Journal of Scientific Computing (2023) 96:86
https://doi.org/10.1007/s10915-023-02307-1

Conjugate Gradients Acceleration of Coordinate Descent for Linear Systems

Dan Gordon[1]

Received: 13 December 2022 / Revised: 24 May 2023 / Accepted: 21 July 2023

Abstract
This paper introduces a conjugate gradients (CG) acceleration of the coordinate descent algorithm (CD) for linear systems. It is shown that the Kaczmarz algorithm (KACZ) can simulate CD exactly, so CD can be accelerated by CG similarly to the CG acceleration of KACZ (Björck and Elfving in BIT 19:145–163, 1979). Experimental results were carried out on large sets of problems of reconstructing bandlimited functions from random sampling. The randomness causes extreme variance between different instances of these problems, thus causing extreme variance in the advantage of CGCD over CD. The reduction of the number of iterations by CGCD varies from about 50–90% and beyond. The implementation of CGCD is simple. CGCD can also be used for the parallel solution of linear systems derived from partial differential equations, and for the efficient solution of multiple right-hand-side problems and matrix inversion.

Keywords Coordinate descent · CD · CGCD · CGMN · Conjugate gradients acceleration · Gauss–Seidel · Kaczmarz algorithm · Linear systems · Matrix inversion · Multiple right-hand-sides · Parallelism

Mathematics Subject Classification 65B99 · 65F10 · 65F20

1 Introduction

The coordinate descent algorithm (CD) is a well-known and widely used optimization algorithm; see, for example, [13, 16] for details and applications. CD is also used for solving linear systems as follows: given a linear system $Ax = b$, each step of CD consists of choosing a coordinate x_i and minimizing $\|Ax-b\|^2$ w.r.t. x_i by setting $\frac{\partial}{\partial x_i}\|Ax-b\|^2 = 0$ and extracting a new value for x_i. This approach is generally referred to as "CD for linear systems". There are several papers on various acceleration methods of CD, such as [1, 9, 10], and specifically for linear systems, see [8, 15]. CD for linear systems is identical to the Gauss–Seidel iteration on

✉ Dan Gordon
gordon@cs.haifa.ac.il

[1] Department of Computer Science, University of Haifa, 34988 Haifa, Israel

Published online: 05 August 2023

the system $A^T Ax = A^T b$; see for example [11, p. 594] and [16, Sect 1.4]. $A^T Ax = A^T b$ is also called NR (for "normal" and "residual" associated with the minimization of $\|b - Ax\|_2$; see Saad [12, Sect. 8.1]. In the following, "CD" will refer to CD for linear systems.

The rest of the paper is organized as follows. Section 2 presents the Kaczmarz algorithm (KACZ) [7] and shows the geometric identity between KACZ and CD. This is followed by the CG acceleration of KACZ, called CGMN, developed by Björck and Elfving [2], and the proof that this acceleration can be applied to CD. Section 3 presents the CG acceleration of CD and Sect. 4 presents the experiments. Section 5 presents an efficient construction of the pre-calculations for sparse systems, and Sect. 6 concludes with a summary and suggestions for the efficient solution of multiple right-hand-sides problems and matrix inversion.

2 Kaczmarz, CD and Their CG Acceleration

Consider a consistent system of linear equations

$$Ax = b. \tag{1}$$

KACZ operates as follows: starting from an arbitrary point as the initial iterate, that point in space is orthogonally projected towards a plane defined by one of the equations, thus creating a new iterate. This process is continued sequentially for all the following equations, and then repeated until some stopping criterion is satisfied. There are several variations of this procedure, such as randomized selection of the equations and a symmetric approach in which the equations are alternately handled in a forward and backward sequence.

Let x^0 be a chosen initial iterate. For $k \geq 0$, x^{k+1} is obtained from x^k by projecting x^k orthogonally towards the hyperplane determined by the kth equation, i.e.,

$$x^{k+1} = x^k + \lambda \frac{b_k - \langle a^k, x^k \rangle}{\|a^k\|_2^2} a^k, \tag{2}$$

where a^k is the kth matrix row, b_k is the kth RHS element, and $0 < \lambda < 2$ is a relaxation parameter that determines whether the projection is exactly on the hyperplane determined by the equation ($\lambda = 1$), or before or after the hyperplane. It is more efficient to initially divide every equation by the L_2 norm of its corresponding row, thus avoiding the division by $\|a^k\|_2^2$ at every step. KACZ is also known as successive overrelaxation (SOR) on the system $AA^T y = b,\ x = A^T y$. Successive SOR (SSOR) is KACZ carried out in the forward direction and then in the backward direction. The robustness of KACZ follows from the following observation: after all the equations are normalized, the diagonal of AA^T consists of ones, while the off-diagonal elements are all smaller than one. In a consistent system, the KACZ iterates converge towards a solution (there may be many solutions if the system is under-determined). We can now prove the following:

