A Greedy Randomized Block Coordinate Descent Algorithm with k-means Clustering for Solving Large Linear Least-squares Problems

Ke Zhang, Chun-Ting Liu and Xiang-Long Jiang

***Abstract*—By exploiting the column partitioning using the k-means clustering, we develop a greedy randomized block coordinate descent algorithm for solving large-scale linear least squares problems. The proposed method works on submatrices of the coefficient matrix, which dramatically reduces the computational time and iterations steps. We prove that the new algorithm converges to the unique solution of the least squares problem with full-rank overdetermined coefficient matrix. The numerical performance of the presented method is validated in comparison with some existing randomized coordinate descent counterparts.**

***Index Terms*—coordinate descent method, GRCD, linear least squares problem, randomized iterations.**

I. Introduction

We consider solving the linear least squares problem in the form

$$Ax \approx b, \tag{1}$$

or equivalently,

$$\min_{x\in\mathbb{R}^n} \|b - Ax\|_2^2,$$

where $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$ and $x \in \mathbb{R}^n$. Throughout this paper, we are concerned with the most commonly occurring case when $m > n$ and $\mathrm{rank}(A) = n$.

Least squares problems can be found in varying fields, including statistics, image reconstruction and control [9], [13], [24], [27]. This has aroused revived interest in developing efficient numerical methods for solving (1); the option among them can be problem-dependent and involves trade-offs among accuracy, efficiency, and stability. The first class of methods for solving (1) is the direct methods. For example, the normal equations method which requires simply the matrix multiplication and Cholesky factorization is perhaps the simplest to implement. However, the condition number of the coefficient matrix in the normal equations is squared. The other well-known direct methods for solving (1) are the orthogonalization methods. Among them, the Householder method highlights itself for its efficiency and accuracy. For more about direct methods for solving (1), we refer to the monographs [9], [18] and references therein.

In general, the foregoing direct methods work well for solving (1) of small-to-medium size. However, when the size is extremely large, the iterative methods are often preferred and in many cases become the only choice [8]. Among them, the randomized extended Kaczmarz (abbreviated to REK) method [32] and the randomized coordinate descent (abbreviated to RCD) method [19] have received much attention over the past decades. The REK method breaks the barrier of the classical Kaczmarz method [16] and the randomized variant (abbreviated to RK) [25] in solving (1) by knitting two stages together; the first stage is a randomized orthogonal projection aimed for computing the the projection of b onto the orthogonal complement of the column space of A, while the second is for approximating the least squares solution via the RK method. More recent developments on RK, REK and related topics are available in, to name a few, [6], [2], [7], [11], [21], [30], [31].

Unlike the REK method, the RCD method [19] projects the current residual vector onto the the orthogonal complement of the working column vector of A such that the norm of the residual vector in the next iteration is minimized, where the working column vector is chosen as per the importance sampling of the columns. Ma et al. provide a unified theoretical finding of RK and RCD and draw connections between them [22]. In [11], Du presents some tighter bounds for the convergence of REK and an extended version of RCD discussed in [22]. In [12], Dumitrescu unravels the relationship between REK and RCD and concludes that RCD requires fewer operations for a given residual-based termination criterion. In [3], Bai et al. come up with an exact formula for residuals generated by the RCD method, based on which they estimate an upper bound for the convergence rate of RCD. To speed up the convergence of RCD, Bai and Wu incorporate the greedy technique used in [4] into the framework of RCD and propose a greedy randomized coordinate descent (GRCD for short) method [5]. Zhang and Guo [29] present a relaxed variant of GRCD by introducing a parameter to the index set which adds more flexibility into the implementation of GRCD. To further exploit information of the index set in GRCD, Tan and Guo [26] select multiple columns independently from the index set and cyclically update the iterate for a user-prescribed number of times. Following a similar reasoning, Chen and Huang [10] put forth a block variant of GRCD by using all columns from the index set in GRCD at each iteration. By exploiting all coordinates from the index set in GRCD, Li and Zhang [20] come up with a greedy block Gauss-Seidel (GBGS)

Manuscript received December 16, 2023; revised April 1, 2024.

This work was supported in part by the National Natural Science Foundation of China under Grant 12271342.

Ke Zhang is a lecturer of the Department of Mathematics, Shanghai Maritime University, Shanghai, 201306, China (corresponding author to provide phone:+86-21-3828-2215; e-mail: kezhang@shmtu.edu.cn).

Chun-Ting Liu is a postgraduate student of the Department of Mathematics, Shanghai Maritime University, Shanghai, 201306, China (e-mail: liuchunting2023@163.com).

Xiang-Long Jiang is a lecturer of the Department of Mathematics, Shanghai Maritime University, Shanghai, 201306, China (e-mail: jiangxl@shmtu.edu.cn).

method. We refer to [28] for more details on coordinate descent methods and their application in solving optimization problems.

For linear systems or least squares problems of large scale, it is inspiring to transform a high-dimensional problem into a lower one that still preserves the essence of the original information. This is the main idea of dimension reduction. In the context of solving consistent linear systems, two authors of this work [15] have successfully applied the clustering algorithm [23] to the rows of the original linear system such that only a much smaller linear system needs to be handled. Following this philosophy, we propose a novel greedy randomized block coordinate descent algorithm with k-means clustering. Instead of applying the k-means method to rows of the coefficient matrix A as done in [15], we impose the k-means clustering on columns of A, resulting in a column partitioning of A. By doing so, the remaining process of GRCD only needs to be enforced on a smaller linear least squares problem, which can be very effective in reducing the number of iterations and CPU time. The resulting algorithm is called the greedy randomized block coordinate descent with k-means clustering (GRBCD(k)).

In this paper, we use $\|A\|_F$ to denote the matrix Frobenius norm and $\|\cdot\|_2$ the 2-norm of a vector. The symbol $A^\dagger$ stands for the Moore-Penrose pseudoinverse. The transpose of a matrix (or vector) is given by $(\cdot)^T$. The smallest nonzero singular value and its largest peer of a matrix are represented by $\sigma_{\min}$ and $\sigma_{\max}$, respectively. The jth column from a matrix A is indicated with $A_{(j)}$. The range of a matrix A is given by $\mathcal{R}(A)$. The least-norm least-squares solution of (1) is represented by x_*. The symbol $[n]$ abbreviates the set $\{1,\ldots,n\}$.

