Les Cahiers du GERAD ISSN: 0711–2440

A semi-conjugate gradient method for solving unsymmetric positive definite linear systems

N. Huang, Y.-H. Dai, D. Orban, M. A. Saunders

G–2022–25

June 2022

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

A semi-conjugate gradient method for solving unsymmetric positive definite linear systems

Na Huang [a]

Yu-Hong Dai [b]

Dominique Orban [c]

Michael A. Saunders [d]

[a] *Department of Applied Mathematics, College of Science, China Agricultural University, Beijing, China*

[b] *LSEC, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, China*

[c] *GERAD & Department of Mathematics and Industrial Engineering, Polytechnique Montréal, Montréal (QC) Canada*

[d] *Systems Optimization Laboratory, Department of Management Science and Engineering, Stanford University, Stanford, CA, USA*

hna@cau.edu.cn
dyh@lsec.cc.ac.cn
dominique.orban@gerad.ca
saunders@stanford.edu

June 2022
Les Cahiers du GERAD
G–2022–25

Abstract : The conjugate gradient (CG) method is a classic Krylov subspace method for solving symmetric positive definite linear systems. We introduce an analogous semi-conjugate gradient (SCG) method for unsymmetric positive definite linear systems. Unlike CG, SCG requires the solution of a lower triangular linear system to produce each semi-conjugate direction. We prove that SCG is theoretically equivalent to the full orthogonalization method (FOM), which is based on the Arnoldi process and converges in a finite number of steps. Because SCG's triangular system increases in size each iteration, we study a sliding window implementation (SWI) to improve efficiency, and show that the directions produced are still locally semi-conjugate. A counter-example illustrates that SWI is different from the direct incomplete orthogonalization method (DIOM), which is FOM with a sliding window. Numerical experiments from the convection-diffusion equation and other applications show that SCG is robust and that the sliding window implementation SWI allows SCG to solve large systems efficiently.

Keywords: Linear system, sparse matrix, iterative method, semi-conjugate gradient method

---

Acknowledgements: Research of the first author is partially supported by National Natural Science Foundation of China (No. 12001531).

We would like to thank our colleague and friend, Dr Oleg Burdakov, for his devotion to research and for his everlasting sense of humor. In particular, we wish to express our gratitude to him for fundamental contributions that initiated this work, and for many constructive suggestions on our early Matlab implementation of SCG and SWI. At the end of 2017, he independently constructed SCG and SWI to solve quasi-definite linear systems. Subsequently, he extended them to general unsymmetric positive definite linear systems. The question of choosing an ideal m remains for future research, as it does for GMRES(m) and DQGMRES(m).

1 Introduction

We consider numerical methods for solving linear systems

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

where $A \in \mathbb{R}^{n\times n}$ is unsymmetric positive definite. Such a matrix A is positive definite if $x^TAx > 0$ holds for all nonzero $x \in \mathbb{R}^n$ [17]. This is true if and only if its symmetric part $(A+A^T)/2$ is symmetric positive definite.

Krylov subspace methods seek an approximate solution x_k from the affine subspace $x_0+\mathcal{K}_k(A, r_0)$, where x_0 is an arbitrary initial point, $r_0 = b - Ax_0$ is the initial residual, and $\mathcal{K}_k(A, r_0)$ is the Krylov subspace

$$\mathcal{K}_k(A, r_0) = \text{span}\{r_0, Ar_0, A^2r_0, \ldots, A^{k-1}r_0\},$$

which we denote by $\mathcal{K}_k$ when there is no ambiguity.

One of the best known Krylov subspace methods is the conjugate gradient (CG) method [18], which was derived in 1952 to solve sparse symmetric positive definite linear systems. When combined with a suitable preconditioning, CG has many successful applications in science and engineering. If A is unsymmetric or rectangular, CG could be applied to the normal equations $A^TAx = A^Tb$. It is numerically preferable to apply LSQR [22] or LSMR [15] to $\min \|Ax - b\|_2^2$, but the squared condition may lead to excessive iterations on compatible systems $Ax = b$.

A variety of other methods have been developed to deal with the square unsymmetric case, such as the generalized CG-type methods (e.g., Orthodir, Orthomin [2, 3, 31, 32]), the biconjugate gradient (BiCG) algorithm and its variations (e.g., CGS, BiCGSTAB, QMR, CSBCG [5, 16, 27, 28, 30]), the Manteuffel-Chebyshev iterations [19, 20, 29], and other generations based on orthogonal factorizations, Lanczos process (e.g., USYMLQ, USYMQR, LSQR, BiLQ [21, 22, 26]).

Arnoldi's method [1] was introduced in 1951 to deal with square unsymmetric matrices. This is an algorithm for constructing an orthonormal basis of the Krylov subspace $\mathcal{K}_k$. Subsequently, based on the Arnoldi process or its variations [23], several Krylov subspace methods for solving unsymmetric linear systems were established, such as the generalized minimum residual method (GMRES) [24], the direct quasi-GMRES method (DQGMRES) [25], the full orthogonalization method (FOM) [23], and the direct incomplete orthogonalization method (DIOM) [23].

Unlike finding an approximation from a Krylov subspace, Yuan et al. [33] sought an approximation following some semi-conjugate directions and presented two semi-conjugate direction (SCD) methods for general linear systems. They showed that SCD has no breakdown for real positive definite systems. Later, Dai and Yuan [9] further studied SCD methods and introduced a new implementation for generating the semi-conjugate directions using only the latest m conjugate directions, where m is a given positive integer. SCD methods also have been considered for solving nonlinear systems of equations and finding saddle points of functions, which called pseudo-orthogonal direction methods in [6, 7].

Here we also focus on SCD methods, taking the first direction to be r_0 as in CG. Hence, we call it the semi-conjugate gradient (SCG) method. We show that SCG is theoretically equivalent to FOM. SCG needs to solve a lower triangular linear system of increasing size at each step. To improve efficiency, we also study the sliding window implementation of SCG (SWI) and show that SWI still belongs to the set of Krylov subspace methods and will not break down. In contrast to the sliding window implementation of FOM (i.e., DIOM), the directions produced by SWI are locally semi-conjugate. This illustrates that SWI is different from DIOM. Several numerical experiments on linear systems from the convection-diffusion equation and other applications show that SWI often solves problems more efficiently than SCG and DIOM.

The paper is organized as follows. In Section 2, we introduce our semi-conjugate gradient method and provide some important properties. In Section 2.2, we prove that SCG is theoretically equivalent to FOM. The sliding window implementation of SCG and its convergence analysis are provided in Section 3. A counter-example in Section 3.1 illustrates that SWI and DIOM are different. Numerical experiments are reported in Section 4. Conclusions and future work are summarized in Section 5.

Notation

For a matrix $W \in \mathbb{R}^{n\times n}$, $\mathcal{H}_W = (W + W^T)/2$ and $\mathcal{S}_W = (W - W^T)/2$ denote the symmetric and skew-symmetric parts of W. $\lambda(W)$ and $\rho(W)$ denote an arbitrary eigenvalue and the spectral radius of W. $\lambda_{\min}(W)$ and $\lambda_{\max}(W)$ denote the minimum and maximum eigenvalues of a symmetric matrix W. A vector e_k is the kth column of an identity matrix. The solution of $Ax = b$ (1) is denoted by $x_\star$. The kth approximation to $x_\star$ is x_k, and the corresponding error is $d_k = x_k - x_\star$. The 2-norm $\|v\|$ is used for vectors v.

2 The semi-conjugate gradient method

In this section, we introduce the semi-conjugate gradient method SCG to solve unsymmetric positive definite linear systems (1). The method is summarized in Algorithm 1. Without loss of generality, we assume that $x_0 = 0$ and then $r_0 = b$.

Algorithm 1 SCG: The semi-conjugate gradient method

1: Given $x_0 = 0$, set $k = 0$, $r_0 = b$, $p_0 = r_0$ and $q_0 = Ap_0$.
2: **while** a stopping condition is not satisfied **do**
3: Compute the step size $\alpha_k = p_k^T r_k / p_k^T q_k$.
Update $x_{k+1} = x_k + \alpha_k p_k$ and $r_{k+1} = r_k - \alpha_k q_k$.
4: Form $v_{k+1} = Ar_{k+1}$, $L_{k+1} = P_{k+1}^T Q_{k+1}$ and solve $L_{k+1}\lambda_{k+1} = P_{k+1}^T v_{k+1}$,
where $P_{k+1} = \begin{bmatrix} p_0 & p_1 & \cdots & p_k \end{bmatrix}$, $Q_{k+1} = \begin{bmatrix} q_0 & q_1 & \cdots & q_k \end{bmatrix}$.
5: Update

$$p_{k+1} = r_{k+1} - P_{k+1}\lambda_{k+1}, \qquad q_{k+1} = v_{k+1} - Q_{k+1}\lambda_{k+1}. \tag{2}$$

6: Increment k by 1.
7: **end while**

As shown in Lemma 2.2 below, L_{k+1} is lower triangular. The system $L_{k+1}\lambda_{k+1} = P_{k+1}^T v_{k+1}$ has the form

$$\begin{pmatrix} p_0^T q_0 & & & & \\ p_1^T q_0 & p_1^T q_1 & & & \\ p_2^T q_0 & p_2^T q_1 & p_2^T q_2 & & \\ \vdots & \vdots & \vdots & \ddots & \\ p_k^T q_0 & p_k^T q_1 & p_k^T q_2 & \cdots & p_k^T q_k \end{pmatrix} \begin{pmatrix} \lambda_{k+1}^{(1)} \\ \lambda_{k+1}^{(2)} \\ \lambda_{k+1}^{(3)} \\ \vdots \\ \lambda_{k+1}^{(k+1)} \end{pmatrix} = \begin{pmatrix} p_0^T v_{k+1} \\ p_1^T v_{k+1} \\ p_2^T v_{k+1} \\ \vdots \\ p_k^T v_{k+1} \end{pmatrix},$$

where $\lambda_{k+1}^{(i)}$ denotes the ith component of λ_{k+1}. Hence

$$\lambda_{k+1}^{(1)} = \frac{p_0^T v_{k+1}}{p_0^T q_0},$$
$$\lambda_{k+1}^{(i)} = \frac{p_{i-1}^T (v_{k+1} - \lambda_{k+1}^{(1)} q_0 - \lambda_{k+1}^{(2)} q_1 - \cdots - \lambda_{k+1}^{(i-1)} q_{i-2})}{p_{i-1}^T q_{i-1}} \quad (i > 1).$$

Algorithm 2 combines this with (2) to compute directions p_{k+1} and q_{k+1}.