Lemma 1 *Let CD operate on a consistent system of equations whose solution is $x_1^*, \ldots, x_n^*$. Then, KACZ can be applied (in parallel to CD) to the system of equations $x_1 = x_1^*, \ldots, x_n = x_n^*$, with relaxation parameters that bring each iterate of KACZ to the same position in space that CD brings it.*

Proof Let $(x_1^k, \ldots, x_n^k)$ be some current value of the coordinates obtained by CD, and consider one step of CD that changes some coordinate x_i^k to a new value x_i^{k+1}. x_i^{k+1} is now closer to x_i^* than x_i^k, so the change of the ith coordinate moves the current iterate $(x_1^k, \ldots, x_n^k)$ closer

to the plane $x_i = x_i^*$. Only the ith coordinate is changed so the direction of this movement is orthogonal to the plane $x_i = x_i^*$.

Consider now a specific KACZ operation on the system $x_1 = x_1^*, \ldots, x_n = x_n^*$, running in parallel to CD. The initial iterate of KACZ is taken as the initial iterate of CD. For each step of CD on a coordinate x_i^k, KACZ performs one step on the equation $x_i = x_i^*$, and the relaxation parameter is chosen to bring the new iterate to the same position that CD brings it. Clearly, KACZ produces exactly the same sequence of iterates as CD. □

In a seminal paper, Björck and Elfving [2] showed that KACZ can be accelerated by CG, and called this algorithm CGMN. Based on Lemma 1, we have the following:

Theorem 1 *CD can be accelerated by CG.*

Proof The original CGMN [2] assumed a constant relaxation parameter, but now we need to accelerate KACZ with variable relaxation parameters as in Lemma 1. In [6, Sect. 3] it was shown that the CG acceleration of KACZ can use different relaxation parameters for different equations, and this enabled a parallel version of CGMN (called CARP-CG) for solving huge linear systems derived from partial differential equations. See also [5, Sect. 2]. It is easy to see from [6, Sect. 3] that the cyclic relaxation parameters can be replaced by any relaxation parameter at every instance of Eq. (2). It now follows from Lemma 1 that the CG acceleration of (the specific) KACZ can be applied to CD. □

The following are some related comments regarding CG. Given the system (1), consider the system NR mentioned previously:

$$A^T Ax = A^T b \tag{3}$$

The system (3) can also be solved directly by CG, and this is known as CGNR; see [12, Sect. 8.3.1]. There is also the CGNE algorithm, also known as Craig's Method, which is a CG acceleration of the system $AA^T u = b,\ x = A^T u$, which is referred to as NE (normal equations); see [12, Sect. 8.3.2]. In [4], CGNR and CGNE produced identical results on the problem of reconstructing bandlimited functions from random sampling, but they were significantly worse than CGMN, as can be seen from [4, Figs. 4 and 6].

3 CG Acceleration of CD

CD and its CG acceleration will now be presented in detail. Let A be a matrix of size $m \times n$ (m rows by n columns), and b the right-hand side. Our purpose is to find a vector $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ such that $\|Ax - b\|$ is minimal. CD proceeds by successively setting $\frac{\partial}{\partial x_i}\|Ax - b\|^2 = 0$, for $i = 1, \ldots, n$. In the following, the dot product of two vectors is represented by "•". We introduce some notations:

- A_i denotes the ith column of A.
- $A_i^j = A_i \bullet A_j$ if $i \neq j$, else 0.
- $A_i^* = (A_i^1, \ldots, A_i^n)$. (Note that the ith element, A_i^i, is zero.)