This work is organized as follows. In Section II, we review the randomized coordinate descent method and its greedy peer. In Section III, we clarify the motivation, algorithmic details and convergence analysis of the proposed algorithm. In Section IV, we carry out some numerical experiments to verify the effectiveness of GRBCD(k) when compared with some other RCD-type methods. In Section V, we conclude this paper by discussing some potential future work.

II. The randomized coordinate descent method and its greedy variant

In this section, we introduce the randomized coordinate descent (abbreviated to RCD) method [19] and the greedy randomized coordinate descent (abbreviated to GRCD) method [5]. These two methods allow a better insight into our new method in Section III.

Let us proceed the discussion with RCD. In its simplest form, the RCD method randomly selects a canonical coordinate vector as the search direction with probability proportional to the ratio below

$$\Pr(j=j_k)=\frac{\|A_{(j_k)}\|^2}{\|A\|_F^2},\quad j_k\in[n],$$

where $A_{(j_k)}$ is the j_kth column of the matrix A and $[n]=\{1,\ldots,n\}$. Then the iterate is updated as

$$x_{k+1}=x_k+\frac{A_{(j_k)}^T(b-Ax_k)}{\|A_{(j_k)}\|_2^2}e_{j_k},\quad k=0,1,\ldots$$

Algorithm 1 The GRCD algorithm

Input: A, b, x_0 and l
Output: x_l
1: **for** $k = 0,1,\ldots,l-1$ **do**
2: Compute

$$\epsilon_k=\frac{1}{2\|A^Tr_k\|_2^2}\max_{1\le j\le n}\frac{|A_{(j)}^Tr_k|^2}{\|A_{(j)}\|_2^2}+\frac{1}{2\|A\|_F^2}.$$

3: Define

$$V_k=\left\{j\big||A_{(j)}^Tr_k|^2\ge\epsilon_k\|A^Tr_k\|_2^2\|A_{(j)}\|_2^2\right\}.$$

4: Let $s_k=A^Tr_k$ and compute the jth entry of the vector $\tilde{s}_k$ via

$$\tilde{s}_k^{(j)}=\begin{cases}s_k^{(j)}, & \text{if } j\in V_k,\\ 0, & \text{otherwise.}\end{cases}$$

5: Select $j_k\in V_k$ as per $\Pr(j=j_k)=\frac{|\tilde{s}_k^{(j_k)}|^2}{\|\tilde{s}_k\|_2^2}$.
6: Set $x_{k+1}=x_k+\frac{s_k^{(j_k)}}{\|A_{(j_k)}\|_2^2}e_{j_k}$.
7: **end for**

with $e_{j_k}\in\mathbb{R}^n$ the j_kth column of the identity matrix. It is justified that RCD is linearly convergent in expectation to the unique solution in [19]. Furthermore, the RCD method can enjoy a fast convergence if A is well conditioned with all singular values far from zero; see [19] for more details.

In implementing the RCD method, the current iterate x_k is projected onto $A_{(j_k)}^TAx=A_{(j_k)}^Tb$. Intuitively, we attempt to project $r_k=b-Ax_k$ to the vertical space of the working column $A_{(j_k)}$ where the angle between them is relatively large. Moreover, if $|A_{(i)}^Tr_k|>|A_{(j)}^Tr_k|$ for $i,j\in[n]$, then the ith column should be prioritized with a higher probability than the jth one. With this insight in hand, Bai and Wu [5] construct a nonempty index set V_k that captures large entries of A^Tr_k and selects the working column from V_k according to certain probability criterion. The resulting procedure is called the greedy randomized coordinate descent (abbreviated to GRCD) method. Numerical experiments in [5] justify the superiority of GRCD over RCD with CPU speed-up as high as 6.43 for solving inconsistent linear systems. We will not elaborate on GRCD but refer to Algorithm 2 for more algorithmic detail.

III. A greedy randomized block coordinate descent algorithm with k-means clustering

In this section, we propose a greedy randomized block coordinate descent algorithm with k-means clustering (GRBCD(k)). As explained in Section II, the GRCD algorithm often outperforms RCD for solving linear least squares problems. However, coefficient matrices in the least squares problems considered in [5] are often of much fewer columns than rows; see [5, Section 4]. In fact, the effectiveness of GRCD may decrease as the number of columns increases. RCD is also afflicted by the same problem. As such, it motivates us to find an effective way to circumvent it.

In multivariate statistics, the k-means clustering algorithm is considered to be one of the best algorithms for grouping

data into different clusters [14], [23]. In [15], two authors of this work have successfully integrated the k-means clustering into the greedy randomized Kaczmarz algorithm, yielding an effective method for solving consistent linear systems. To improve the numerical efficiency of GRCD, we employ the k-means clustering to yield a column partitioning (instead of the row partitioning in [15]) for GRCD such that only a few columns of A are treated per iteration.

Before diving into GRBCD(k), we shall give a sketch of the k-means. The k-means clustering is an unsupervised learning algorithm that exploits data patterns and classifies the unlabeled data X into different clusters $B_1, \ldots, B_k$, i.e.,

$$\bigcup_{i=1}^{k} B_i = X \quad \text{and} \quad B_i \cap B_j = \emptyset,$$

where $\emptyset$ is an empty set and $i \neq j$. The number of clusters k is pre-determined. For each iteration in k-means, we randomly initialize k points (a.k.a. means or cluster centroids). Then each item will be classified according to its nearest mean and the average of all items in that cluster aggregated so far shall be selected as a new centroid. The process is repeated within a user-prescribed maximum number of iterations and finally yields the required clusters. More details about the implementation and application of the clustering algorithms can be found in [1] and references therein.

Having said that, we are ready to introduce a greedy randomized block coordinate descent algorithm with k-means clustering (GRBCD(k)) which is given in Algorithm 2. Different from GRCD, a column partitioning $\{\tau_1, \ldots, \tau_k\}$ is obtained by employing the k-means at the initial step of GRBCD(k), where $\tau_i \subseteq [n]$ for $i = 1, \ldots, k$. The cluster centers of each block A_{τ_i}, i.e., $\bar{A}_{\tau_i}$'s, are allocated into a condensed matrix $\bar{A} = [\bar{A}_{\tau_1}, \ldots, \bar{A}_{\tau_k}] \in \mathbb{R}^{m \times k}$. In the remaining procedure of GRBCD(k), one only needs to work on the submatrix $\bar{A}$ or $\bar{A}_{\tau_i}$ rather than the whole matrix A, which presents the major difference between GRBCD(k) and GRCD. We do not dwell on it but refer to Algorithm 2 for more algorithmic details.

