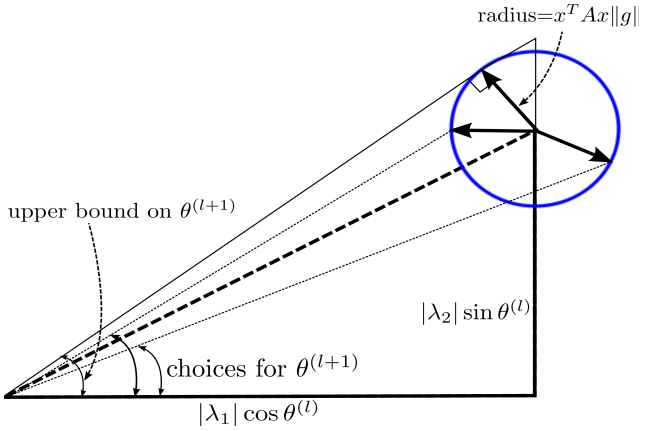


Figure 5: The central right triangle has a base-side of length $\lambda_1\cos\theta^{(l)}$ and height of at most $\lambda_2\sin\theta^{(l)}$. The dashed line that ends in the center of the circle is $A\boldsymbol{x}$ and the straight lines with an arrow are possible directions of $\boldsymbol{g}$. Then Eq (11) can be represented by the tangent of the angle between the base-side and the dotted lines that ends on the circle of radius $\boldsymbol{x}^T A\boldsymbol{x}\|\boldsymbol{g}\|$. Therefore $\tan\theta^{(l+1)} \leq \frac{\lambda_2\sin\theta^l + \boldsymbol{x}^T A\boldsymbol{x}\|\boldsymbol{g}\|/\cos\theta^{(l+1)}}{\lambda_1\cos\theta^{(l)}}$.

Let $U \in \mathbb{R}^{n\times(n-1)} = [\boldsymbol{v}_2|\boldsymbol{v}_3|\cdots|\boldsymbol{v}_n]$ denote the orthonormal space of $\boldsymbol{v}_1$. The next iterate satisfies:

$$\begin{aligned}
& \tan\theta^{(l+1)} \\
= & \frac{\|U^T\boldsymbol{y}^{(l+1)}\|}{\boldsymbol{v}_1^T\cdot\boldsymbol{y}^{(l+1)}} \\
= & \frac{\|U^T\frac{A\boldsymbol{x}^{(l)}}{(\boldsymbol{x}^{(l)})^TA\boldsymbol{x}^{(l)}}+U^T\boldsymbol{g}^{(l)}\|}{\boldsymbol{v}_1^T\frac{A\boldsymbol{x}^{(l)}}{(\boldsymbol{x}^{(l)})^TA\boldsymbol{x}^{(l)}}+\boldsymbol{v}_1^T\boldsymbol{g}^{(l)}} \\
\leq & \frac{\sin\theta^{(l)}\lambda_2+(\boldsymbol{x}^{(l)})^TA\boldsymbol{x}^{(l)}\|U^T\boldsymbol{g}^{(l)}\|}{\cos\theta^{(l)}\lambda_1+(\boldsymbol{x}^{(l)})^TA\boldsymbol{x}^{(l)}\boldsymbol{v}_1^T\boldsymbol{g}^{(l)}} \qquad (11) \\
\leq & \frac{\sin\theta^{(l)}\lambda_2+(\boldsymbol{x}^{(l)})^TA\boldsymbol{x}^{(l)}\|\boldsymbol{g}^{(l)}\|/\cos\theta^{(l+1)}}{\cos\theta^{(l)}\lambda_1} \qquad (12)
\end{aligned}$$

The logic from Eq (11) to Eq (12) is interpreted in Figure A.1.