In order to save some computation time, the columns of A are normalized, i.e., each column is divided by its L_2-norm. It is easy to see that setting $\frac{\partial}{\partial x_i}\|Ax - b\|^2 = 0$ produces the following iteration: $x_i = A_i \bullet b - x \bullet A_i^*$. The CD algorithm is the following:

CD: The coordinate descent algorithm

1. compute all the terms $A_i \bullet b$ and A_i^*;
2. **set** $x \in \mathbb{R}^n$ to an arbitrary value;
3. **repeat** until some termination goal is reached:
4. **for** $i = 1$ to n:
5. $x_i = A_i \bullet b - x \bullet A_i^*$;
6. **end repeat**

Note that in the term $x \bullet A_i^*$ in line 5, x_i is multiplied by zero, so x_i does not appear in the RHS.

In the following CGCD, $\alpha, \beta, \gamma, \delta, \delta 1$ are auxiliary variables and p, q, r are vectors of size n.

CGCD: CG acceleration of CD:

1. compute all the terms $A_i \bullet b$ and A_i^* for $1 \le i \le n$;
2. **set** $x = r \in \mathbb{R}^n$ to an arbitrary value;
3. *forward and backward loops with b*:
4. **for** $i = 1$ to n:
5. $r_i = A_i \bullet b - r \bullet A_i^*$;
6. **for** $i = n$ downto 1:
7. $r_i = A_i \bullet b - r \bullet A_i^*$;
8. $p = r = r - x$;
9. $\delta = r \bullet r$;
10. **CG iterations:**
11. **repeat** until some termination goal is reached:
12. set $q = p$;
13. *forward and backward loops without b*:
14. **for** $i = 1$ to n:
15. $q_i = -q \bullet A_i^*$;
16. **for** $i = n$ downto 1:
17. $q_i = -q \bullet A_i^*$;
18. $q = p - q \;\; ; \;\; \gamma = p \bullet q$;
19. $\alpha = \delta / \gamma \;\; ; \;\; x = x + \alpha p$;
20. $r = r - \alpha q \;\; ; \;\; \delta 1 = r \bullet r$;
21. $\beta = \delta 1 / \delta$;
22. $p = r + \beta * p \;\; ; \;\; \delta = \delta 1$;
23. **end repeat**

Consider now the complexity question: does CG impose a great burden on the coordinate descent? Note that the computations of A_i^j and A_i^* are used for both CD and CDCG, so they are not considered in this comparison. From the CD algorithm, we can see that the number of products of one iteration in the loop of lines 4–5 is of order $O(n^2)$, and so is the number of products (in one iteration) of the loops of CGCD (lines 15–16 and 18–19). In contrast, the number of products in one CG iteration needed for the CG in CGCD is only $5n$, and this is negligible. In comparing the number of iterations required by CD and CGCD, for the same problem, each iteration of CGCD is counted as two iterations, due to the two loops in CGCD.

Note that both CD and CGCD require the pre-computation of two sets of constants:

$$AB = \{A_i \bullet b \mid 1 \le i \le n\} \tag{4}$$

$$AA = \{A_i^* \mid 1 \le i \le n\} \tag{5}$$

The space required by AB is $O(nm)$, and the space of AA, in the case of a full rank matrix, is $O(n^2 m)$. The details for constructing AB and AA for sparse systems are presented in Sect. 5.

4 Experiments

4.1 Background

CD and CGCD were tested on the problem of recovering bandlimited signals from random sampling. A well-known paper on this topic is due to Strohmer and Vershynin [14], who got excellent results on this problem by using the Kaczmarz algorithm. Every sample contributes one equation to the problem. Before applying KACZ, the equations were sorted and every equation was assigned a probability weight proportional to the average distance of the sample from its nearest points. This concept is due to Feichtinger and Gröchenig [3]. In the experiments of [14], KACZ was applied randomly to the equations according to the weights. The experiments were done on a problem of 101 variables and 700 equations (samples). For convenience, the ratio between equations and variables will be denoted as EVR, so EVR≈ 7 in this problem.

This problem was also studied by the author in [4], where it turned out that when EVR was smaller than 5, the number of projections required by KACZ rises (seemingly) exponentially; see the KACZ plots in [4, Fig. 3]. However, CGMN provided excellent results for $2 \le \text{EVR} \le 6$.