The following lemma shows that the index set U_j defined in Algorithm 2 is well-defined.

Lemma 1. *The set U_j defined in Algorithm 2 is nonempty.*

Proof: The proof is motivated by the analysis in [5]. Let

$$\frac{|\bar{A}_t^T r_t|^2}{\|\bar{A}_t\|_2^2} = \max_{1 \le i \le k} \frac{|\bar{A}_{\tau_i}^T r_j|^2}{\|\bar{A}_{\tau_i}\|_2^2} \text{ for } t \in [k].$$

Since

$$\frac{\max\limits_{1 \le i \le k} \frac{|\bar{A}_{\tau_i}^T r_j|^2}{\|\bar{A}_{\tau_i}\|_2^2}}{\sum\limits_{i=1}^{k} \frac{\|\bar{A}_{\tau_i}\|_2^2}{\|\bar{A}\|_F^2} \frac{|\bar{A}_{\tau_i}^T r_j|^2}{\|\bar{A}_{\tau_i}\|_2^2}} \ge 1,$$

then we have

$$\frac{\max\limits_{1 \le i \le k} \frac{|\bar{A}_{\tau_i}^T r_j|^2}{\|\bar{A}_{\tau_i}\|_2^2}}{\|\bar{A}^T r_j\|_2^2} \ge \frac{1}{\|\bar{A}\|_F^2}.$$

As a result, it holds that

$$\frac{\frac{|\bar{A}_t^T r_t|^2}{\|\bar{A}_t\|_2^2}}{\|\bar{A}^T r_j\|_2^2} = \frac{\max\limits_{1 \le i \le k} \frac{|\bar{A}_{\tau_i}^T r_j|^2}{\|\bar{A}_{\tau_i}\|_2^2}}{\|\bar{A}^T r_j\|_2^2} \ge \frac{\max\limits_{1 \le i \le k} \frac{|\bar{A}_{\tau_i}^T r_j|^2}{\|\bar{A}_{\tau_i}\|_2^2}}{2\|\bar{A}^T r_j\|_2^2} + \frac{1}{2\|\bar{A}\|_F^2} = \epsilon_j,$$

Algorithm 2 The GRBCD(k) algorithm

Input: A, b, x_0, l and k
Output: x_l

1: Imposing the k-means clustering on the column set $[n]$ of A such that a partitioning $\{\tau_1, \ldots, \tau_k\}$ is obtained, where the set $\tau_i \subseteq [n]$ for $i = 1, \ldots, k$. The cluster centers of each block A_{τ_i}, i.e., $\bar{A}_{\tau_i}$'s, are condensed into an $m \times k$ matrix $\bar{A} = [\bar{A}_{\tau_1}, \ldots, \bar{A}_{\tau_k}]$.
2: **for** $j = 0, 1, \ldots, l-1$ **do**
3: Compute

$$\epsilon_j = \frac{1}{2\|\bar{A}^T r_j\|_2^2} \max_{1 \le i \le k} \frac{|\bar{A}_{\tau_i}^T r_j|^2}{\|\bar{A}_{\tau_i}\|_2^2} + \frac{1}{2\|\bar{A}\|_F^2}.$$

4: Define

$$U_j = \left\{ \tau_j \big| |\bar{A}_{\tau_j}^T r_j|^2 / \|\bar{A}^T r_j\|_2^2 \ge \epsilon_j \|\bar{A}_{\tau_j}\|_2^2 \right\}.$$

5: Compute the τth entry $\tilde{s}_j^{(\tau)}$ of the vector $\tilde{s}_j$ via

$$\tilde{s}_j^{(\tau)} = \begin{cases} \bar{A}_\tau^T r_j, & \text{if } \tau \in U_j, \\ 0, & \text{otherwise.} \end{cases}$$

6: Select $\tau_j \in U_j$ as per $\Pr(\tau{=}\tau_j){=}\frac{|\tilde{s}_j^{(\tau)}|^2}{\|\tilde{s}_j\|_2^2}$.
7: Set $x_{j+1} = x_j + I_{\tau_j} A_{\tau_j}^\dagger r_j$ with $I_{\tau_j} \in \mathbb{R}^{n \times |\tau_j|}$ a submatrix of the identity matrix indexed from the set τ_j and $|\tau_j|$ the cardinality of τ_j.
8: **end for**

where ϵ_j is defined at step 3 of Algorithm 2. It follows that

$$|\bar{A}_t^T r_t|^2 \ge \epsilon_j \|\bar{A}^T r_j\|_2^2 \|\bar{A}_t\|_2^2,$$

that is, there exists at least an entry $t \in U_j$ such that U_j is nonempty. ■

The following result [4] that characterizes the lower bound of the norm of a matrix-vector product will be used in the proof of our main theoretical finding.

Lemma 2. *For any vector $z \in \mathcal{R}(A^T)$, it holds that*

$$\|Az\|_2^2 \ge \sigma_{\min}^2(A) \|z\|_2^2.$$

We are now in a position to establish the convergence result of GRBCD(k) which is precisely stated in the following theorem.

Theorem 1. *For any initial guess $x_0 \in \mathbb{R}^n$, the iteration sequence $\{x_j\}_{j=0}^{\infty}$ generated by GRBCD(k) converges to the unique least-squares solution x_* of (1) in expectation. Moreover, for $j = 1, 2, \ldots$, we have*

$$\mathbb{E}\|x_j - x_*\|_{A^T A}^2 \le (1 - \frac{\alpha \eta^2}{\beta})^j \|x_0 - x_*\|_{A^T A}^2, \quad (2)$$

where $\alpha \le \sigma_{\min}^2(A_\tau^\dagger)$, $\sigma_{\max}^2(A_\tau^\dagger) \le \beta$ and $\eta \in (0, 1]$.