Applying Lemma A.1 on Inequality (11), one gets

$$\tan\theta^{(l+1)} \leq \frac{\sin\theta^{(l)}\lambda_2+\phi(k)\sin\theta^{(l)}\lambda_1}{\cos\theta^{(l)}\lambda_1-\phi(k)\sin\theta^{(l)}\lambda_1} = \tan\theta^{(l)}\frac{\lambda_2+\lambda_1\phi(k)}{\lambda_1(1-\tan\theta^{(l)}\phi(k))}$$

Therefore with large enough k such that $\phi(k) \leq \frac{\lambda_1-\lambda_2}{\lambda_1(1+\tan\theta^{(l)})}$, we could guarentee that $\theta^{(l+1)} < \theta^{(l)}$, $\frac{1}{\cos\theta^{(l+1)}} < \frac{1}{\cos\theta^{(l)}}$. So continuing Eq. (12), we have

$$\begin{aligned}
\tan\theta^{(l+1)} & \leq \frac{\sin\theta^{(l)}\lambda_2+(\boldsymbol{x}^{(l)})^TA\boldsymbol{x}^{(l)}\|\boldsymbol{g}^{(l)}\|/\cos\theta^{(l)}}{\cos\theta^{(l)}\lambda_1} \\
& \leq \frac{\sin\theta^{(l)}\lambda_2+\phi(k)\sin\theta^{(l)}\lambda_1/\cos\theta^{(l)}}{\cos\theta^{(l)}\lambda_1} \\
& = \tan\theta^{(l)}\left(\frac{\lambda_2}{\lambda_1}+\frac{\phi(k)}{\cos\theta^{(l)}}\right)
\end{aligned}$$

A.2 Proof of Theorem 4.2

When $\phi^{(l)}(k) \leq (\lambda_1-\lambda_2)/(2\lambda_1(1+\tan\theta^{(l)}))$, we obtain that,

$$\frac{\phi^{(l)}(k)}{\cos\theta^{(l)}} \leq \frac{\lambda_1-\lambda_2}{2\lambda_1(\cos\theta^{(l)}+\sin\theta^{(l)})} \leq (\lambda_1-\lambda_2)/(2\lambda_1)$$

$$\begin{aligned}
\tan\theta^{(l)} & \leq \tan\theta^{(l-1)}(\lambda_2/\lambda_1+(\lambda_1-\lambda_2)/(2\lambda_1)) \qquad (13) \\
& \leq \tan\theta^{(0)}\left(\frac{\lambda_1+\lambda_2}{2\lambda_1}\right)^l \qquad (14) \\
& \leq \tan\theta^{(0)}e^{-l(\lambda_1-\lambda_2)/(2\lambda_1)} \qquad (15)
\end{aligned}$$

Therefore when $l \geq 2\frac{\lambda_1}{\lambda_1-\lambda_2}\log\frac{\tan\theta^{(0)}}{\varepsilon}$, $\tan\theta^{(l)} \leq \varepsilon$.

A.3 Proof of Corollary

To compare convergence rate between CPM and PM in comparable operations, one should notice one iteration of CPM costs around $\frac{k}{n}$ percentage of operations as PM does. Therefore we should compare our convergence rate $\frac{\lambda_1+\lambda_2}{2\lambda_1}$ with $(\frac{\lambda_2}{\lambda_1})^{\frac{k}{n}}$. Therefore when

$$k < \frac{\log\left(\frac{\lambda_1+\lambda_2}{2\lambda_1}\right)}{\log\frac{\lambda_2}{\lambda_1}}n,$$

our convergence rate is better than power method in terms of equivalent passes over data.