As stated, Algorithm 1 and Algorithm 2 together form a special case of [9, Algorithm 2.3] in which $p_0 = r_0$. In other words, SCG is a special case of SCD [33, Algorithm 4.1], though the latter does not explicitly use q_k. We now derive some further important properties of SCG and show that it generates the same iterates as FOM. Thus, SCG theoretically follows the iterations of CG if A is SPD.

Algorithm 2 Computation of p_{k+1} and q_{k+1}

Assume that $p_0, \ldots, p_k$, $q_0, \ldots, q_k$ and $r_0, \ldots, r_{k+1}$ have been computed.
Set $p \leftarrow r_{k+1}$ and $q \leftarrow v_{k+1}$.
for $i = 1, 2, \ldots, k+1$ **do**
  Compute $\lambda_{k+1}^{(i)} = p_{i-1}^T q / p_{i-1}^T q_{i-1}$.
  Set $p \leftarrow p - \lambda_{k+1}^{(i)} p_{i-1}$ and $q \leftarrow q - \lambda_{k+1}^{(i)} q_{i-1}$.
end for
Set $p_{k+1} = p$ and $q_{k+1} = q$.

2.1 Convergence analysis of SCG

We provide properties of SCG and prove that it converges in a finite number of steps.

Lemma 2.1. The sequence $\{p_k, q_k\}$ produced by SCG satisfies $q_k = Ap_k$.

Proof. For $k = 0$, we have $q_0 = Ap_0$ and $Q_1 = AP_1$. For $k \geq 1$, by recursion, (2), we have

$$q_k = v_k - Q_k \lambda_k = Ar_k - AP_k \lambda_k = Ap_k. \qquad \square$$

Lemma 2.2. The matrices L_k $(k \geq 1)$ are nonsingular and lower triangular with positive diagonal elements.

Proof. The proof is by induction on k. From Lemma 2.1 and the fact that A is positive definite, $L_1 = P_1^T Q_1 = p_0^T A p_0 > 0$, so the result holds for $k = 1$. Assume L_k possesses the desired property. By definition of L_k,

$$P_k^T q_k = P_k^T (v_k - Q_k L_k^{-1} P_k^T v_k) = P_k^T v_k - P_k^T Q_k L_k^{-1} P_k^T v_k = 0,$$

and from Lemma 2.1 we see that

$$L_{k+1} = P_{k+1}^T Q_{k+1} = \begin{bmatrix} P_k & p_k \end{bmatrix}^T \begin{bmatrix} Q_k & q_k \end{bmatrix} = \begin{pmatrix} P_k^T Q_k & P_k^T q_k \\ p_k^T Q_k & p_k^T q_k \end{pmatrix}$$
$$= \begin{pmatrix} L_k & 0 \\ p_k^T Q_k & p_k^T A p_k \end{pmatrix}$$

is also nonsingular and lower triangular with positive diagonal elements. $\square$

Remark 2.1. As A is positive definite, for $p_k \neq 0$ we get $p_k^T q_k = p_k^T A p_k > 0$. The step size α_k in SCG is therefore well defined. In addition, from Lemma 2.2, we see that L_{k+1} is nonsingular if $p_j \neq 0$, $j = 0, \ldots, k$. Therefore, SCG will not break down as long as $p_k \neq 0$.

From Lemmas 2.1 and 2.2 we immediately obtain the following result.

Corollary 2.1. For all $j > i \geq 0$, it holds that $p_i^T q_j = p_i^T A p_j = 0$.

Lemma 2.3. After k iterations of SCG we have

$$r_k^T q_k = p_k^T q_k, \tag{3}$$
$$P_k^T r_k = 0, \tag{4}$$
$$r_i^T r_k = 0, \quad i = 0, 1, \ldots, k-1. \tag{5}$$

Proof. It follows from (2) that

$$r_k = P_k L_k^{-1} P_k^T A r_k + p_k. \tag{6}$$

This along with Corollary 2.1 leads to

$$r_k^T q_k = (P_k L_k^{-1} P_k^T A r_k + p_k)^T q_k = r_k^T A^T P_k L_k^{-T} P_k^T q_k + p_k^T q_k = p_k^T q_k.$$

Therefore, (3) holds.

The proof of (4) and (5) is by induction on k. For $k = 1$ with $p_0 = r_0$,

$$r_0^T r_1 = r_0^T (r_0 - \alpha_0 q_0) = r_0^T r_0 - \frac{p_0^T r_0}{p_0^T q_0} r_0^T q_0 = r_0^T r_0 - \frac{r_0^T r_0}{r_0^T q_0} r_0^T q_0 = 0,$$

and $P_1^T r_1 = p_0^T r_1 = r_0^T r_1 = 0$. Hence, (4) and (5) hold for $k = 1$.

Suppose (4) and (5) hold for some $k \geq 1$. Then if $i < k$,

$$r_i^T r_{k+1} = r_i^T (r_k - \alpha_k q_k) = r_i^T r_k - \alpha_k r_i^T q_k = -\alpha_k r_i^T q_k. \tag{7}$$

With (6) and Corollary 2.1 this yields

$$r_i^T q_k = (P_i L_i^{-1} P_i^T A r_i + p_i)^T q_k = r_i^T A^T P_i L_i^{-T} P_i^T q_k + p_i^T q_k = 0.$$

Substituting into (7) gives $r_i^T r_{k+1} = 0$.

If $i = k$, we check directly that

$$r_k^T r_{k+1} = r_k^T (r_k - \alpha_k q_k) = r_k^T r_k - \frac{p_k^T r_k}{p_k^T q_k} r_k^T q_k. \tag{8}$$

Using (2) and the inductive assumption, we have

$$p_k^T r_k = (r_k - P_k L_k^{-1} P_k^T A r_k)^T r_k = r_k^T r_k - r_k^T A^T P_k L_k^{-T} P_k^T r_k = r_k^T r_k. \tag{9}$$

Substituting (3) and (9) into (8) gives

$$r_k^T r_{k+1} = r_k^T r_k - \frac{r_k^T r_k}{p_k^T q_k} p_k^T q_k = 0,$$

so that (5) also holds for $k + 1$.

By Corollary 2.1 and the inductive assumption, we know that

$$P_k^T r_{k+1} = P_k^T (r_k - \alpha_k q_k) = P_k^T r_k - \alpha_k P_k^T q_k = 0,$$
$$p_k^T r_{k+1} = p_k^T (r_k - \alpha_k q_k) = p_k^T r_k - \frac{p_k^T r_k}{p_k^T q_k} p_k^T q_k = 0.$$

This implies that $P_{k+1}^T r_{k+1} = \begin{pmatrix} P_k^T r_{k+1} \\ p_k^T r_{k+1} \end{pmatrix} = 0.$ □

It follows from (2) and Lemma 2.3 that

$$p_k^T r_k = (r_k - P_k \lambda_k)^T r_k = r_k^T r_k - \lambda_k^T P_k^T r_k = r_k^T r_k.$$

This implies that α_k in SCG can also be defined by $\alpha_k = r_k^T r_k / p_k^T q_k$, which saves the computation of $p_k^T r_k$ and allows us to reuse the quantity $r_k^T r_k$ that typically appears in a stopping condition.

Lemma 2.4. After $k - 1$ iterations of SCG, if $p_k = 0$ then $r_k = 0$.