Proof: From Algorithm 2, we know that $x_{j+1} = x_j + I_{\tau_j} A_{\tau_j}^\dagger r_j$. Thus

$$\begin{aligned} A(x_{j+1} - x_j) &= A_{\tau_j} A_{\tau_j}^\dagger (b - A x_j) \\ &= A_{\tau_j} A_{\tau_j}^\dagger (b_{R(A)} + b_{R(A)^\perp} - A x_j) \\ &= A_{\tau_j} A_{\tau_j}^\dagger (b_{R(A)} - A x_j) \\ &= A_{\tau_j} A_{\tau_j}^\dagger A (x_* - x_j), \end{aligned} \quad (3)$$

where we have used $A_{\tau_j}^{\dagger} b_{R(A)^{\perp}} = 0$ and $Ax_* = b_{R(A)}$. Hence $A(x_{j+1} - x_j)$ belongs to $\mathcal{R}(A_{\tau_j} A_{\tau_j}^{\dagger})$, the column space of $A_{\tau_j} A_{\tau_j}^{\dagger}$. On the other hand, since

$$\begin{aligned} A(x_{j+1} - x_*) &= A(x_j - x_* + I_{\tau_j} A_{\tau_j}^{\dagger}(b - Ax_j)) \\ &= A(x_j - x_*) + A_{\tau_j} A_{\tau_j}^{\dagger} A(x_* - x_j) \\ &= (I - A_{\tau_j} A_{\tau_j}^{\dagger}) A(x_j - x_*), \end{aligned}$$

then multiplying both sides of the above equality with $A_{\tau_j}^T$ yields

$$\begin{aligned} A_{\tau_j}^T A(x_{j+1} - x_*) &= A_{\tau_j}^T (I - A_{\tau_j} A_{\tau_j}^{\dagger}) A(x_j - x_*) \\ &= A_{\tau_j}^T A(x_j - x_*) - A_{\tau_j}^T A_{\tau_j} A_{\tau_j}^{\dagger} A(x_j - x_*) \\ &= A_{\tau_j}^T A(x_j - x_*) - A_{\tau_j}^T A(x_j - x_*) \\ &= 0. \end{aligned}$$

Thus $(A_{\tau_j}^{\dagger})^T A_{\tau_j}^T A(x_{j+1} - x_*) = 0$, i.e., $A(x_{j+1} - x_*)$ is orthogonal to $A_{\tau_j} A_{\tau_j}^{\dagger}$, which coupled with (3) shows that $A(x_{j+1} - x_j)$ is orthogonal to $A(x_{j+1} - x_*)$. By Pythagorean theorem, we obtain that

$$\|A(x_{j+1} - x_*)\|_2^2 = \|A(x_j - x_*)\|_2^2 - \|A(x_{j+1} - x_j)\|_2^2. \quad (4)$$

By taking the conditional expectation on both sides of (4), we obtain that

$$\begin{aligned} & \mathbb{E}_{j+1} \|A(x_{j+1} - x_*)\|_2^2 \\ = \; & \|A(x_j - x_*)\|_2^2 - E_{j+1} \|A(x_{j+1} - x_j)\|_2^2 \\ = \; & \|A(x_j - x_*)\|_2^2 - E_{j+1} \|(A_{\tau_j}^{\dagger})^T A_{\tau_j}^T A(x_j - x_*)\|_2^2 \\ = \; & \|A(x_j - x_*)\|_2^2 - \\ & \sum_{\tau_j \in U_j} \frac{|\bar{A}_{\tau_j}^T r_j|^2}{\sum_{\tau \in U_j} |\bar{A}_{\tau}^T r_j|^2} \|(A_{\tau_j}^{\dagger})^T A_{\tau_j}^T A(x_j - x_*)\|_2^2, \quad (5) \end{aligned}$$

where in the second equality we have used the property $(A_{\tau_j} A_{\tau_j}^{\dagger})^T = A_{\tau_j} A_{\tau_j}^{\dagger}$. Since $A_{\tau_j}^{\dagger} = (A_{\tau_j}^T A_{\tau_j})^{-1} A_{\tau_j}^T$, then

$$A_{\tau_j}^T A(x_j - x_*) \in \mathcal{R}(A_{\tau_j}^T) = \mathcal{R}(A_{\tau_j}^{\dagger}). \quad (6)$$

It follows from Lemma 2 and (6) that the equation (5) renders the inequality

$$\begin{aligned} & \mathbb{E}_{j+1} \|A(x_{j+1} - x_*)\|_2^2 \\ \leq \; & \|A(x_j - x_*)\|_2^2 - \\ & \sum_{\tau_j \in U_j} \frac{|\bar{A}_{\tau_j}^T r_j|^2}{\sum_{\tau \in U_j} |\bar{A}_{\tau}^T r_j|^2} \sigma_{\min}^2(A_{\tau_j}^{\dagger}) \|A_{\tau_j}^T A(x_j - x_*)\|_2^2 \quad (7) \end{aligned}$$

Denote by $\tilde{r}_j = A(x_* - x_j)$ with $\tilde{r}_j = \tilde{r}_{j_1} + \tilde{r}_{j_2}$, where $\tilde{r}_{j_1} \in \mathcal{R}(A_{\tau_j})$ and $\tilde{r}_{j_2} \perp \mathcal{R}(A_{\tau_j})$. Then $A_{\tau_j}^T \tilde{r}_{j_2} = 0$. Let $\|\tilde{r}_{j_1}\| = \eta_j \|\tilde{r}_j\|$ with $\eta_j \in (0, 1]$. Thus

$$\begin{aligned} & \|A_{\tau_j}^T A(x_j - x_*)\|_2^2 \\ = \; & \|A_{\tau_j}^T \tilde{r}_{j1}\|_2^2 \\ \geq \; & \sigma_{\min}^2(A_{\tau_j}) \|\tilde{r}_{j_1}\|_2^2 \\ = \; & \eta_j^2 \sigma_{\min}^2(A_{\tau_j}) \|A(x_j - x_*)\|_2^2. \quad (8) \end{aligned}$$