A.4 Proof of Theorem 4.3

Lemma A.2. *Let $f(\boldsymbol{x}) := \|A - \boldsymbol{x}\boldsymbol{x}^T\|_F^2$. Then within the area $x \in B_r(\sqrt{\lambda_1}\boldsymbol{v}_1) = \{y | \|y - \sqrt{\lambda_1}\boldsymbol{v}_1\| \leq r\}, r = O(\sqrt{\lambda_1} - \frac{\lambda_2}{\sqrt{\lambda_1}})$, $f(\boldsymbol{x})$ is strongly convex.*

Proof of A.2. Notice that the objective function f has gradient $\nabla f(\boldsymbol{x}) = -4(A\boldsymbol{x} - \|\boldsymbol{x}\|^2\boldsymbol{x})$, Hessian matrix $H(\boldsymbol{x}) = -4(A - \|\boldsymbol{x}\|^2 I - 2\boldsymbol{x}\boldsymbol{x}^T)$, and stationary points $x_i = \sqrt{\lambda_i}\boldsymbol{v}_i$. Denote the eigenvalues $\lambda_1 > \lambda_2 \geq \cdots \geq 0 \geq \cdots \lambda_n$, and with the assumption that the dominant eigenvalue is positive, we have $\lambda_1 > |\lambda_n|$.
At point $\sqrt{\lambda_1}\boldsymbol{v}_1$, the Hessian matrix of f is positive definite:

$$\begin{aligned} H(\sqrt{\lambda_1}\boldsymbol{v}_1) &= -4(A - \lambda_1 I - 2\lambda_1 \boldsymbol{v}_1\boldsymbol{v}_1^T) \\ &= 4\lambda_1\boldsymbol{v}_1\boldsymbol{v}_1^T + 4\lambda_1 I - 4\sum_{i=2}^{n}\lambda_i\boldsymbol{v}_i\boldsymbol{v}_i^T \end{aligned}$$

In particular, H has the same eigenvectors as A: $\boldsymbol{v}_1, \boldsymbol{v}_2 \cdots \boldsymbol{v}_n$, with respect to eigenvalues $8\lambda_1, 4(\lambda_1 - \lambda_2), 4(\lambda_1 - \lambda_3), \cdots, 4(\lambda_1 - \lambda_n)$, which indicates that H is positive definite with its smallest eigenvalue $4(\lambda_1 - \lambda_2) > 0$.
Now to show f is strongly convex within the neighborhood $B_r(\sqrt{\lambda_1}\boldsymbol{v}_1)$, we denote $\boldsymbol{x} = \sqrt{\lambda_1}\boldsymbol{v}_1 + \boldsymbol{h}$, $\|\boldsymbol{h}\| \leq r$, and introduce

$$G(\boldsymbol{h}, \boldsymbol{g}) \stackrel{\text{def}}{=} \frac{\boldsymbol{g}^T H(\sqrt{\lambda_1}\boldsymbol{v}_1 + \boldsymbol{h})\boldsymbol{g}}{\boldsymbol{g}^T\boldsymbol{g}},$$

which could represent the range of eigenvalues of $H(\sqrt{\lambda_1}\boldsymbol{v}_1 + \boldsymbol{h})$. Notice

$$\begin{aligned} \nabla_{\boldsymbol{h}} G(\boldsymbol{h}, \boldsymbol{g}) &= 8\sqrt{\lambda_1}\boldsymbol{v}_1 + 8\boldsymbol{h} + 16(\sqrt{\lambda_1}\boldsymbol{v}_1^T\frac{\boldsymbol{g}}{\|\boldsymbol{g}\|} + \boldsymbol{h}^T\frac{\boldsymbol{g}}{\|\boldsymbol{g}\|})\frac{\boldsymbol{g}}{\|\boldsymbol{g}\|} \\ \text{, and } \|\nabla_{\boldsymbol{h}} G(\boldsymbol{h}, \boldsymbol{g})\| &\leq 8\sqrt{\lambda_1} + 8\|\boldsymbol{h}\| + 16(\sqrt{\lambda_1} + \|\boldsymbol{h}\|) \\ &= 24(\sqrt{\lambda_1} + \|\boldsymbol{h}\|) \\ &\leq 24(\sqrt{\lambda_1} + r), \forall \boldsymbol{h} \in B_r(0) \end{aligned}$$