Proof. Assume $p_j \neq 0,\ j = 0, \ldots, k-1$. From the definition of L_k and Lemmas 2.1 and 2.2 we know that P_k has full column rank. Let

$$P_k = U_k \begin{pmatrix} \Sigma_k \\ 0 \end{pmatrix} V_k^T \tag{10}$$

be the singular value decomposition (SVD), where $n \times n$ U_k and $k \times k$ V_k are unitary matrices, and $\Sigma_k = \text{diag}\{\sigma_1, \sigma_2, \ldots, \sigma_k\}$ with all $\sigma_j > 0$ contains the singular values of P_k. It follows from Lemma 2.3 that

$$0 = P_k^T r_k = V_k \begin{pmatrix} \Sigma_k & 0 \end{pmatrix} U_k^T r_k = V_k \begin{pmatrix} \Sigma_k & 0 \end{pmatrix} \begin{pmatrix} \tilde{r}_k^{(1)} \\ \tilde{r}_k^{(2)} \end{pmatrix} = V_k \Sigma_k \tilde{r}_k^{(1)},$$

where $U_k^T r_k = ((\tilde{r}_k^{(1)})^T, (\tilde{r}_k^{(2)})^T)^T$. This implies that $\tilde{r}_k^{(1)} = 0$.

If $p_k = 0$, by (2) and Lemma 2.1 we get $(I - P_k(P_k^T A P_k)^{-1} P_k^T A) r_k = 0$, i.e.,

$$(U_k^T A U_k - U_k^T A P_k (P_k^T A P_k)^{-1} P_k^T A U_k) U_k^T r_k = 0$$

as U_k is unitary and A is nonsingular. Substituting (10) gives

$$\left[U_k^T A U_k - U_k^T A U_k \begin{pmatrix} \Sigma_k \\ 0 \end{pmatrix} \left(\begin{pmatrix} \Sigma_k & 0 \end{pmatrix} U_k^T A U_k \begin{pmatrix} \Sigma_k \\ 0 \end{pmatrix} \right)^{-1} \begin{pmatrix} \Sigma_k & 0 \end{pmatrix} U_k^T A U_k \right] U_k^T r_k = 0. \tag{11}$$

Let $U_k^T A U_k = \begin{pmatrix} \tilde{A}_k^{(1)} & \tilde{A}_k^{(2)} \\ \tilde{A}_k^{(3)} & \tilde{A}_k^{(4)} \end{pmatrix} \in \mathbb{R}^{n \times n}$, where $\tilde{A}_k^{(1)} \in \mathbb{R}^{k \times k}$. Then (11) reads

$$0 = \begin{pmatrix} 0 & 0 \\ 0 & \tilde{A}_k^{(4)} - \tilde{A}_k^{(3)} (\tilde{A}_k^{(1)})^{-1} \tilde{A}_k^{(2)} \end{pmatrix} \begin{pmatrix} \tilde{r}_k^{(1)} \\ \tilde{r}_k^{(2)} \end{pmatrix} = \begin{pmatrix} 0 \\ \left[\tilde{A}_k^{(4)} - \tilde{A}_k^{(3)} (\tilde{A}_k^{(1)})^{-1} \tilde{A}_k^{(2)} \right] \tilde{r}_k^{(2)} \end{pmatrix}.$$

Since A is positive definite, we have $\tilde{r}_k^{(2)} = 0$. This along with $\tilde{r}_k^{(1)} = 0$ leads to $U_k^T r_k = 0$, which gives the result by the full column rank of U_k. □

Remark 2.2. Combining Remark 2.1 with Lemma 2.4, we see that SCG will not break down unless $r_k = 0$.

We are now ready to prove that SCG converges in a finite number of steps.

Theorem 2.1. SCG converges to the unique solution of the linear system (1) within $n + 1$ steps if roundoff errors are ignored.

Proof. If $r_k \neq 0$ for all $k = 0, 1, \ldots, n-1$, by Lemma 2.3 r_0, $r_1, \ldots, r_{n-1}$ are orthogonal, and therefore linearly independent. Then there exist $a_0, a_1, \ldots, a_{n-1}$ such that $r_n = \sum_{i=0}^{n-1} a_i r_i$. Lemma 2.3 then yields $r_n^T r_n = \sum_{i=0}^{n-1} a_i r_n^T r_i = 0$. □

2.2 SCG is equivalent to FOM

In this section, we show that SCG and FOM are theoretically equivalent.

FOM is a Krylov subspace method introduced by Saad [23] in which the residual associated with the kth solution approximation $\hat{x}_k$ satisfies the Galerkin condition

$$\hat{r}_k = b - A\hat{x}_k \perp \mathcal{K}_k. \tag{12}$$

Given the initial guess $\hat{x}_0 = 0$ and $\beta = \|\hat{r}_0\| = \|b\|$, Arnoldi's method sets $\hat{v}_1 = \hat{r}_0/\beta$ and, for $j = 1, 2, \ldots, k-1$, computes

$$\begin{cases} h_{ij} = \hat{v}_i^T A \hat{v}_j, \ i = 1, 2, \ldots, j, \\ w_j = A\hat{v}_j - \sum_{i=1}^{j} h_{ij} \hat{v}_i, \\ h_{j+1,j} = \|w_j\|, \\ \hat{v}_{j+1} = w_j / h_{j+1,j}. \end{cases} \tag{13}$$

This process constructs $\hat{V}_k = \begin{bmatrix} \hat{v}_1 & \hat{v}_2 & \dots & \hat{v}_k \end{bmatrix}$ whose columns form an orthonormal basis of $\mathcal{K}_k$ such that

$$A\hat{V}_k = \hat{V}_k H_k + h_{k+1,k}\hat{v}_{k+1}e_k^T,$$

where H_k is $k\times k$ upper Hessenberg and $h_{k+1,k}$ will appear in H_{k+1}. FOM seeks a solution of the form $\hat{x}_k = \hat{V}_k y_k$. Thus,

$$\hat{r}_k = \beta\hat{v}_1 - A\hat{V}_k y_k = \hat{V}_{k+1}\begin{pmatrix} \beta e_1 - H_k y_k \\ -h_{k+1,k}e_k^T y_k \end{pmatrix}.$$

By (12), the kth approximate solution $\hat{x}_k$ in FOM is given by

$$y_k = H_k^{-1}(\beta e_1).$$

If A is positive definite, so is each H_k in exact arithmetic because $H_k = \hat{V}_k^T A\hat{V}_k$. Therefore, it possesses an LU factorization without pivoting,[1] say $H_k = \hat{L}_k\hat{U}_k$ [17] with $\hat{L}_k$ unit lower triangular and $\hat{U}_k$ upper triangular with positive diagonal elements, which allows us to state FOM in the form of Algorithm 3.

Algorithm 3 FOM [23, Algorithms 6.4 and 6.8]

Given $\hat{x}_0 = 0$, set $\hat{r}_0 = b$, $\beta = \|\hat{r}_0\|$, and $\hat{v}_1 = \hat{r}_0/\beta$.
for $k = 1, 2, \dots$ **do**
  Compute h_{ik}, $i = 1, 2, \dots, k$ and $\hat{v}_{k+1}$ by the Arnoldi process.
  Update the LU factorization of H_k, i.e., obtain the last column u_k of $\hat{U}_k$.
  Compute $\zeta_k = \{\text{if } k = 1 \text{ then } \beta, \text{else } -l_{k,k-1}\zeta_{k-1}\}$.
  Compute $\hat{p}_k = (\hat{v}_k - \sum_{i=1}^{k-1} u_{ik}\hat{p}_i)/u_{kk}$.
  Compute $\hat{x}_k = \hat{x}_{k-1} + \zeta_k\hat{p}_k$.
  Compute $\hat{r}_k = \hat{r}_{k-1} - \zeta_k A\hat{p}_k$.
end for

We now state properties of FOM used to analyze its connection with SCG.

Lemma 2.5. [23, Proposition 6.7] In FOM,

$$\hat{r}_k = -h_{k+1,k}e_k^T y_k \hat{v}_{k+1} = \hat{v}_{k+1}/t_{k+1},$$

where $y_k = H_k^{-1}(\beta e_1)$ and $t_{k+1} = 1/(-h_{k+1,k}e_k^T y_k)$.

Lemma 2.6. [23, Properties on page 157] In FOM,

- the directions $\hat{p}_k$ are semi-conjugate, i.e., $\hat{p}_i^T A\hat{p}_j = 0$ for $i < j$;
- the residual vectors $\hat{r}_k$ are orthogonal, i.e., $\hat{r}_i^T\hat{r}_j = 0$ for $i \neq j$.

The connection between SCG and FOM can now be summarized as follows.

Theorem 2.2. Assume that $\hat{r}_k$ and $\hat{p}_k$ are produced by FOM, and r_k and p_k are produced by SCG. Then for all $k \geq 1$, if $\hat{r}_{k-1} \neq 0$ and $r_{k-1} \neq 0$, there exists $a_k \neq 0$ such that

$$\hat{p}_k = a_{k-1}p_{k-1}, \tag{14}$$
$$\hat{r}_k = r_k. \tag{15}$$

Proof. We use induction on k. For $k = 1$, as $\hat{r}_0 = r_0 = p_0$, it is easy to see that

$$\hat{p}_1 = u_{11}^{-1}\hat{v}_1 = \frac{1}{\beta u_{11}}r_0 = \frac{1}{\beta h_{11}}p_0.$$

Note that $h_{11} = \hat{v}_1^T A\hat{v}_1 = (r_0^T Ar_0)/\beta^2$ leads to

$$\hat{p}_1 = \frac{\beta}{r_0^T Ar_0}p_0 = \frac{\|r_0\|}{r_0^T Ar_0}p_0.$$

[1] In practice, pivoting remains advisable in general for stability.