Combining (7) and (8) gives

$$\begin{aligned} & \mathbb{E}_{j+1} \|A(x_{j+1} - x_*)\|_2^2 \\ \leq \; & \|A(x_j - x_*)\|_2^2 - \\ & \sum_{\tau_j \in U_j} \frac{|\bar{A}_{\tau_j}^T r_j|^2}{\sum_{\tau \in U_j} |\bar{A}_{\tau}^T r_j|^2} \sigma_{\min}^2(A_{\tau_j}^{\dagger}) \sigma_{\min}^2(A_{\tau_i}) \eta_j^2 \|A(x_j - x_*)\|_2^2 \\ \leq \; & (1 - \frac{\alpha \eta_j^2}{\beta}) \|A(x_j - x_*)\|_2^2, \end{aligned}$$

where $\alpha \leq \sigma_{\min}^2(A_{\tau}^{\dagger})$, $\sigma_{\min}^{-2}(A_{\tau}) = \sigma_{\max}^2(A_{\tau}^{\dagger}) \leq \beta$ and $\eta_j \in (0, 1]$. Taking expectations on both sides gives

$$\mathbb{E}\|x_{j+1} - x_*\|_{A^T A}^2 \leq (1 - \frac{\alpha \eta_j^2}{\beta}) \|x_j - x_*\|_{A^T A}^2.$$

Let $\eta = \min_{1 \leq i \leq j} \{\eta_i\} \in (0, 1]$. Then the inequality (2) follows by induction on j. ■

Remark 1. *In Algorithm 2, if $k = n$, then GRBCD(k) reduces to GRCD. However, we stress that the column partitioning used in GRBCD(k) is effective in reducing the computing time and number of iterations. In fact, numerical examples in the next section show that GRBCD(k) often outperforms GRCD even for relatively small values of k, say $k \leq 10$; see Section IV for more numerical evidence.*

IV. Numerical Experiments

In this section, we compare GRBCD(k) with the randomized coordinate descent (RCD) [19], greedy randomized coordinate descent (GRCD) [5], and greedy block Gauss-Seidel (GBGS) [20] methods to illustrate its efficiency. All experiments were done on a laptop with AMD Ryzen 7 4800U Radeon Graphics @1.80 GHZ 16.0 GB RAM using MATLAB(R2022a). Numerical performance of the four algorithms is appraised regarding CPU time in seconds (CPU) and the number of iteration steps (IT). Specifically, CPU and IT denote the average quantities by running each of the four algorithms for five times. To delineate the advantage of GRBCD(k) over its counterparts, we consider the CPU speed-up, viz.,

$$\begin{aligned} \text{speed} - \text{up}_{\text{RCD}} &= \tfrac{\text{CPU of RCD}}{\text{CPU of GRBCD}(k)}, \\ \text{speed} - \text{up}_{\text{GRCD}} &= \tfrac{\text{CPU of GRCD}}{\text{CPU of GRBCD}(k)}, \\ \text{speed} - \text{up}_{\text{GBGS}} &= \tfrac{\text{CPU of GBGS}}{\text{CPU of GRBCD}(k)}. \end{aligned}$$

The initial guess is set as $x_0 = \mathbf{0}$. For a fair comparison, we follow the practice adopted in [20] by setting the parameter θ in the GBGS method to be 0.5. All experiments are terminated when the squared relative solution error (RSE) at the jth iteration is $\text{RSE} < 10^{-6}$ or when IT exceeds the maximum 200000 steps, where

$$\text{RSE} = \frac{\|x_j - x_*\|_2^2}{\|x_*\|_2^2}.$$

A symbol "–" is used to indicate that the underlying algorithm fails to reach the required accuracy within the above-mentioned conditions. For the linear least squares problem (1), the right-hand side vector b is defined as $b = Ax_* + r$, where the solution x_* is generated by the MATLAB function `rand` and the nonzero vector $r \in \text{null}(A^T)$ with $\text{null}(A^T)$ being figured out by invoking the MATLAB function `null`.

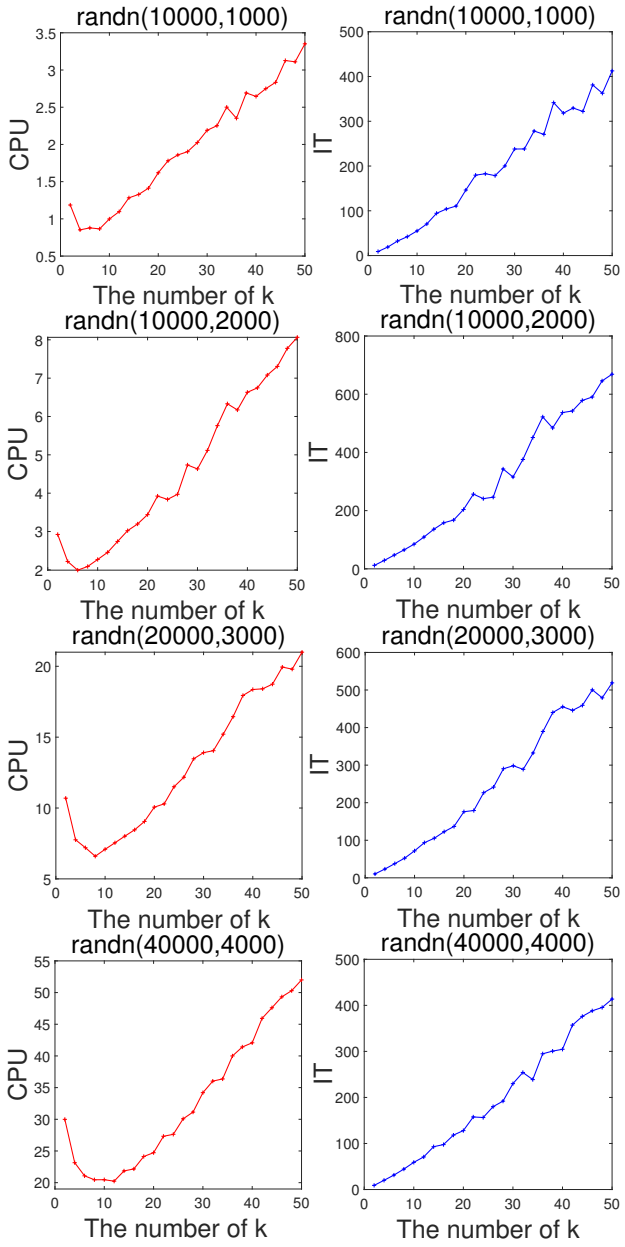

Fig. 1: CPU and IT against k for `randn` matrices.