The problem of recovering bandlimited signals from random samples was chosen as a testbed for comparing CD and CGCD due to the extreme variance between different instances of seemingly identical problems. The experimental problem that is being solved appears in [14, Sect. 4.1], and also in [4, Sect. 3.1]. It can be described as follows:

A bandlimited function $f(t)$ is known to be of the form

$$f(t) = \sum_{\ell=-r}^{r} x_\ell e^{2\pi i \ell t} \tag{6}$$

where t is time, $x_\ell \in \mathbb{C}$, r is a known positive integer (called the "bandwidth"), n is an integer set as $2r+1$, $m \ge n$, and $f(t_j)$ for $t_j \in \mathbb{R}$ and $1 \le j \le m$ are known samples of the values of $f(t)$ taken at irregular points in time. Our problem is to determine the values of $x_1, \ldots, x_n$ which will provide the values of $f(t)$ at any point in time. The linear system that provides the values of $x_1, \ldots, x_n$ is obtained by setting

$$f(t_j) = \sum_{\ell=-r}^{r} x_\ell e^{2\pi i \ell t_j}, \quad \text{for} \quad 1 \le j \le m. \tag{7}$$

The main problem of irregular sampling is that large gaps between samples make it more difficult to determine the values of x_i, hence the large computational differences between different instances of this problem; see [4, Sect. 4.1] for the effects of large gaps. Note that every instance of setting up a linear system is derived from randomly selecting the values of $t_1, \ldots, t_m$; if these values are spread quite evenly, then CD will converge fairly quickly, otherwise, CD converges slowly, and this is where CGCD has a big advantage.

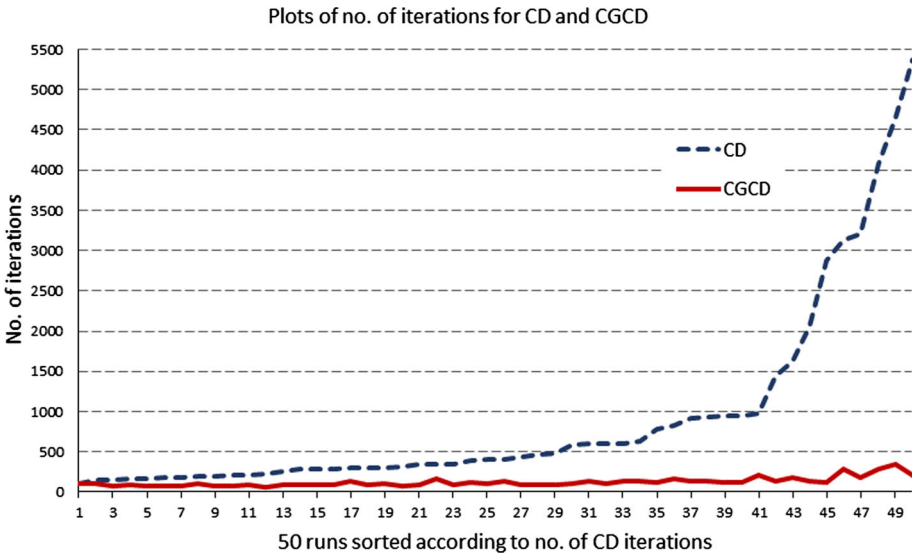

Fig. 1 Comparison of CD and CGCD on the same 50 problems. The results are sorted according to increasing number of iterations required by CD

4.2 Results

The only way to compare CD and CGCD is to run a large number of experiments in which every instance is solved by both CD and by CGCD. Let R denote the ratio of the number of iterations required by CGCD divided by the number of iterations required by CD on exactly the same problem. The experiments show that R can vary in different cases from about 0.03 up to about 0.5. In many cases CGCD provides a very significant improvement over CD. The C program for these experiments, called cgcd.c, is available for download[1]; the program is self-explanatory.

Figures 1 and 2 show comparisons between the number of iterations required by CD and CGCD for 50 and 100 runs, respectively. The relative residual goal was set at 10^{-13} and the maximum number of iterations was set at 100,000. The results are sorted according to increasing numbers of iterations required by CD. The extreme variance stands out clearly, and in other experiments, some CD runs required significantly more iterations. The statistics show that the average number of iterations required by CGCD is around 13% of the average of CD. The most significant result of these experiments is that while about 20% of the CD runs required thousands of iterations, CGCD did not need more than 350 iterations. Some of the runs fail to reach the prescribed relative residual goal, and the failure percentage is typically around 2% for CD and 4% for CGCD, but it may vary widely. The consequence is that CGCD is somewhat less stable than CD on some of these particular problems, so in practice, it may be useful to run both versions in parallel.