By mean-value theorem,

$$\begin{aligned} |G(\boldsymbol{h}, \boldsymbol{g}) - G(0, \boldsymbol{g})| &\leq (\sup_{\boldsymbol{h} \in B_r(0)} \|\nabla_{\boldsymbol{h}} G(\boldsymbol{h}, \boldsymbol{g})\|)\|\boldsymbol{h}\| \\ &\leq 24(\sqrt{\lambda_1} + r)r, \forall \boldsymbol{h} \in B_r(0), \forall \boldsymbol{g} \in \mathbb{R}^n \end{aligned}$$

When $r = \frac{1}{30}\frac{\lambda_1 - \lambda_2}{\sqrt{\lambda_1}}$, we have $|G(\boldsymbol{h}, \boldsymbol{g}) - G(0, \boldsymbol{g})| \leq \lambda_1 - \lambda_2$.

Recall $G(0, \boldsymbol{g}) = \frac{\boldsymbol{g}^T H(\sqrt{\lambda_1}\boldsymbol{v}_1)\boldsymbol{g}}{\|\boldsymbol{g}\|^2} \geq 4(\lambda_1 - \lambda_2),\ \forall \boldsymbol{g} \in \mathbb{R}^n$.

$$\begin{aligned} G(\boldsymbol{h}, \boldsymbol{g}) &\geq G(0, \boldsymbol{g}) - |G(\boldsymbol{h}, \boldsymbol{g}) - G(0, \boldsymbol{g})| \\ &\geq 3(\lambda_1 - \lambda_2),\ \forall \boldsymbol{g} \in \mathbb{R}^n, \|\boldsymbol{h}\| < r, \\ &i.e. \\ H(\sqrt{\lambda_1}\boldsymbol{v}_1 + \boldsymbol{h}) &\succeq 3(\lambda_1 - \lambda_2), \forall \boldsymbol{h}, \|\boldsymbol{h}\| \leq \frac{1}{30}\frac{\lambda_1 - \lambda_2}{\sqrt{\lambda_1}} \end{aligned}$$

Therefore the cost function is $3(\lambda_1 - \lambda_2)$-strongly convex within the area $\boldsymbol{x} \in B_r(\sqrt{\lambda_1}\boldsymbol{v}_1)$. □

Lemma A.3. *In area $B_r(\sqrt{\lambda_1}\boldsymbol{v}_1)$, where r=$\frac{\lambda_1 - \lambda_2}{30\sqrt{\lambda_1}}$, $\nabla_i f$ satisfies coordinate-wise Lipschitz continuous with parameter $L \leq 14\lambda_1 - 2\lambda_2 + 4\max_i |a_{ii}|$.*

Proof of Lemma A.3: Our goal is to find L that satisfies $|\nabla_i f(\boldsymbol{x} + \alpha\boldsymbol{e}_i) - \nabla_i f(\boldsymbol{x})| \leq L|\alpha|$, $\forall \boldsymbol{x}, \alpha\ s.t.\ \boldsymbol{x}, \boldsymbol{x} + \alpha\boldsymbol{e}_i \in B_r(\sqrt{\lambda_1}\boldsymbol{v}_1)$.

Notice that $r = \frac{\lambda_1 - \lambda_2}{30\sqrt{\lambda_1}}$, and $\|\boldsymbol{x}\| \leq \sqrt{\lambda_1} + r$, $|\alpha| \leq 2r$.
Now