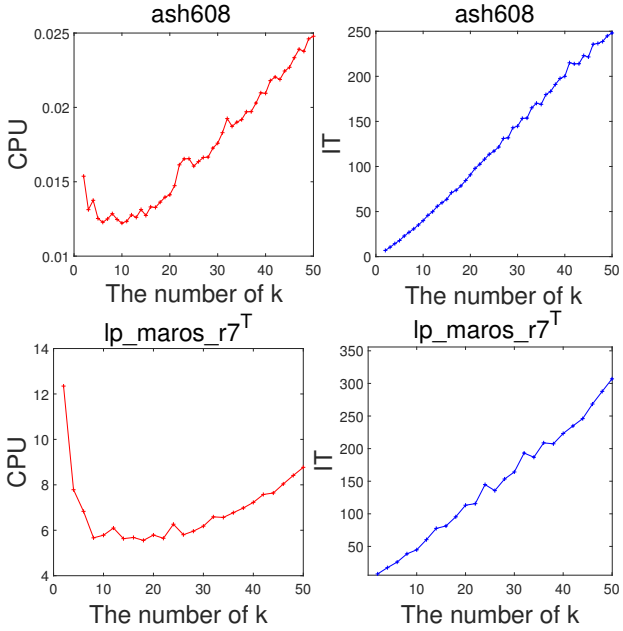

Fig. 2: CPU and IT against k for real-world problems.

TABLE I: Properties of the test matrices.

Matrix	Size	Density	cond(A)
`abtaha1`	14956×209	1.68%	12.23
`ash608`	608×188	1.06%	3.3729
`ash958`	958×292	0.68%	3.2014
`well1033`	1033×320	1.43%	166.1333
`cari`^{T}	1200×400	31.83%	3.1292
`rosen10`^{T}	6152×2056	0.51%	194.0816
`lp_maros_r7`^{T}	9408×3136	0.49%	2.3231

The coefficient matrix A in (1) is either given by the MATLAB built-in function `randn` or from some real-world problems [17]; see Table I for the matrix size, density and condition number of the test matrices.

This section is divided into two subsections. Each subsection serves a specific purpose. In Section IV-A, we investigate the impact of k on the performance of GRBCD(k) and come up with some practical choices of k. In Section IV-B, we validate the effectiveness of GRBCD(k) in comparison with RCD, GRCD and GBGS.

A. Choices of k for GRBCD(k)

In this subsection, we look at how the parameter k affects the performance of GRBCD(k) in terms of CPU and IT. Test problems with coefficient matrices generated by `randn` or from the SuiteSparse Matrix Collection [17] are employed to track suitable choices of k.

For least squares (1) with coefficient matrices given by `randn`, as illustrated in Figure 1, the CPU time often exhibits a V-shaped curve; It initially decreases as the value of k increases and then rises as k goes up. Nevertheless, the number of iteration steps generally increases as k grows. A similar observation is also witnessed for linear least squares problem (1) with the coefficient matrices from real-world applications; see Figure 2. It can be explained as follows. If k is small, or equivalently, the number of blocks obtained from the column partitioning is small, then one needs to deal with pseudoinverses of some large blocks at step 7 of Algorithm 2, which can be rather time-consuming. Conversely, if k is chosen to be large, then more iterations are demanded which in turn may increase the total CPU time. As noted in Remark 1, GRBCD(k) is mathematically equivalent to GRCD when $k = n$, which implies the former can converge much faster than the latter in terms of CPU and IT; see Section IV-B.

From Figures 1-2, we conclude that a moderate value of k suffices to yield fast convergence; for instance, choices with $k \leq 10$ often result in sound numerical performance regarding CPU. In light of this, we use $k \leq 10$ for GRBCD(k), say $k = 4$, 6, 8 and 10, when compared with other RCD-type algorithms in the following examples.

B. Numerical comparison of four algorithms

In this subsection, we compare GRBCD(k) with RCD, GRCD and GBGS in terms of CPU and IT.

The first example involves test problems (1) with coefficient matrices generated by the MATLAB function `randn`.

TABLE II: Numerical results for $\texttt{randn}(10000, n)$ with various values of n.

$m \times n$		10000×500	10000×1000	10000×2000	10000×3000	10000×4000
RCD	IT	6592.80	14902.00	37753.00	77381.00	144700.00
	CPU	21.1567	91.1659	457.3488	1390.4000	3461.0000
GRCD	IT	1339.80	3396.00	9821.40	22907.00	43129.00
	CPU	4.3390	21.9101	130.7259	422.0608	1047.8000
GBGS	IT	36.00	55.00	107.00	187.00	324.00
	CPU	0.5134	1.5442	4.3328	10.1941	21.8677
GRBCD(4)	IT	16.40	19.60	28.80	41.00	59.80
	CPU	**0.3178**	**0.7797**	2.4660	4.4643	7.6850
GRBCD(6)	IT	26.00	31.40	47.80	71.80	102.00
	CPU	0.3669	0.8221	**2.0009**	4.1355	**6.9736**
GRBCD(8)	IT	34.40	43.60	76.40	98.60	142.60
	CPU	0.3971	0.8760	2.1894	**3.9186**	6.9888
GRBCD(10)	IT	48.40	72.60	99.80	140.80	191.60
	CPU	0.4632	1.0874	2.4064	4.4296	7.4184
$\text{speed} - \text{up}_{\text{RCD}}$		66.5724	116.9243	228.5715	354.8206	496.3003
$\text{speed} - \text{up}_{\text{GRCD}}$		13.6532	28.1007	65.3335	107.7070	150.2523
$\text{speed} - \text{up}_{\text{GBGS}}$		1.6155	1.9805	2.1654	2.6015	3.1358

TABLE III: Numerical results for $\texttt{randn}(m, 2000)$ with various values of m.