[1] https://cs.haifa.ac.il~gordon/cgcd.c.

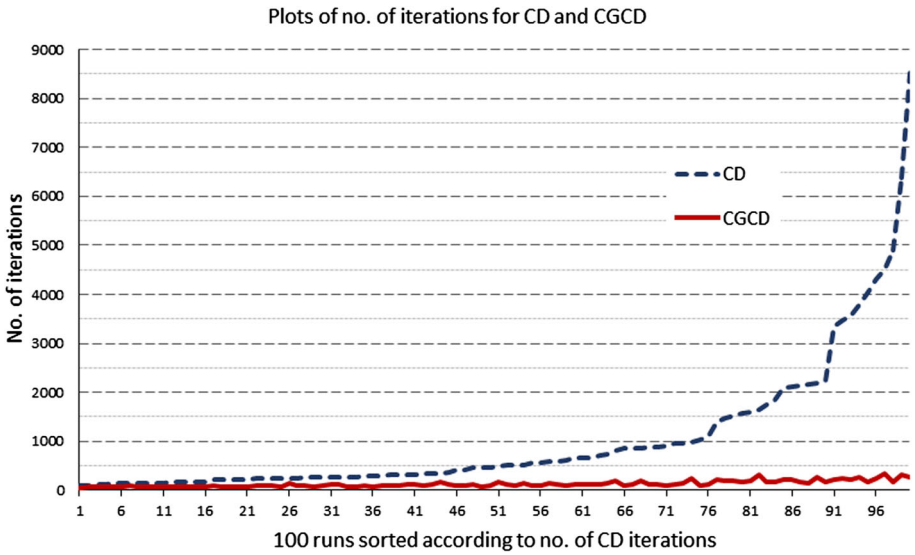

Fig. 2 Comparison of CD and CGCD on the same 100 problems. The results are sorted according to increasing number of iterations required by CD

5 Construction of *AB* and *AA* for Sparse Systems

The issue of setting up AB and AA, Eqs. (4) and (5), needs some explanation. If the system matrix is full then AB and AA can be calculated easily as arrays, as in the above experiments. Otherwise, AB and AA can be calculated as explained below. An important issue here is that in each row of A, the nonzero elements can be accessed easily, i.e., without traversing the entire row. This should not be a problem because every row is made up by the parameters of a single equation and the sequential access can be enabled when the matrix is formed. Also, in huge matrices, it is customary to have some compact form of the non-zero coefficients of a matrix row.

Consider first the data structure required by AB. $A_j \bullet b$ is the dot product of column A_j and b. We compute $A_j \bullet b$ for every j as follows: for every $1 \leq j \leq n$, a variable AB_j is set to zero. Next, for every matrix row i, we access all its non-zero elements, e.g., $a_{i,5}$, $a_{i,50}$, $a_{i,90}$, multiply every one of them by b_i, and add these products to their corresponding AB_j, e.g., $AB_5 \mathrel{+}= a_{i,5} \times b_i$, $AB_{50} \mathrel{+}= a_{i,50} \times b_i$, and $AB_{90} \mathrel{+}= a_{i,90} \times b_i$.

The data structure required by AA consists of the sequences $A_i^* = (A_{i,1}, \ldots, A_{i,n})$, for $1 \leq i \leq n$. Every $A_{i,j}$ is $A_i \bullet A_j$ if $i \neq j$ and 0 otherwise. Consider the following approach: for every A_i^*, a linked list is constructed with a header $[i|A_i^*]$. The pointer of the header is initially set to null. This list will contain the non-zero elements of $A_i \bullet A_j$ together with the index j, and ordered by increasing values of j; for example:

$$[i|A_i^*] \rightarrow [5|A_i \bullet A_5] \rightarrow [12|A_i \bullet A_{12}] \rightarrow \cdots \rightarrow [340|A_i \bullet A_{340}] \tag{8}$$

This structure will enable the efficient calculation of dot products such as $q \bullet A_i^*$ in lines 16 and 19 of CGCD.