In addition, by $\hat{r}_0 = r_0 = p_0$ and $\zeta_1 = \beta$, we have

$$\hat{r}_1 = \hat{r}_0 - \zeta_1 \hat{p}_1 = r_0 - \frac{\beta \|r_0\|}{r_0^T A r_0} p_0 = r_0 - \frac{r_0^T r_0}{r_0^T A r_0} p_0 = r_0 - \frac{p_0^T r_0}{p_0^T A p_0} p_0 = r_0 - \alpha_0 p_0 = r_1.$$

We then have (14)–(15) satisfied for $k = 1$ with $a_0 = \|r_0\|/(r_0^T A r_0) \neq 0$.

Suppose there exist constants $a_i \neq 0$ $(0 \leq i \leq K-2)$ such that (14)–(15) are satisfied for all $1 \leq k < K$. We show that (14)–(15) also hold for $k = K$. Let

$$\hat{P}_{K-1} = \begin{bmatrix} \hat{p}_1 & \cdots & \hat{p}_{K-1} \end{bmatrix}, \quad \tilde{u}_K = (u_{1,K}, \ldots, u_{K-1,K})^T, \quad D_{K-1} = \operatorname{diag}\{a_0, \ldots, a_{K-2}\}.$$

From the induction hypothesis, D_{K-1} is nonsingular and $\hat{P}_{K-1} = P_{K-1} D_{K-1}$. With Lemma 2.5, this leads to

$$\begin{aligned} \hat{p}_K &= u_{K,K}^{-1} \big(\hat{v}_K - \sum_{i=1}^{K-1} u_{i,K} \hat{p}_i \big) \\ &= u_{K,K}^{-1} \big(\hat{v}_K - \hat{P}_{K-1} \tilde{u}_K \big) \\ &= u_{K,K}^{-1} \big(t_K \hat{r}_{K-1} - P_{K-1} D_{K-1} \tilde{u}_K \big) \\ &= u_{K,K}^{-1} \big(t_K r_{K-1} - P_{K-1} D_{K-1} \tilde{u}_K \big). \end{aligned} \tag{16}$$

From Lemma 2.6, $\hat{p}_i^T A \hat{p}_K = 0$ holds for all $i < K$, including $\hat{P}_{K-1}^T A \hat{p}_K = 0$. Since $\hat{P}_{K-1} = P_{K-1} D_{K-1}$ and the matrix D_{K-1} is nonsingular, we get

$$P_{K-1}^T A \hat{p}_K = 0.$$

Multiplying both sides of (16) by $P_{K-1}^T A$ and using SCG and Lemma 2.1, we obtain

$$\begin{aligned} 0 = t_K P_{K-1}^T A r_{K-1} - P_{K-1}^T A P_{K-1} D_{K-1} \tilde{u}_K &= t_K P_{K-1}^T v_{K-1} - P_{K-1}^T Q_{K-1} D_{K-1} \tilde{u}_K \\ &= t_K P_{K-1}^T v_{K-1} - L_{K-1} D_{K-1} \tilde{u}_K, \end{aligned}$$

where we used the identity $A r_{K-1} = v_{K-1}$ from Algorithm 1. This shows that

$$D_{K-1} \tilde{u}_K = t_K L_{K-1}^{-1} P_{K-1}^T v_{K-1} = t_K \lambda_{K-1}.$$

We substitute the above into (16) and use (2) to obtain

$$\hat{p}_K = u_{K,K}^{-1} \big(t_K r_{K-1} - t_K P_{K-1} \lambda_{K-1} \big) = u_{K,K}^{-1} t_K p_{K-1}.$$

Therefore, (14) holds for $k = K$ with $a_{K-1} = u_{K,K}^{-1} t_K \neq 0$.

It follows from FOM, the induction hypothesis and Lemma 2.1 that

$$\hat{r}_K = \hat{r}_{K-1} - \zeta_K A \hat{p}_K = r_{K-1} - \zeta_K a_{K-1} A p_{K-1} = r_{K-1} - \zeta_K a_{K-1} q_{K-1}. \tag{17}$$

This along with Lemma 2.6, the induction hypothesis, and (3) gives

$$\begin{aligned} 0 = \hat{r}_{K-1}^T \hat{r}_K = r_{K-1}^T \hat{r}_K &= r_{K-1}^T r_{K-1} - \zeta_K a_{K-1} r_{K-1}^T q_{K-1} \\ &= r_{K-1}^T r_{K-1} - \zeta_K a_{K-1} p_{K-1}^T q_{K-1}. \end{aligned}$$

By Corollary 2.1 and the positive definiteness of A, $p_{K-1}^T q_{K-1} > 0$. Thus,

$$\zeta_K a_{K-1} = \frac{r_{K-1}^T r_{K-1}}{p_{K-1}^T q_{K-1}}.$$

By (2) and Lemma 2.3, we obtain

$$r_{K-1}^T r_{K-1} = (p_{K-1} + P_{K-1}\lambda_{K-1})^T r_{K-1} = p_{K-1}^T r_{K-1} + \lambda_{K-1}^T P_{K-1}^T r_{K-1} = p_{K-1}^T r_{K-1}.$$

We then have $\zeta_K a_{K-1} = \frac{p_{K-1}^T r_{K-1}}{p_{K-1}^T q_{K-1}} = \alpha_{K-1}$. Combining with (17) gives

$$\hat{r}_K = r_{K-1} - \alpha_{K-1} q_{K-1} = r_K.$$

Hence, (15) also holds for $k = K$. □

In the following, we show that neither $\{\|x_k - x_\star\|\}$ nor $\{\|x_k\|\}$ produced by FOM or SCG is monotonic. Consider

$$A = \begin{pmatrix} 1 & 0 & -2 \\ 0 & 1 & 0 \\ 2 & 0 & 2 \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}.$$

Note that A is positive definite. With $\hat{x}_0 = 0$, we have $\hat{r}_0 = b$ and $\beta = \|\hat{r}_0\| = 1$. It follows from (13) that

$$\hat{v}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad \hat{v}_2 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \quad H_1 = 1, \quad H_2 = \begin{pmatrix} 1 & -2 \\ 2 & 2 \end{pmatrix}.$$

Then we have

$$\hat{x}_1 = \hat{V}_1 y_1 = \hat{V}_1 H_1^{-1}(\beta e_1) = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}^T,$$
$$\hat{x}_2 = \hat{V}_2 y_2 = \hat{V}_2 H_2^{-1}(\beta e_1) = \begin{pmatrix} \frac{1}{3} & 0 & -\frac{1}{3} \end{pmatrix}^T = x_\star.$$

This implies that $\|\hat{x}_0 - x_\star\| = \sqrt{2}/3$, $\|\hat{x}_1 - x_\star\| = \sqrt{5}/3$, $\|\hat{x}_2 - x_\star\| = 0$ and $\|\hat{x}_0\| = 0$, $\|\hat{x}_1\| = 1$, $\|\hat{x}_2\| = \sqrt{2}/3$. Therefore, the sequences $\{\|\hat{x}_k - x_\star\|\}$ and $\{\|\hat{x}_k\|\}$ produced by FOM (and SCG) are not monotonic.

3 Sliding window implementation of SCG

In this section, we study the sliding window implementation of SCG, which is described in Algorithm 4.

Algorithm 4 SWI: Sliding window implementation of SCG

1: Given $x_0 = 0$ and a nonnegative integer m, set $r_0 = b$, $p_0 = r_0$ and $q_0 = Ap_0$.
2: **while** a stopping condition is not satisfied **do**
3: Compute the step size $\alpha_k = r_k^T p_k / p_k^T q_k$.
Update $x_{k+1} = x_k + \alpha_k p_k$ and $r_{k+1} = r_k - \alpha_k q_k$.
4: Form $v_{k+1} = Ar_{k+1}$, $L_{k+1} = P_{k+1}^T Q_{k+1}$ and solve $L_{k+1}\lambda_{k+1} = P_{k+1}^T v_{k+1}$,
where $P_{k+1} = \begin{bmatrix} p_{k-m_k} & \cdots & p_k \end{bmatrix}$, $Q_{k+1} = \begin{bmatrix} q_{k-m_k} & \cdots & q_k \end{bmatrix}$, $m_k = \min\{k, m\}$.
5: Update

$$p_{k+1} = r_{k+1} - P_{k+1}\lambda_{k+1}, \qquad q_{k+1} = v_{k+1} - Q_{k+1}\lambda_{k+1}. \tag{18}$$

6: Increment k by 1.
7: **end while**

Dai and Yuan [9, Algorithm 5.1] proposed the limited-memory left conjugate direction method, which is theoretically equivalent to SWI, but they did not provide an analysis of the method. In the following, we derive some properties of SWI and show that it is convergent under reasonable conditions.

As in the proof of Lemma 2.1, we obtain a relation between p_k and q_k in SWI.

Lemma 3.1. The sequence $\{p_k, q_k\}$ produced by SWI satisfies $q_k = Ap_k$.

Lemma 3.2. The SWI matrices L_k $(k \geq 1)$ are nonsingular and lower triangular.