$$\begin{aligned} & |\nabla_i f(\boldsymbol{x} + \alpha \boldsymbol{e}_i) - \nabla_i f(\boldsymbol{x})| \\ = \; & 4|\|\boldsymbol{x} + \alpha \boldsymbol{e}_i\|^2 (x_i + \alpha) - a_{ii}\alpha - \|\boldsymbol{x}\|^2 x_i| \\ \leq \; & 4|\|\boldsymbol{x} + \alpha \boldsymbol{e}_i\|^2 \alpha + \alpha^2 x_i + 2\alpha x_i^2| + 4|a_{ii}\alpha| \\ \leq \; & 4|\alpha|((\sqrt{\lambda_1} + r)^2 + 2r(\sqrt{\lambda_1} + r) + 2(\sqrt{\lambda_1} + r)^2) + 4|a_{ii}\alpha| \\ = \; & 4|\alpha|(3\lambda_1 + 10\sqrt{\lambda_1} r + 5r^2) + 4|a_{ii}\alpha| \\ \leq \; & [12\lambda_1 + 2(\lambda_1 - \lambda_2) + 4|a_{ii}|]|\alpha| \end{aligned}$$

□

Remark: $L = 14\lambda_1 - 2\lambda_2 + 4\max_i |a_{ii}|$, for real application like social network, $a_{ii} = 0$ and $L = 14\lambda_1 - 2\lambda_2$.

With the Lipschitz continuous and strongly convex properties, we show convergence by quoting the result of [13]:

Lemma A.4. *Let f be a strongly convex with $\nabla^2 f \succeq \alpha I$ and ∇f satisfy coordinate-wise L-Lipschitz continuous, meaning*

$$|\nabla_i f(\boldsymbol{x} + \alpha \boldsymbol{e}_i) - \nabla_i f(\boldsymbol{x})| \leq L|\alpha|,$$

$\forall i = 1, 2, \cdots n, \forall \boldsymbol{x} \in$ convex set $\mathbb{S}$, and $\forall \alpha$ such that $\boldsymbol{x} + \alpha \in \mathbb{S}$. Then with Gauss-Southwell rule the greedy coordinate descent on f satisfies linear convergence:

$$f(\boldsymbol{x}^{(l+1)}) - f(\boldsymbol{x}^*) \leq (1 - \frac{\mu_1}{L})[f(\boldsymbol{x}^{(l)}) - f(\boldsymbol{x}^*)]. \tag{16}$$

Here $\mu_1 = \inf_{\boldsymbol{x}, \boldsymbol{y} \in \mathbb{S}} \frac{\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\|_\infty}{\|\boldsymbol{x} - \boldsymbol{y}\|_1} \in [\frac{\alpha}{n}, \alpha]$

Therefore, the convergence rate for updating one coordinate at a time with Gauss-Southwell rule becomes $(1 - \frac{\mu}{L})$, $\mu = \inf_{\boldsymbol{x}, \boldsymbol{y}} \frac{\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\|_\infty}{\|\boldsymbol{x} - \boldsymbol{y}\|_1} \in [\frac{3(\lambda_1 - \lambda_2)}{n}, 3(\lambda_1 - \lambda_2)]$, $L = 14\lambda_1 - 2\lambda_2 + 4\max_i |a_{ii}|$.

A.5 Greedy Coordinate Descent and Coordinate Selection Rules

For an arbitrary matrix $A \in \mathbb{R}^{n \times d}$, we can formulate rank-1 matrix approximation:

$$\underset{\boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{y} \in \mathbb{R}^d}{\arg\min} \; f(\boldsymbol{x}, \boldsymbol{y}) = \|A - \boldsymbol{x}\boldsymbol{y}^T\|_F^2 \tag{17}$$

Notice that $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}, \boldsymbol{y}) = 2(\|\boldsymbol{y}\|^2 \boldsymbol{x} - A\boldsymbol{y})$. When fixing $\boldsymbol{y}$, we obtain the optimal solution of $\boldsymbol{x}$ to be $\boldsymbol{x} = \frac{A\boldsymbol{y}}{\|\boldsymbol{y}\|^2}$ and vice versa, $\boldsymbol{y} = \frac{A^T \boldsymbol{x}}{\|\boldsymbol{x}\|^2}$. And for symmetric matrices, this alternating minimization algorithm is exactly power method apart from the normalization constant.