The construction of a list as in Eq. (8) proceeds as follows: We traverse the rows in sequence, for $1 \leq k \leq m$. The non-zero elements of row 1 are the parameters of one equation, e.g., $a_{1,5}$, $a_{1,20}$, and $a_{1,70}$. We now initiate 3 constants to zero, each constant labeled by a pair

of the nonzero parameters: $A_{5,20}$, $A_{5,70}$, and $A_{20,70}$. We now compute all possible products of the parameters and add them to the 3 constants that were set up, according to the indices of the constants $A_{i,j}$:

$$\begin{aligned} A_{5,20} &+= a_{1,5} \times a_{1,20} \\ A_{5,70} &+= a_{1,5} \times a_{1,70} \\ A_{20,70} &+= a_{1,20} \times a_{1,70} \end{aligned} \tag{9}$$

We now add the three elements $A_{i,j}$ of Eq. (9) to the corresponding lists of A_5^*, A_{20}^*, A_{70}^*—see Eq. (8). Every $A_{i,j}$ in the lists must carry the column number which differs from the header, and the lists are sorted by the indices:

$$\begin{aligned} [5|A_5^*] &\rightarrow [20|A_{5,20}] \rightarrow [70|A_{5,70}] \\ [20|A_{20}^*] &\rightarrow [5|A_{5,20}] \rightarrow [70|A_{20,70}] \\ [70|A_{70}^*] &\rightarrow [5|A_{5,70}] \rightarrow [20|A_{20,70}] \end{aligned} \tag{10}$$

Note that $A_{i,j} = A_{j,i}$. Continuing with the following rows, we get two types of new products. For example, if in row 2 we get a product $c = a_{2,5} \times a_{2,70}$, then c should be added to $A_{5,70}$ in the list of A_5^* and also to $A_{5,70}$ in the list of A_{70}^*. Otherwise, new elements of type $A_{i,j}$ are created, the remaining constants are added to them as above, and they are inserted into the corresponding lists $[i|A_i]$ as in Eq. (9). Note that if there are k non-zero elements in a row, then there will be $\binom{k}{2}$ products.

At the end, the linked lists can be converted to arrays to save some computation time. In some cases, the sizes of the lists can be quite large, so it would be more efficient to use binary trees instead of linked lists because the insertion time is $\log_2$ of the time in lists. At the end, the inorders of the trees need to be converted to linked lists or arrays.

6 Conclusions and Further Applications

It has been shown that the CG acceleration of coordinate descent for linear systems, denoted CGCD, can be significantly faster than plain CD, so it can benefit applications that already use CD for linear systems, and perhaps other applications for which CD is too slow.

CGCD can also be useful for the problematic issue of multiple right-hand-sides because the time consuming setup of AA depends only on the matrix, and setting up new structures for AB is simple. As a consequence, CGCD can also deal with matrix inversion efficiently by successively choosing right-hand-sides as $(1, 0, \ldots, 0)$, $(0, 1, 0, \ldots, 0)$, $\ldots$, $(0, \ldots, 0, 1)$. In a parallel setting, after AA has been set up, the computations with the various RHSs can be distributed among several processors.

Parallelism for sparse matrices: in a full-rank matrix, such as in the experiments of this paper, CD (and CGCD) cannot be paralleled because CD is inherently sequential. In a sparse matrix, CD can be parallelize as follows: we define two variables x_i and x_j as *dependent* if they appear together in some equation, and otherwise they are *independent*. We will first show that if x_i and x_j are independent, then they can be handled in parallel. Consider the dot product $A_i \bullet A_j = 0$ and also line 5 of the CD algorithm $x_i = A_i \bullet b - x \bullet A_i^*$: the vector A_i^* is defined as $(A_{i,1}, \ldots, A_{i,n})$ and $A_{i,j} = A_i \bullet A_j = 0$. Hence, x_j has no effect on the new value of x_i (obtained from line 5), and vice versa. It follows that the two updates of x_i and x_j can be done in parallel.

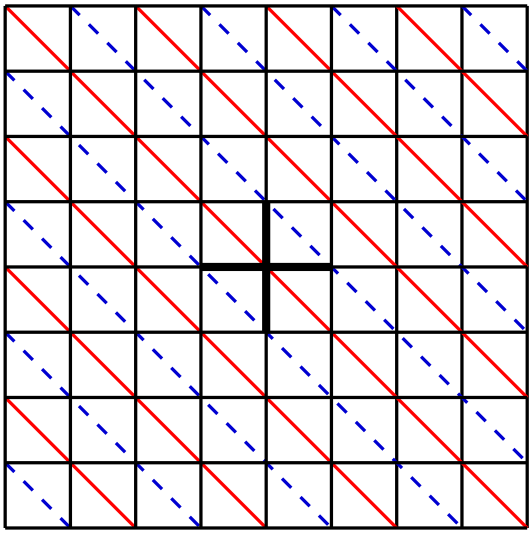

Fig. 3 Coloring a 2D grid with diagonal lines of two colors for 5-point stencils for PDE problems