Proof. If $1 \leq k \leq m+1$, it follows from Lemma 2.2 that L_k is nonsingular and lower triangular. Assume that L_k is nonsingular and lower triangular for some $k \geq m+1$. Let us now show that the same is true of L_{k+1}. Let $\tilde{P}_k = \begin{bmatrix} p_{k-m} & \dots & p_{k-1} \end{bmatrix}$, $\tilde{Q}_k = \begin{bmatrix} q_{k-m} & \dots & q_{k-1} \end{bmatrix}$ and $k_m = k-m-1$. It is easy to see that

$$\begin{aligned} P_k &= \begin{pmatrix} p_{k_m} & \tilde{P}_k \end{pmatrix}, \quad P_{k+1} = \begin{pmatrix} \tilde{P}_k & p_k \end{pmatrix}, \\ Q_k &= \begin{pmatrix} q_{k_m} & \tilde{Q}_k \end{pmatrix}, \quad Q_{k+1} = \begin{pmatrix} \tilde{Q}_k & q_k \end{pmatrix}. \end{aligned} \tag{19}$$

As L_k is nonsingular and lower triangular, we have

$$L_k = P_k^T Q_k = \begin{pmatrix} p_{k_m}^T \\ \tilde{P}_k^T \end{pmatrix} \begin{pmatrix} q_{k_m} & \tilde{Q}_k \end{pmatrix} = \begin{pmatrix} p_{k_m}^T q_{k_m} & p_{k_m}^T \tilde{Q}_k \\ \tilde{P}_k^T q_{k_m} & \tilde{P}_k^T \tilde{Q}_k \end{pmatrix} = \begin{pmatrix} p_{k_m}^T q_{k_m} & 0 \\ \tilde{P}_k^T q_{k_m} & \tilde{P}_k^T \tilde{Q}_k \end{pmatrix}, \tag{20}$$

which implies that $\tilde{P}_k^T \tilde{Q}_k$ is also nonsingular and lower triangular. Together, (19) and (20) yield

$$\tilde{P}_k^T Q_k L_k^{-1} P_k^T = \begin{pmatrix} \tilde{P}_k^T q_{k_m} & \tilde{P}_k^T \tilde{Q}_k \end{pmatrix} \begin{pmatrix} p_{k_m}^T q_{k_m} & 0 \\ \tilde{P}_k^T q_{k_m} & \tilde{P}_k^T \tilde{Q}_k \end{pmatrix}^{-1} \begin{pmatrix} p_{k_m}^T \\ \tilde{P}_k^T \end{pmatrix} = \tilde{P}_k^T. \tag{21}$$

Combining (18) with (21) gives $\tilde{P}_k^T q_k = \tilde{P}_k^T v_k - \tilde{P}_k^T Q_k L_k^{-1} P_k^T v_k = 0$. Thus,

$$L_{k+1} = \begin{pmatrix} \tilde{P}_k^T \\ p_k^T \end{pmatrix} \begin{pmatrix} \tilde{Q}_k & q_k \end{pmatrix} = \begin{pmatrix} \tilde{P}_k^T \tilde{Q}_k & \tilde{P}_k^T q_k \\ p_k^T \tilde{Q}_k & p_k^T q_k \end{pmatrix} = \begin{pmatrix} \tilde{P}_k^T \tilde{Q}_k & 0 \\ p_k^T \tilde{Q}_k & p_k^T q_k \end{pmatrix}$$

is also lower triangular. Using Lemma 3.1 and the fact that A is positive definite, we have $p_k^T q_k = p_k^T A p_k > 0$. Therefore, L_{k+1} is nonsingular. □

With all L_k nonsingular, SWI is well defined. Lemma 3.2 also implies the following.

Corollary 3.1. For all $i \in [\max\{0,\, k-m\}, k-1]$, it holds that $p_i^T q_k = 0$.

Lemma 3.3. After k iterations in SWI we have $P_k^T r_k = 0$.

Proof. If $k \leq m+1$, it follows from Lemma 2.3 that $P_k^T r_k = 0$. Now we prove that $P_k^T r_k = 0$ also holds for $k > m+1$. The proof is by induction on k. Assume that $P_k^T r_k = 0$ for some $k \geq m+1$. For the case of $k+1$, it follows from (19) that

$$P_k^T r_k = \begin{pmatrix} p_{k_m}^T \\ \tilde{P}_k^T \end{pmatrix} r_k = \begin{pmatrix} p_{k_m}^T r_k \\ \tilde{P}_k^T r_k \end{pmatrix} = 0.$$

This together with Corollary 3.1 yields

$$\tilde{P}_k^T r_{k+1} = \tilde{P}_k^T (r_k - \alpha_k q_k) = \tilde{P}_k^T r_k - \alpha_k \tilde{P}_k^T q_k = 0.$$

From the expression for α_k, we have

$$p_k^T r_{k+1} = p_k^T (r_k - \alpha_k q_k) = p_k^T r_k - \alpha_k p_k^T q_k = p_k^T r_k - \frac{r_k^T p_k}{p_k^T q_k} p_k^T q_k = 0.$$

This shows that

$$P_{k+1}^T r_{k+1} = \begin{pmatrix} \tilde{P}_k^T \\ p_k^T \end{pmatrix} r_{k+1} = \begin{pmatrix} \tilde{P}_k^T r_{k+1} \\ p_k^T r_{k+1} \end{pmatrix} = 0.$$

The proof follows by induction. □

Remark 3.1. It follows from Lemma 3.3 that

$$r_k^T p_k = r_k^T (r_k - P_k \lambda_k) = r_k^T r_k - r_k^T P_k \lambda_k = r_k^T r_k. \tag{22}$$

Hence, the step size α_k in SWI can also be updated by $\alpha_k = r_k^T r_k / p_k^T q_k$.

Remark 3.2. In the same way as for Lemma 2.4, we can prove that if $p_k = 0$, then $r_k = 0$. Hence, SWI will not break down unless $r_k = 0$.

In the following, we show that SWI is a Krylov subspace method.

Lemma 3.4. The sequence $\{x_k, r_k, p_k\}$ produced by SWI satisfies $x_k \in \mathcal{K}_k$ and $r_k, p_k \in \mathcal{K}_{k+1}$.

Proof. For $k = 0$, the results hold naturally. Assume that the results hold for some $k \geq 0$. Then for $k+1$, it follows from SWI, Lemma 3.1 and the induction hypothesis that

$$\begin{aligned} x_{k+1} &= x_k + \alpha_k p_k \in \mathcal{K}_{k+1}, \\ r_{k+1} &= r_k - \alpha_k q_k = r_k - \alpha_k A p_k \in \mathcal{K}_{k+2}, \\ p_{k+1} &= r_{k+1} - P_{k+1} \lambda_{k+1} \in \mathcal{K}_{k+2}. \end{aligned}$$

The proof follows by induction. □

We are now ready to establish the convergence theorem for SWI. For any $k \geq m$, let the SVD of $n \times (m+1)$ matrix P_k be

$$P_k = U_k \begin{pmatrix} \Sigma_k \\ 0 \end{pmatrix} V_k^T, \tag{23}$$

where $\Sigma_k = \mathrm{diag}\{\sigma_1, \sigma_2, \ldots, \sigma_{m+1}\}$ and $\sigma_j > 0$. Also let

$$U_k^T A U_k = \begin{pmatrix} \tilde{A}_k^{(1)} & \tilde{A}_k^{(2)} \\ \tilde{A}_k^{(3)} & \tilde{A}_k^{(4)} \end{pmatrix} \in \mathbb{R}^{n \times n} \quad \text{and} \quad U_k^T r_k = \begin{pmatrix} \tilde{r}_k^{(1)} \\ \tilde{r}_k^{(2)} \end{pmatrix} \in \mathbb{R}^n, \tag{24}$$

where $\tilde{A}_k^{(1)} \in \mathbb{R}^{(m+1)\times(m+1)}$ and $\tilde{r}_k^{(1)} \in \mathbb{R}^{m+1}$. As A is positive definite, so is $U_k^T A U_k$. From [4, Theorem 3.9], it follows that $\tilde{A}_k^{(1)}$ is also positive definite. We can then define the Schur complement of $\tilde{A}_k^{(1)}$:

$$S_k = \tilde{A}_k^{(4)} - \tilde{A}_k^{(3)} \big(\tilde{A}_k^{(1)}\big)^{-1} \tilde{A}_k^{(2)}. \tag{25}$$

By [4, Theorem 3.9], we know that S_k is positive definite and

$$(U_k^T A U_k)^{-1} = \begin{pmatrix} \big(\tilde{A}_k^{(1)}\big)^{-1} + \big(\tilde{A}_k^{(1)}\big)^{-1} \tilde{A}_k^{(2)} S_k^{-1} \tilde{A}_k^{(3)} \big(\tilde{A}_k^{(1)}\big)^{-1} & -\big(\tilde{A}_k^{(1)}\big)^{-1} \tilde{A}_k^{(2)} S_k^{-1} \\ -S_k^{-1} \tilde{A}_k^{(3)} \big(\tilde{A}_k^{(1)}\big)^{-1} & S_k^{-1} \end{pmatrix}. \tag{26}$$

It follows from the Courant-Fischer min-max theorem that

$$\begin{aligned} \lambda\big(\mathcal{H}_{S_k^{-1}}\big) &\leq \lambda_{\max}\big(\mathcal{H}_{S_k^{-1}}\big) \leq \lambda_{\max}\big(\mathcal{H}_{(U_k^T A U_k)^{-1}}\big) = \lambda_{\max}\big(\mathcal{H}_{A^{-1}}\big), \\ \lambda\big(\mathrm{i}\mathcal{S}_{\mathrm{S_k^{-1}}}\big) &\leq \lambda_{\max}\big(\mathrm{i}\mathcal{S}_{\mathrm{S_k^{-1}}}\big) \leq \lambda_{\max}\big(\mathrm{i}\mathcal{S}_{(\mathrm{U_k^T A U_k})^{-1}}\big) = \lambda_{\max}\big(\mathrm{i}\mathcal{S}_{\mathrm{A^{-1}}}\big). \end{aligned}$$

Similarly, we have

$$\lambda\big(\mathcal{H}_{S_k^{-1}}\big) \geq \lambda_{\min}\big(\mathcal{H}_{A^{-1}}\big) \qquad \text{and} \qquad \lambda\big(\mathrm{i}\mathcal{S}_{\mathrm{S_k^{-1}}}\big) \geq \lambda_{\min}\big(\mathrm{i}\mathcal{S}_{\mathrm{A^{-1}}}\big).$$

Summing up, we have the following results.