Recall our coordinate-wise power method. At each iteration we only update the coordinates with the largest changes. Nevertheless here we can formally interpret this rule as the well-studied Gauss-Southwell rule [12], where the coordinates that maximize the gradient norm is selected. As $\nabla_{x_i} f(\boldsymbol{x}, \boldsymbol{y}) = 2(\|\boldsymbol{y}\|^2 x_i - \boldsymbol{a}_i^T \boldsymbol{y}) = 2\|\boldsymbol{y}\|^2 (x_i - \frac{\boldsymbol{a}_i^T \boldsymbol{y}}{\|\boldsymbol{y}\|^2})$, Gauss-Southwell gives the same choice of coordinates as our coordinate-wise power method.

Meanwhile, specifically for quadratic objectives, Gauss-Southwell rule actually select the coordinates based on the decrease in the objective function, leading to optimal updates, i.e.,

$$\Delta f_i \quad := \quad f(\boldsymbol{x}', \boldsymbol{y}) - f(\boldsymbol{x}, \boldsymbol{y}) = -\|\boldsymbol{y}\|_2^2 (x_i - \frac{\boldsymbol{a}_i^T \boldsymbol{y}}{\|\boldsymbol{y}\|_2^2})^2 = -\frac{(\nabla_{x_i} f)^2}{4\|\boldsymbol{y}\|^2}$$

where $\boldsymbol{x}' = \boldsymbol{x} + (x_i' - x_i)\mathbf{e}_i$, and $x_i' = \frac{\boldsymbol{a}_i^T \boldsymbol{y}}{\|\boldsymbol{y}\|^2}$ is the updated coordinate.

Here we summarize the three coordinate selection rules: **(a)** largest coordinate value change, $|x_i' - x_i|$, where x_i' is the next iterate; **(b)** largest partial gradient (Gauss-Southwell), $|\nabla_i f(\boldsymbol{x})|$; **(c)** largest

function value decrease, $|f(\boldsymbol{x}') - f(\boldsymbol{x})|$, where $\boldsymbol{x}'_i = \boldsymbol{x} + (x'_i - x_i)\boldsymbol{e}_i$. With the good property of quadratic function Eq. (4), for each alternating minimization step, the three selection rules are equivalent. Therefore now with the aid of the objective function, our coordinate selection strategy in CPM, similar as in **(a)**, is now consistent with the rule **(c)** with its nature in choosing the most "important" coordinates.

Given the optimization interpretation, the extension of CPM to computing the top-r eigenvectors of a symmetric matrix is straightforward. For the objective function $f(X, Y) = \|A - XY^T\|_F^2$, where $X, Y \in \mathbb{R}^{n\times r}$, the partial gradient of $f(X, Y)$ with respect to matrices X, Y becomes $2(XY^TY - AY)$ and $2(YX^TX - AX)$ respectively. By evaluating the norm in each rows of the gradient, we could select and update row by row for X and Y by $\boldsymbol{a}_i^TY(Y^TY)^{-1}$ and $\boldsymbol{a}_i^TX(X^TX)^{-1}$. Although the algorithm is well-defined and can speedup power method for computing top-r eigenvectors, power method (a.k.a. subspace iteration) is typically not used for computing the dominant r(especially for large r) eigenvectors [17]. Therefore we don't expand the discussion of this direction here.

A.6 Choice of k

The choice of k could be viewed as choosing the block size for greedy block coordinate descent, which is usually tuned in practice or determined by the objective's separable property.

However, it would be better if k could be prescribed and only depend on n, as we don't know other properties like $\frac{\lambda_2}{\lambda_1}$ beforehand. In Corollary 4.2.1 it shows the upper bound of k ranges from $6\%n$ to $50\%n$ when $\frac{\lambda_2}{\lambda_1}$ ranges from 10^{-5} to $1-10^{-5}$. Meanwhile, experiments also show that the performance of our algorithms isn't too sensitive to the choice of k. As shown in Figure 6, a large range of k guarantees good performances. Thus we choose $k = \frac{n}{20}$ throughout our experiments in this paper, which is a theoretically and experimentally favorable choice.