$m \times n$		5000×2000	10000×2000	20000×2000	30000×2000	40000×2000
RCD	IT	74444.00	37778.00	29528.00	26103.00	25513.00
	CPU	456.6743	442.5031	731.1411	963.3075	1241.4000
GRCD	IT	22230.00	10109.00	6768.60	5809.60	5349.00
	CPU	138.2274	122.5926	179.1934	214.7894	255.7404
GBGS	IT	294.00	105.00	59.00	46.00	41.00
	CPU	5.4427	4.3901	5.6978	8.0676	10.2764
GRBCD(4)	IT	60.00	30.00	20.40	17.40	16.60
	CPU	**1.53**	2.2198	4.4185	6.2648	9.2155
GRBCD(6)	IT	98.20	48.60	33.00	26.60	23.80
	CPU	1.5425	**2.0189**	4.1049	**5.6995**	**8.3161**
GRBCD(8)	IT	160.80	65.60	45.00	39.80	34.60
	CPU	1.7328	2.1024	**3.8251**	5.8199	9.4648
GRBCD(10)	IT	244.20	90.80	53.60	52.00	49.00
	CPU	2.1400	2.3280	4.0465	6.4192	9.8974
$\text{speed} - \text{up}_{\text{RCD}}$		298.4799	219.1803	191.1430	169.0161	149.2767
$\text{speed} - \text{up}_{\text{GRCD}}$		90.3447	60.7225	46.8467	37.6857	30.7524
$\text{speed} - \text{up}_{\text{GBGS}}$		3.5573	2.1745	1.4896	1.4155	1.2357

Iteration steps and CPU time for varying values of n or m are tabulated in Tables II-III. In Table II, we run these four algorithms for different `randn` matrices with changing number of columns. As suggested in Section IV-A, the values $k = 4, 6, 8$ and 10 are picked for GRBCD(k). We are ready to give some remarks. Both GBGS and GRBCD(k) outperform RCD and GRCD regarding CPU and IT. Moreover, GRBCD(k) outstrips the others when k is chosen properly. Take the case 10000×4000 for example. The speed-up of GRBCD(6) over RCD is up to 496.3003. It should be noted that the speed-up in Table II is computed by the ratio of CPU time by the corresponding algorithm to that of shortest CPU time by GRBCD(k); for instance, in the case 10000×4000, the speed-up 3.1358 is figured out by the ratio of 21.8677 (CPU time GBGS) to 6.9736 (the shortest CPU by GRBCD(k)). Such advantage offered by GRBCD(k) becomes more pronounced as n grows. In Table III, we test these four algorithms for different `randn` matrices with varying number of rows. As reported in Table III, it is evident that the column partitioning brought by k-means clustering works well in reducing CPU and IT; for example, the speed-ups of GRBCD(4) over RCD, GRCD and GBGS are respectively 298.4799, 90.3447 and 3.5573 for the 5000×2000 case.

In the second example, we compare GRBCD(k) with RCD, GRCD and GBGS for test matrices from real-world problems, as shown in Table IV. Note that the superscript "T" in Table IV denotes the transpose of the associated matrix; for example, $\texttt{cari}^T$ stands for the transpose of the matrix `cari`. In Table IV, analogous to the previous example for `randn` matrices, GRBCD(k) exceeds its counterparts for all test matrices in terms of CPU time. Furthermore, GRBCD(k) and GBGS succeed even when RCD or GRCD fail to converge within the maximum number of iterations; see, for instance, the cases `well1033` and $\texttt{rosen10}^T$ in Table IV.

TABLE IV: Numerical results for test matrices from real-world problems.

name		abtaha1	ash608	ash958	well1033	cari^T	lp_maros_r7^T	rosen10^T
RCD	IT	92385.00	3520.40	5930.80	–	5198.40	45077.00	–
	CPU	191.6012	0.1488	0.5942	–	0.9304	806.6439	–
GRCD	IT	13655.00	666.40	1004.40	–	1203.20	10565.00	30053.00
	CPU	28.6108	0.0424	0.1249	–	0.3536	196.2309	250.3977
GBGS	IT	974.00	54.00	55.00	97634.00	70.00	92.00	806.00
	CPU	8.2779	0.0154	0.0429	37.0455	0.1083	5.7758	13.4299
GRBCD(2)	IT	139.40	6.92	6.90	9288.80	14.00	7.80	69.40
	CPU	**0.5147**	0.0154	0.0410	3.2717	0.0529	12.3468	3.9513
GRBCD(4)	IT	453.30	14.42	13.80	8472.20	34.00	17.40	159.80
	CPU	1.0618	0.0138	**0.0382**	**2.2444**	**0.0482**	7.7849	**3.9387**
GRBCD(6)	IT	726.40	22.86	21.30	30673.80	50.00	26.00	218.20
	CPU	1.3493	0.0123	0.0395	6.1613	0.0547	6.8316	4.2800
GRBCD(8)	IT	1306.30	30.78	28.90	46686.20	70.30	38.60	259.00
	CPU	2.1397	0.0129	0.0389	8.9176	0.0579	**5.6665**	4.1988
GRBCD(10)	IT	1637.90	39.98	37.20	76588.20	80.40	44.80	309.00
	CPU	2.5633	**0.0122**	0.0408	14.1778	0.0618	5.7824	4.5684
speed $-$ up$_{\mathrm{RCD}}$		372.2580	12.1967	15.555	–	19.3029	142.3531	–
speed $-$ up$_{\mathrm{GRCD}}$		55.5873	3.4754	3.2696	–	7.3361	34.6300	63.5736
speed $-$ up$_{\mathrm{GBGS}}$		16.0830	1.2623	1.1230	16.5057	2.2469	1.0193	3.4097

We wrap up this section by concluding that GRBCD(k) can be very competitive and is suitable for solving large linear least squares problem with large number of columns.

V. Conclusions

We present a greedy randomized block coordinate descent algorithm for solving large linear least squares problems. The k-means clustering algorithm is used to extract column blocks from the coefficient matrix. We establish the convergence result for the proposed approach and demonstrate that it is very promising in comparison with some other randomized coordinate descent methods. Such advantage becomes more pronounced when the number of columns in the coefficient matrix is relatively large.

Finally, we stress that the selection of k in GRBCD(k) merits further investigation, despite the fact that automatically determining the number of clusters has been one of the most difficult problems in cluster analysis. In fact, as already stated in [15], more sophisticated statistical techniques [1], [14] can be exploited to yield sound choices of k.

Acknowledgement

The authors thank Xiang-Xiang Chen and Hong-Yan Yin of Shanghai Maritime University for helpful discussions.

References

[1] C. C. Aggarwal and C. K. Reddy, "Data clustering: algorithms and applications," *CRC Press*, 2014.

[2] Z.-Z. Bai and L. Wang, "On convergence rates of Kaczmarz-type methods with different selection rules of working rows," *Appl. Numer. Math.*, vol. 186, pp. 289-319, 2023.

[3] Z.-Z. Bai, L. Wang, and W.-T. Wu, "On convergence rate of the randomized Gauss-Seidel method," *Linear Algebra Appl.*, vol. 611, pp. 237-252, 2021.