Lemma 3.5. The eigenvalues of $\mathcal{H}_{S_k^{-1}}$ and $\mathcal{S}_{S_k^{-1}}$ satisfy the inequalities

$$\lambda_{\min}\big(\mathcal{H}_{A^{-1}}\big) \leq \lambda\big(\mathcal{H}_{S_k^{-1}}\big) \leq \lambda_{\max}\big(\mathcal{H}_{A^{-1}}\big) \qquad \text{and} \qquad \big|\lambda\big(\mathcal{S}_{S_k^{-1}}\big)\big| \leq \rho\big(\mathcal{S}_{A^{-1}}\big).$$

Theorem 3.1. If $A \in \mathbb{R}^{n\times n}$ is positive definite and $\lambda_{\min}(\mathcal{H}_{A^{-1}}) > \rho(\mathcal{S}_{A^{-1}})$, the sequence $\{x_k\}$ produced by SWI converges to the unique solution $x_\star$ of $Ax = b$ (1).

Proof. Let $d_k = x_\star - x_k$ be the error vector. Without loss of generality, we assume that $k > m$. Then $m_k = \min\{k, m\} = m$. From Lemma 3.3 and (23)–(24), we have

$$0 = P_k^T r_k = V_k \begin{pmatrix}\Sigma_k & 0\end{pmatrix} U_k^T r_k = V_k \begin{pmatrix}\Sigma_k & 0\end{pmatrix} \begin{pmatrix}\tilde{r}_k^{(1)} \\ \tilde{r}_k^{(2)}\end{pmatrix} = V_k \Sigma_k \tilde{r}_k^{(1)},$$

which leads to $\tilde{r}_k^{(1)} = 0$.

It follows from SWI, Lemma 3.1, (22), and $Ad_k = r_k$ that

$$\begin{aligned}
d_{k+1}^T A d_{k+1} &= (x_\star - x_{k+1})^T A (x_\star - x_{k+1}) = (x_\star - x_k - \alpha_k p_k)^T A (x_\star - x_k - \alpha_k p_k) \\
&= (d_k - \alpha_k p_k)^T A (d_k - \alpha_k p_k) = d_k^T A d_k - \alpha_k d_k^T A p_k - \alpha_k p_k^T A d_k + \alpha_k^2 p_k^T A p_k \\
&= d_k^T A d_k - \frac{r_k^T p_k}{p_k^T q_k} d_k^T A p_k - \frac{r_k^T p_k}{p_k^T q_k} p_k^T A d_k + \frac{(r_k^T p_k)^2}{(p_k^T q_k)^2} p_k^T A p_k \\
&= d_k^T A d_k - \frac{r_k^T p_k}{p_k^T A p_k} d_k^T A p_k - \frac{r_k^T p_k}{p_k^T A p_k} p_k^T r_k + \frac{(r_k^T p_k)^2}{(p_k^T A p_k)^2} p_k^T A p_k \\
&= d_k^T A d_k - \frac{r_k^T p_k}{p_k^T A p_k} d_k^T A p_k = \left(1 - \frac{r_k^T p_k}{d_k^T A d_k} \frac{d_k^T A p_k}{p_k^T A p_k}\right) d_k^T A d_k \\
&= \left(1 - \frac{r_k^T r_k}{r_k^T A^{-1} r_k} \frac{d_k^T A p_k}{p_k^T A p_k}\right) d_k^T A d_k.
\end{aligned} \tag{27}$$

From (23) and (24), we have

$$\begin{aligned}
&U_k^T A U_k - U_k^T A P_k (P_k^T A P_k)^{-1} P_k^T A U_k \\
&\quad = U_k^T A U_k - U_k^T A U_k \begin{pmatrix}\Sigma_k \\ 0\end{pmatrix} \left(\begin{pmatrix}\Sigma_k & 0\end{pmatrix} U_k^T A U_k \begin{pmatrix}\Sigma_k \\ 0\end{pmatrix} \right)^{-1} \begin{pmatrix}\Sigma_k & 0\end{pmatrix} U_k^T A U_k \\
&\quad = \begin{pmatrix}0 & 0 \\ 0 & S_k\end{pmatrix}.
\end{aligned} \tag{28}$$

Note that U_k is unitary. By SWI, Lemma 3.1, (26), and (28), we obtain

$$\begin{aligned}
U_k^T A p_k &= U_k^T A \big(I - P_k (P_k^T A P_k)^{-1} P_k^T A\big) r_k \\
&= \big(U_k^T A U_k - U_k^T A P_k (P_k^T A P_k)^{-1} P_k^T A U_k\big) U_k^T r_k \\
&= \begin{pmatrix}0 & 0 \\ 0 & S_k\end{pmatrix} \begin{pmatrix}\tilde{r}_k^{(1)} \\ \tilde{r}_k^{(2)}\end{pmatrix} = \begin{pmatrix}0 \\ S_k \tilde{r}_k^{(2)}\end{pmatrix}
\end{aligned}$$

$$\text{and} \quad U_k^T p_k = U_k^T A^{-1} U_k U_k^T A p_k = (U_k^T A U_k)^{-1} U_k^T A p_k = \begin{pmatrix}-\big(\tilde{A}_k^{(1)}\big)^{-1} \tilde{A}_k^{(2)} \tilde{r}_k^{(2)} \\ \tilde{r}_k^{(2)}\end{pmatrix}.$$

This together with (26) and $\tilde{r}_k^{(1)} = 0$ yields

$$\begin{aligned}
p_k^T A p_k &= (U_k^T p_k)^T U_k^T A p_k = \begin{pmatrix}-\big(\tilde{r}_k^{(2)}\big)^T \big(\tilde{A}_k^{(2)}\big)^T \big(\tilde{A}_k^{(1)}\big)^{-T} & \big(\tilde{r}_k^{(2)}\big)^T\end{pmatrix} \begin{pmatrix}0 \\ S_k \tilde{r}_k^{(2)}\end{pmatrix} \\
&= \big(\tilde{r}_k^{(2)}\big)^T S_k \tilde{r}_k^{(2)},
\end{aligned} \tag{29}$$

$$d_k^T A p_k = r_k^T A^{-T} A p_k = r_k^T U_k U_k^T A^{-T} U_k U_k^T A p_k$$
$$= (U_k^T r_k)^T (U_k^T A U_k)^{-T} U_k^T A p_k = \left(\tilde{r}_k^{(2)}\right)^T S_k^{-T} S_k \tilde{r}_k^{(2)}. \tag{30}$$

Substituting (29)–(30) into (27) gives

$$d_{k+1}^T A d_{k+1} = \left(1 - \frac{r_k^T r_k}{r_k^T A^{-1} r_k} \frac{\left(\tilde{r}_k^{(2)}\right)^T S_k^{-T} S_k \tilde{r}_k^{(2)}}{\left(\tilde{r}_k^{(2)}\right)^T S_k \tilde{r}_k^{(2)}}\right) d_k^T A d_k. \tag{31}$$

Let $v = S_k \tilde{r}_k^{(2)} \in \mathbb{R}^{n-m-1}$. It follows from Lemma 3.5 and $\lambda_{\min}(\mathcal{H}_{A^{-1}}) > \rho(\mathcal{S}_{A^{-1}})$ that

$$\frac{\left(\tilde{r}_k^{(2)}\right)^T S_k^{-T} S_k \tilde{r}_k^{(2)}}{\left(\tilde{r}_k^{(2)}\right)^T S_k \tilde{r}_k^{(2)}} = \frac{v^T (S_k^{-T})^2 v}{v^T S_k^{-T} v} = \frac{v^T (S_k^{-1})^2 v}{v^T S_k^{-1} v} = \frac{v^T \left(\mathcal{H}_{S_k^{-1}} + \mathcal{S}_{S_k^{-1}}\right)^2 v}{v^T \mathcal{H}_{S_k^{-1}} v}$$
$$= \frac{v^T \mathcal{H}_{S_k^{-1}}^2 v + v^T \mathcal{S}_{S_k^{-1}}^2 v}{v^T \mathcal{H}_{S_k^{-1}} v} \geq \lambda_{\min}(\mathcal{H}_{S_k^{-1}}) - \frac{\max\left\{\left|\lambda_{\min}(\mathcal{S}_{S_k^{-1}}^2)\right|, \left|\lambda_{\max}(\mathcal{S}_{S_k^{-1}}^2)\right|\right\}}{\lambda_{\min}(\mathcal{H}_{S_k^{-1}})}$$
$$\geq \lambda_{\min}(\mathcal{H}_{A^{-1}}) - \frac{\rho(\mathcal{S}_{A^{-1}})^2}{\lambda_{\min}(\mathcal{H}_{A^{-1}})} > 0.$$