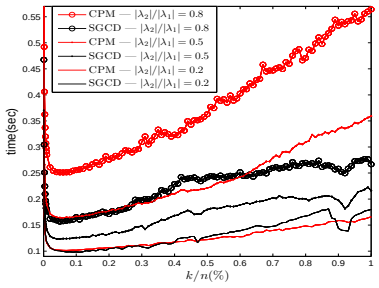


Figure 6: Convergence time with different k for different λ_2/λ_1

A.7 Out-of-core Algorithm

Here we formally present the algorithm for the out-of-core case.

Algorithm 3 PM,CPM,SGCD for out-of-core matrix A

1: **Initialization:** Separate and save matrix $A \in \mathbb{R}^{n\times n}$ into m files, each containing n/m rows of A and being able to fit into memory. Initialize a random unit vector $x^{(0)}$.
2: **for** $l = 1$ **to** L **do**
3: **for** $i = 1$ **to** m **do**
4: Set $\Omega = (\frac{(i-1)n}{m} + 1) : \frac{in}{m}$.
5: For PM, calculate $A_{\Omega,:}\boldsymbol{x}^{(l-1)}$
6: For CPM, do Step 4 in Algorithm 1 for t times.
7: For SGCD, do Step 4 in Algorithm 2 for t times.
8: Update $\boldsymbol{x}^{(l)}$.
9: **Output:** Approximate dominant eigenvector $\boldsymbol{x}^{(L)}$

A.8 Extension of Coordinate-wise Mechanism on the Jacobi method

The algorithm for coordinate-wise Jacobi method for solving $A\boldsymbol{x} = \boldsymbol{b}$ is included in Alg. 4.

At each iteration, it takes $O(nnz(R) + n)$ operations for naive Jacobi, and $O(\frac{k}{n}nnz(R) + n)$ for coordinate-wise Jacobi. This coordinate-wise methanism also reminds us of Gauss-Seidel method. Recall that Gauss-Seidel:

Initialize: $A = L + U$, where L is a lower triangular matrix and U is an upper trianglular matrix
Iterations: $\boldsymbol{x}^+ \leftarrow L^{-1}(\boldsymbol{b} - U\boldsymbol{x})$.

Algorithm 4 Coordinate-wise Jacobi Method

1: **Input:** Symmetric diagonal dominant matrix $A \in \mathbb{R}^{n\times n}$, vector $\boldsymbol{b} \in \mathbb{R}^n$, number of selected coordinates, k, and number of iterations, L.
2: Initialize $\boldsymbol{x}^{(0)} \in \mathbb{R}^n$ and set $A = D + R$, where D is diagonal component of A and R is the remainder part. $\boldsymbol{z}^{(0)} = R\boldsymbol{x}^{(0)}$. Set the coordinate selecting criterion $\boldsymbol{c}^{(0)} = \boldsymbol{b} - A\boldsymbol{x}^{(0)} = b - D\boldsymbol{x}^{(0)} - \boldsymbol{z}^{(0)}$.
3: **for** $l = 1$ **to** L **do**
4: Let $\Omega^{(l)}$ be a set containing k coordinates of $\boldsymbol{c}^{(l-1)}$ with the largest magnitude. Execute the following updates:

$$\begin{aligned} x_j^{(l)} &= \begin{cases} (b_j - z_j^{(l-1)})/D_{jj}, & j \in \Omega^{(l)} \\ x_j^{(l-1)}, & j \notin \Omega^{(l)} \end{cases} \\ \boldsymbol{z}^{(l)} &= \boldsymbol{z}^{(l-1)} + R(\boldsymbol{x}^{(l)}_{\Omega^{(l)}} - \boldsymbol{x}^{(l-1)}_{\Omega^{(l)}}) \\ \boldsymbol{c}^{(l)} &= \boldsymbol{b} - D\boldsymbol{x}^{(l)} - \boldsymbol{z}^{(l)} \end{aligned}$$