To deal with dependent variables, we need to divide the variables into disjoint sets $S_1, ..., S_k$, which will be handled sequentially, and, if a variable x_i is in set S_j, then all its dependent variables belong to different sets. This way, all the required updates of a set can be done in parallel. The number of sets should be as small as possible. As an example, consider the problem of solving a partial differential equation in two dimensions using a finite difference, second-order scheme, in which every equation consists of 5 variables forming a 5-point stencil on the grid—see the central cross in Fig. 3. The division of the variables into two sets is shown by the diagonal red (solid) and blue (dashed) lines. Note, for example, that the central grid point is in the red set and all its 4 neighbors are in the blue set. A 3×3 stencil for higher order accuracy will require three sets.

Acknowledgements The author is grateful to the reviewers for their very useful comments.

Funding The author declares that no funds, grants, or other support were received during the preparation of this manuscript.

Data Availability The link https://cs.haifa.ac.il~gordon/cgcd.c provides the C-program for the experiments. The text at the beginning of the program explains in detail how to run it.

Declarations

Conflict of interest The author declares that there is no conflict of interest.

References

1. Bertrand, Q., Massias, M.: Anderson acceleration of coordinate descent. In: Proceedings of International Conference on Artificial Intelligence and Statistics, vol. 130, no. 10 (2021)
2. Björck, Å., Elfving, T.: Accelerated projection methods for computing pseudoinverse solutions of systems of linear equations. BIT **19**, 145–163 (1979)
3. Feichtinger, H., Gröchenig, K.: Theory and practice of irregular sampling. In: Frazier, M. (ed.) Wavelets: Mathematics and Applications, pp. 305–363. CRC Press, Boca Raton (1994)

4. Gordon, D.: A derandomization approach to recovering bandlimited signals across a wide range of random sampling rates. Numer. Algorithms **77**(4), 1141–1157 (2018). https://doi.org/10.1007/s11075-017-0356-3
5. Gordon, D., Gordon, R.: CGMN revisited: robust and efficient solution of stiff linear systems derived from elliptic partial differential equations. ACM Trans. Math. Softw. **35**(3), 18:1-18:27 (2008)
6. Gordon, D., Gordon, R.: CARP-CG: a robust and efficient parallel solver for linear systems, applied to strongly convection-dominated PDEs. Parallel Comput. **36**(9), 495–515 (2010)
7. Kaczmarz, S.: Angenäherte Auflösung von Systemen linearer Gleichungen. Bull. l'Acad. Pol. Sci. Lett. **A35**, 355–357 (1937)
8. Lee, Y.T., Sidford, A.: Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems. In: Proceedings of 2013 IEEE 54th Annual Symposium on Foundations of Computer Science, FOCS 2013, pp. 147—156. IEEE Computer Society, USA (2013)
9. Nesterov, Y.E.: Efficiency of coordinate descent methods on huge-scale optimization problems. SIAM J. Optim. **22**(2), 341–362 (2012)
10. Nesterov, Y.E., Stich, S.U.: Efficiency of the accelerated coordinate descent method on structured optimization problems. SIAM J. Optim. **27**(1), 110–123 (2017)
11. Ruhe, A.: Numerical aspects of Gram–Schmidt orthogonalization of vectors. Linear Algebra Appl. **52**(53), 591–601 (1983)
12. Saad, Y.: Iterative Methods for Sparse Linear Systems, 2nd edn. SIAM, Philadelphia (2003)
13. Shi, H.-J.M., Tu, S., Xu, Y., Yin, W.: A primer on coordinate descent algorithms (2016)
14. Strohmer, T., Vershynin, R.: A randomized Kaczmarz algorithm with exponential convergence. J. Fourier Anal. Appl. **15**, 262–278 (2009)
15. Wang, Q., Li, W., Bao, W., Zhang, F.: Accelerated randomized coordinate descent for solving linear systems. Mathematics **10**(22), 4379 (2022)
16. Wright, S.J.: Coordinate descent algorithms. Math. Program. **151**(1), 3–34 (2015)