This along with the fact that

$$\frac{r_k^T r_k}{r_k^T A^{-1} r_k} = \frac{r_k^T r_k}{r_k^T \mathcal{H}_{A^{-1}} r_k} \geq \frac{1}{\lambda_{\max}(\mathcal{H}_{A^{-1}})} > 0$$

leads to

$$d_{k+1}^T A d_{k+1} \leq \left(1 - \frac{\left(\lambda_{\min}(\mathcal{H}_{A^{-1}})\right)^2 - \rho(\mathcal{S}_{A^{-1}})^2}{\lambda_{\min}(\mathcal{H}_{A^{-1}}) \lambda_{\max}(\mathcal{H}_{A^{-1}})}\right) d_k^T A d_k.$$

Hence when $\lambda_{\min}(\mathcal{H}_{A^{-1}}) > \rho(\mathcal{S}_{A^{-1}})$, $d_k^T A d_k \to 0$ as $k \to \infty$. Since A is positive definite, we have $d_k \to 0$. □

If A is symmetric positive definite, $\mathcal{S}_{A^{-1}} = 0$ and $\lambda_{\min}(\mathcal{H}_{A^{-1}}) > \rho(\mathcal{S}_{A^{-1}})$ holds naturally, and SWI converges unconditionally.

If A is a normal matrix, we have $A = X^* \Lambda X$, where X is a unitary matrix and $\Lambda = \operatorname{diag}\{a_1 + \mathrm{i}b_1, \ldots, a_n + \mathrm{i}b_n\}$ with $0 < a_1 \leq a_2 \leq \cdots \leq a_n$ is the diagonal matrix containing the eigenvalues of A. Then

$$\lambda_{\min}(\mathcal{H}_{A^{-1}}) = \frac{1}{2} \min_j \left\{\frac{1}{a_j + \mathrm{i}b_j} + \frac{1}{a_j - \mathrm{i}b_j}\right\} = \min_j \left\{\frac{a_j}{a_j^2 + b_j^2}\right\},$$
$$\rho(\mathcal{S}_{A^{-1}}) = \frac{1}{2} \max_j \left|\frac{1}{a_j + \mathrm{i}b_j} - \frac{1}{a_j - \mathrm{i}b_j}\right| = \max_j \left\{\frac{|b_j|}{a_j^2 + b_j^2}\right\}.$$

If $|b_j| \ll a_j$ for all $1 \leq j \leq n$, we have

$$\lambda_{\min}(\mathcal{H}_{A^{-1}}) \approx \frac{1}{a_n} \qquad \text{and} \qquad \rho(\mathcal{S}_{A^{-1}}) \approx \max_j \left\{\frac{|b_j|}{a_j^2}\right\} \leq \frac{1}{a_1} \max_j \left\{\frac{|b_j|}{a_j}\right\}.$$

Then from Theorem 3.1 we know that SWI is convergent provided

$$\frac{a_1}{a_n} > \max_j \left\{\frac{|b_j|}{a_j}\right\}.$$

Algorithm 5 DIOM [23, Algorithm 6.8]

Given $\hat{x}_0 = 0$ and a positive integer m, set $\hat{r}_0 = b$, $\beta = \|\hat{r}_0\|$, $\hat{v}_1 = \hat{r}_0/\beta$.
for $k = 1, 2, \ldots$ **do**
  Compute h_{ik}, $i = \max\{1, k-m+1\}, 2, \ldots, k+1$ and $\hat{v}_{k+1}$ using the incomplete orthogonalization process [23, Algorithm 6.6].
  Update the LU factorization of H_k, i.e., obtain the last column u_k of $\hat{U}_k$.
  Compute $\zeta_k = \{\text{if } k = 1 \text{ then } \beta, \text{else } -l_{k,k-1}\zeta_{k-1}\}$.
  Compute $\hat{p}_k = (\hat{v}_k - \sum_{i=k-m+1}^{k-1} u_{ik}\hat{p}_i)/u_{kk}$.
  Compute $\hat{x}_k = \hat{x}_{k-1} + \zeta_k \hat{p}_k$.
  Compute $\hat{r}_k = \hat{r}_{k-1} - \zeta_k A\hat{p}_k$.
end for

3.1 SWI is not equivalent to DIOM

Although SCG and FOM are equivalent, their sliding window implementations are different. DIOM, the sliding window implementation of FOM, is stated as Algorithm 5.

From Algorithms 4 and 5 we see that the main difference between them is that in DIOM, Saad [23] applies the sliding window idea to the Arnoldi vectors, whereas in SWI, we apply it to the transformed search directions $\{p_k\}$. As the following simple example shows, the directions $\{\hat{p}_k\}$ produced by DIOM do not satisfy

$$\hat{p}_i^T A\hat{p}_k = 0 \qquad (\max\{0,\, k-m\} \le i \le k-1), \tag{32}$$

which establishes that SWI and DIOM are different.

Consider

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 2 \end{pmatrix} \qquad \text{and} \qquad b = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}. \tag{33}$$

Note that A is positive definite. With $m = 2$ and $\hat{x}_0 = 0$, we have $\hat{r}_0 = b$ and $\beta = \|\hat{r}_0\| = \sqrt{3}$. It follows from DIOM that

$$H_4 = \begin{pmatrix} 1 & -\sqrt{\frac{2}{3}} & 0 & 0 \\ \sqrt{\frac{2}{3}} & \frac{3}{2} & -\frac{1}{2\sqrt{21}} & 0 \\ 0 & \frac{\sqrt{21}}{6} & \frac{17}{14} & -\frac{9}{7\sqrt{6}} \\ 0 & 0 & \frac{2\sqrt{6}}{7} & \frac{29}{28} \end{pmatrix}, \quad \hat{V}_4 = \begin{pmatrix} \frac{1}{\sqrt{3}} & 0 & -\frac{2}{\sqrt{42}} & -\frac{3}{2\sqrt{7}} \\ \frac{1}{\sqrt{3}} & 0 & -\frac{2}{\sqrt{42}} & \frac{2}{\sqrt{7}} \\ \frac{1}{\sqrt{3}} & 0 & \frac{4}{\sqrt{42}} & -\frac{1}{2\sqrt{7}} \\ 0 & \frac{1}{\sqrt{2}} & -\frac{3}{\sqrt{42}} & -\frac{1}{2\sqrt{7}} \\ 0 & \frac{1}{\sqrt{2}} & \frac{3}{\sqrt{42}} & \frac{1}{2\sqrt{7}} \end{pmatrix}.$$

The LU factors of H_4 are

$$\hat{L}_4 = \begin{pmatrix} 1 & & & \\ \sqrt{\frac{2}{3}} & 1 & & \\ 0 & \frac{\sqrt{21}}{13} & 1 & \\ 0 & 0 & \frac{26\sqrt{6}}{114} & 1 \end{pmatrix}, \quad \hat{U}_4 = \begin{pmatrix} 1 & -\sqrt{\frac{2}{3}} & 0 & 0 \\ & \frac{13}{6} & -\frac{1}{2\sqrt{21}} & 0 \\ & & \frac{114}{91} & -\frac{9}{7\sqrt{6}} \\ & & & \frac{101}{76} \end{pmatrix}.$$

Note that $\hat{P}_4 = [\hat{p}_1, \hat{p}_2, \hat{p}_3, \hat{p}_4] = \hat{V}_4\hat{U}_4^{-1}$ [23, p155]. Thus,

$$\hat{p}_1 = \tfrac{1}{\sqrt{3}}\begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \quad \hat{p}_2 = \tfrac{2\sqrt{2}}{13}\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \quad \hat{p}_3 = \tfrac{\sqrt{42}}{114}\begin{pmatrix} -4 \\ -4 \\ 9 \\ -6 \\ 7 \end{pmatrix}, \quad \hat{p}_4 = \tfrac{2}{101\sqrt{7}}\begin{pmatrix} -69 \\ 64 \\ 8 \\ -37 \\ 40 \end{pmatrix}.$$

Since $\hat{p}_3^T A\hat{p}_4 = \frac{\sqrt{42}}{114} \cdot \frac{2}{101\sqrt{7}} \cdot 19 = \frac{\sqrt{6}}{303} \neq 0$, we know that DIOM does not possess the properties in (32).