5: **Output:** $\boldsymbol{x}^{(L)}$

Taking advantage of triangular form, the procedure could be simplified as the following version,

$$x_i^{(l+1)} \leftarrow \frac{1}{a_{ii}}(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(l+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(l)})$$

which is very similar to Jacobi method, but uses a forward substitution on newly computed x_i. Therefore our method is also like a greedy block version of Gauss-Seidel method. While Gauss-Seidel is like a cyclic coordinate version of our method.

We use Matlab to do some simple experiments on some synthetic data to measure the convergence time until the error is less than $1e-5$. Here the error is measured by A-quadratic norm between current iteration $\boldsymbol{x}^{(l)}$ and ground truth $\boldsymbol{x}^*$, namely, $\|\boldsymbol{x}^{(l)} - \boldsymbol{x}^*\|_A = \sqrt{(\boldsymbol{x}^{(l)} - \boldsymbol{x}^*)^T A(\boldsymbol{x}^{(l)} - \boldsymbol{x}^*)}$.

Table 2: Comparison between Jacobi method and Coordinate-wise Jacobi method. N/A denotes the algorithm doesn't converge.

Dataset	n	$\frac{\lambda_2}{\lambda_1}(A)$	$\sigma(D^{-1}R)$	Flops($/n^2$)			Speedup	
				Jacobi	C-Jacobi	Gauss-Seidel	on Jacobi	on G-S
1	1000	0.7803	0.6870	35.035	**4.794**	7.007	**7.308**	**1.462**
2	1000	0.5565	0.9524	254.254	**4.284**	9.009	**59.350**	**2.103**
3	1000	0.5224	0.9942	2115.113	**4.488**	9.009	**471.282**	**2.007**
4	1000	0.5206	0.9986	8505.50	**4.08**	9.009	**2084.68**	**2.2081**
5	1000	0.495	**1.11**	N/A	**4.386**	9.009	N/A	**2.054**
6	5000	0.7792	0.6948	40.01	**5.321**	8.002	**7.519**	**1.504**
7	5000	0.5443	0.9529	290.058	**4.317**	9.002	**67.187**	**2.085**
8	5000	0.5146	0.9949	2703.54	**5.622**	10.002	**480.852**	**1.779**
9	5000	0.5111	0.9992	19760.0	**6.256**	10.002	**3158.76**	**1.599**
10	5000	0.5063	**1.02**	N/A	**6.256**	10.002	N/A	**1.599**

From Table A.8, we can see that coordinate-wise Jacobi method shows significant speedup over the naive Jacobi method. And even when the matrix is no longer diagonal dominant, (see table when $\sigma(D^{-1}R) > 1$), but still positive definite, coordinate-wise Jacobi method still converges. This trait meets the convergence requirement for Gauss-Seidel method.

Although in this comparison coordinate-wise Jacobi doesn't beat up Gauss-Seidel that much, Gauss-Seidel has the disadvantage that it can not be done in parallel, while our method could be more flexible on that. For example, we could greedily update coordinates in each worker, rather than choose globally the most greedy coordinates.

However, since for symmetric diagonal dominant matrices, Jacobi or Gauss-Seidel is not the state-of-the-art method for solving linear system, we will need to compare with other more powerful methods. And this algorithm lacks theoretical support at this point, so we consider this as an expansion of our current work on coordinate-wise power method. But still, it's worth mentioning that the coordinate-wise mechanism could be powerful applying to Jacobi method and maybe to other iterative methods in linear algebra too. Therefore in the future, we may continue exploiting the theory behind, and analyze why and how greediness impacts on Jacobi method or other iterative methods in linear algebra.