[4] Z.-Z. Bai and W.-T. Wu, "On greedy randomized Kaczmarz method for solving large sparse linear systems," *SIAM J. Sci. Comput.*, vol. 40, no. 1, pp. A592-A606, 2018.

[5] Z.-Z. Bai and W.-T. Wu, "On greedy randomized coordinate descent methods for solving large linear least-squares problems," *Numer. Linear Algebra Appl.*, vol. 26, no. 4, pp. 1-15, 2019.

[6] Z.-Z. Bai and W.-T. Wu, "On partially randomized extended Kaczmarz method for solving large sparse overdetermined inconsistent linear systems," *Linear Algebra Appl.*, vol. 578, pp. 225-250, 2019.

[7] Z.-Z. Bai and W.-T. Wu, "Randomized Kaczmarz iteration methods: Algorithmic extensions and convergence theory," *Jpn. J. Ind. Appl. Math.*, vol. 40, pp. 1421-1443, 2023.

[8] R. Bru, J. Marín, J. Mas, and M. Tůma, "Preconditioned iterative methods for solving linear least squares problems," *SIAM J. Sci. Comput.*, vol. 36, no.4, pp. A2002-A2022, 2014.

[9] Å. Björck, "Numerical methods for least squares problems," *SIAM*, Philadelphia, 1996.

[10] J.-Q. Chen and Z.-D. Huang, "A fast block coordinate descent method for solving linear least-squares problems," *East Asian J. Appl. Math.*, vol. 12, no. 2, pp. 406-420, 2022.

[11] K. Du, "Tight upper bounds for the convergence of the randomized extended Kaczmarz and Gauss–Seidel algorithms," *Numer. Linear Algebra Appl.*, vol. 26, no. 3, pp.1-14, 2019.

[12] B. Dumitrescu, "On the relation between the randomized extended Kaczmarz algorithm and coordinate descent," *BIT Numer. Math.*, vol. 55, pp. 1005-1015, 2015.

[13] R. El Jid, "Moving least squares and Gauss Legendre for solving the integral equations of the second kind," *IAENG International Journal of Applied Mathematics*, vol. 49, no.1, pp. 90-97, 2019.

[14] A. K. Jain, "Data clustering: 50 years beyond k-means," *Pattern Recognit. Lett.*, vol. 31, no. 8, pp. 651-666, 2010.

[15] X.-L. Jiang, K. Zhang, and J.-F. Yin, "Randomized block Kaczmarz methods with k-means clustering for solving large linear systems," *J. Comput. Appl. Math.*, vol. 403, no. 113828, pp. 1-14, 2022.

[16] S. Karczmarz, "Angenäherte Auflösung von systemen linearer gleichungen, "*Bull. Pol. Acad. Sci. Lett. Ser. A*, pp. 335-357, 1937.

[17] S. P. Kolodziej, M. Aznaveh, M. Bullock, J. David, T. A. Davis, M. Henderson, Y. Hu, and R. Sandstrom, "The SuiteSparse matrix collection website interface," *J. Open Source Softw.*, vol. 4, pp. 1244-1247, 2019.

[18] C. L. Lawson and R. J. Hanson, "Solving least squares problems," *SIAM*, Philadelphia, 1995.

[19] D. Leventhal and A. S. Lewis, "Randomized methods for linear constraints: convergence rates and conditioning," *Math. Oper. Res.*, vol. 35, no. 3, pp. 641-654, 2010.

[20] H. Li and Y. Zhang, "Greedy block Gauss-Seidel methods for solving large linear least squares problems," https://arxiv.org/abs/2004.02476, pp. 1-14, 2020.

[21] Y. Liu and C. Gu, "Two greedy subspace Kaczmarz algorithm for image reconstruction," *IAENG International Journal of Applied Mathematics*, vol. 50, no. 4, pp. 853-859, 2020.

[22] A. Ma, D. Needell, and A. Ramdas, "Convergence properties of the randomized extended Gauss–Seidel and Kaczmarz methods," *SIAM J. Matrix Anal. Appl.*, vol. 36, no. 4, pp. 1590-1604, 2015.

[23] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," *Proc. Fifth Berkeley Symp. on Math. Statist. and Prob.*, 1, Univ. of Calif. Press, pp. 281-297, 1967.
[24] K. S. Sim, F. F. Ting, J. W. Leong, and C. P. Tso, "Signal-to-noise ratio estimation for SEM single image using cubic spline interpolation with linear least square regression," *Engineering Letters*, vol. 27, no. 1, pp. 151-165, 2019.
[25] T. Strohmer and R. Vershynin, "A randomized Kaczmarz algorithm with exponential convergence," *J. Fourier Anal. Appl.*, vol. 15, pp. 262-278, 2009.
[26] L.-Z. Tan and X.-P. Guo, "On multi-step greedy randomized coordinate descent method for solving large linear least-squares problems," *Comput. Appl. Math.*, vol. 42, no. 37, pp. 1-20, 2023.
[27] C. Wang, C. Che, and X. Xia, "Oblique QR decomposition based block partition strategy for block Landweber scheme," *IAENG International Journal of Applied Mathematics*, vol. 53, no.3, pp. 884-891, 2023.
[28] S. J. Wright, "Coordinate descent algorithms," *Math. Program. Ser. B*, vol. 151, pp. 3-34, 2015.
[29] J. Zhang and J. Guo, "On relaxed greedy randomized coordinate descent methods for solving large linear least-squares problems," *Appl. Numer. Math.*, vol. 157, pp. 372-384, 2020.
[30] K. Zhang, F.-T. Li, and X.-L. Jiang, "Multi-step greedy Kaczmarz algorithms with simple random sampling for solving large linear systems," *Comp. Appl. Math.*, vol. 41, no. 332, pp. 1-25, 2022.
[31] K. Zhang, H.-Y. Yin, and X.-L. Jiang, "An efficient variant of the greedy block Kaczmarz algorithm for solving large linear systems," *AIMS Math.*, vol. 9, no. 1, pp. 2473-2499, 2024.
[32] A. Zouzias and N. M. Freris, "Randomized extended Kaczmarz for solving least squares," *SIAM J. Matrix Anal. Appl.*, vol. 34, no. 2, pp. 773-793, 2013.