In addition, the SWI residuals do not satisfy

$$r_i^T r_k = 0 \qquad (\max\{0,\, k-m\} \leq i \leq k-1),$$

a property that the DIOM residuals possess [23, p157]. Indeed, for the same linear system (33) and the same setting $m = 2$ and $x_0 = 0$, the SWI residuals $\{r_k\}$ are

$$r_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ -1 \\ -1 \end{pmatrix},\ r_2 = \frac{1}{13}\begin{pmatrix} -2 \\ -2 \\ 4 \\ -3 \\ 3 \end{pmatrix},\ r_3 = \frac{1}{19}\begin{pmatrix} 3 \\ -4 \\ 1 \\ 1 \\ -1 \end{pmatrix},\ r_4 = \frac{1}{15}\begin{pmatrix} -1 \\ 0 \\ 1 \\ 1 \\ -1 \end{pmatrix},\ r_5 = \frac{1}{289}\begin{pmatrix} -1 \\ 2 \\ -1 \\ 13 \\ 13 \end{pmatrix}.$$

This implies that $r_3^T r_5 = -\frac{12}{5491} \neq 0$.

4 Numerical experiments

We report numerical experience with SCG (Algorithm 1), SWI (Algorithm 4), FOM (Algorithm 3), and DIOM (Algorithm 5). For completeness, we include results obtained with GMRES [24], DQGMRES [25], and BICGSTAB [30]. All experiments were run using MATLAB R2015b on a PC with an Intel(R) Core(TM) i7-8550U CPU @ 1.8GHz and 16GB of RAM.

In our implementation, $x_0 = 0$. Each method is terminated when either the number of iterations exceeds 10^4 or

$$\text{Res} := \frac{\|r_k\|}{\|r_0\|} < 10^{-6}.$$

We compare the performance by reporting the number of iterations, the CPU time and the relative residual, denoted by "Iter", "CPU" and "Res", respectively. For SWI, DIOM, and DQGMRES, we tested several values of the memory m, and denote the corresponding algorithms SWI(m), DIOM(m), and DQGMRES(m), respectively.

Example 1. [13, Example 3.1.1] We consider the 2D convection-diffusion equation

$$-\epsilon\nabla^2 u + \vec{w} \cdot \nabla u = 0 \quad \text{in } (-1,\, 1) \times (-1,\, 1),$$

with boundary conditions

$$u(x,-1) = x, \quad u(x,1) = 0, \quad u(-1,y) \approx -1, \quad u(1,y) \approx 1.$$

If $\vec{w} = (0, 1)$, an exact solution is

$$u(x,y) = x(1 - e^{\frac{y-1}{\epsilon}})/(1 - e^{-\frac{2}{\epsilon}}),$$

which satisfies the boundary conditions, save for the last two near $y = 1$.

In our tests, we set $\epsilon = 1/200$ and discretize the convection-diffusion equation using the standard Q1 finite element approximation [14] on uniform grids with grid parameter $h = 1/2^5$, $1/2^6$, $1/2^7$, $1/2^8$, $1/2^9$, $1/2^{10}$. The resulting matrices are unsymmetric positive definite. This discretization was accomplished using IFISS [14]. We use $m = 2$, 5, and 10 in SWI, DIOM, and DQGMRES. We report our numerical results in Figure 1 and Table 1.

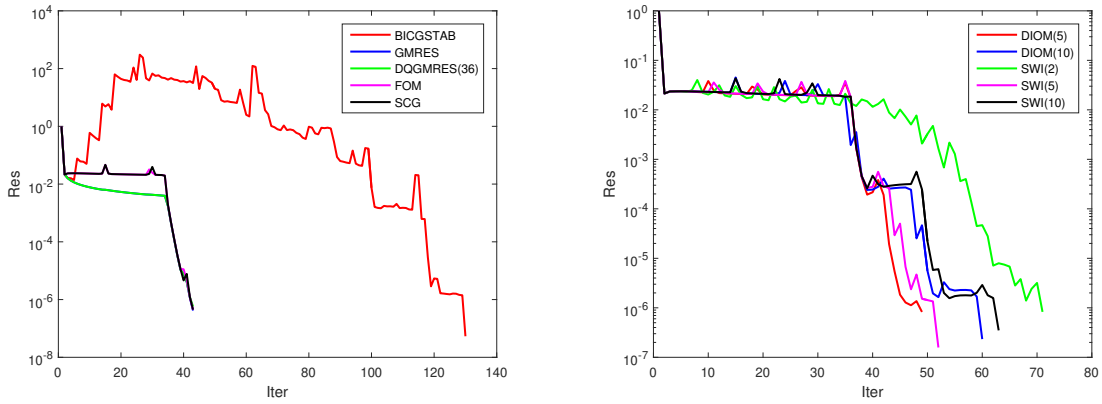


Figure 1: Evolution of the relative residual of the methods tested on Example 1 with $n = 1089$

Table 1: Numerical results for Example 1

h		$1/2^5$	$1/2^6$	$1/2^7$	$1/2^8$	$1/2^9$	$1/2^{10}$
n		1089	4225	16641	66049	263169	1050625
GMRES	Iter	43	75	149	298	597	1193
	CPU	0.015	0.20	1.21	14.40	347.50	36374.39
	Res	4.17e-07	8.11e-07	4.1934e-07	7.98e-07	7.73e-07	8.35e-07
FOM	Iter	43	75	149	298	598	1195
	CPU	0.010	0.18	1.17	17.46	336.75	36299.16
	Res	4.47e-07	8.25e-07	4.35e-07	9.77e-07	6.46e-07	9.09e-07
DIOM(5)	Iter	49	89	179	363	750	1544
	CPU	0.0039	0.049	0.42	2.81	26.86	192.49
	Res	8.25e-07	7.80e-07	5.42e-07	8.97e-07	5.99e-07	9.60e-07
DIOM(10)	Iter	60	99	166	318	621	1237
	CPU	0.0065	0.062	0.75	4.01	35.74	242.13
	Res	2.33e-07	6.89e-08	5.73e-07	3.87e-07	6.74e-07	9.26e-07
SCG	Iter	43	75	148	297	594	1185
	CPU	0.0036	0.043	0.47	5.25	75.31	21488.05
	Res	4.47e-07	8.25e-07	4.46e-07	7.24e-07	6.56e-07	9.23e-07
SWI(2)	Iter	71	121	210	417	803	1598
	CPU	0.0042	0.049	0.27	1.75	16.90	103.43
	Res	8.28e-07	8.54e-07	8.87e-07	5.65e-07	7.04e-07	9.68e-07
SWI(5)	Iter	52	95	181	360	728	1435
	CPU	0.0042	0.036	0.29	2.33	21.75	145.28
	Res	1.58e-07	4.29e-08	8.15e-07	6.33e-07	8.16e-07	9.29e-07
SWI(10)	Iter	63	102	178	348	691	1381
	CPU	0.0048	0.039	0.57	3.53	26.07	205.36
	Res	3.50e-07	2.97e-07	1.87e-07	5.50e-07	9.58e-07	7.70e-07

In Example 1, BICGSTAB and DIOM(2) were not able to solve problems within 10^4 iterations when $h \leq 1/2^6$ (DIOM(2) also failed when $h = 1/2^5$), so we do not report their results in Table 1. DQGMRES failed for $m = 2$, 5, and 10. We found that its performance is highly sensitive to the value of m. Indeed, while testing other values of m, we observed that the choice of $m = 36$ is effective but $m = 35$ is not when $h = 1/2^5$. The value of m for DQGMRES on this example should not be too much smaller than the number of iterations of GMRES, which is clearly not practical. Figure 1 also illustrates that the convergence behaviors of SWI and DIOM are different.

In Table 1, the iteration numbers for all tested methods increase in a regular way, each time nearly twice its previous value, but the CPU times rise sharply. When $h = 1/2^{10}$ ($n = 1,050,625$), the CPU times of GMRES, FOM, and SCG are more than 100 times those of the sliding window versions. When $h \leq 1/2^7$, the best performances are by SWI(2). SWI becomes increasingly better as the problem size increases.

Example 2. We select matrices from the SuiteSparse Matrix Collection [10, 11] and set b so that the solution is $x_\star = (1, 1, \ldots, 1)$.

The total number of tested matrices in Example 2 is 24, where the matrices arise from applications such as computational fluid dynamics, circuit simulation, directed weighted graphs, optimization, and power networks. Their name, dimensions and nature are given in Table 2, where SPD, UPD and UID mean that the matrix is symmetric positive definite, unsymmetric positive definite and unsymmetric indefinite.

Table 2: Dimensions and nature of 24 problems in Example 2

Problem	n	Nature	Problem	n	Nature
ACTIVSg10K	20000	UPD	fpga_dcop_35	1220	UPD
ACTIVSg2000	4000	UPD	majorbasis	160000	UPD
add20	2395	UPD	pde2961	2961	UPD
add32	4960	UPD	raefsky2	3242	UID
adder_dcop_01	1813	UPD	raefsky4	19779	SPD
cage12	130228	UPD	raefsky5	6316	UPD
cage13	445315	UPD	rajat01	6833	UID
crashbasis	160000	UID	rajat03	7602	UPD
ex11	16614	UPD	rajat13	7598	UID
ex18	5773	UPD	rajat16	94294	UPD
ex19	12005	UPD	rajat27	20640	UID
ex35	19716	UPD	swang1	3169	UPD

For DQGMRES, DIOM and SWI, we use $m = 2$, 5, 10, and 100. The best of the four results is presented along with the corresponding value of m. The numerical results are reported in Figures 2 and 3 and Tables 3 to 9, where "-" means that the method failed to solve the problem.

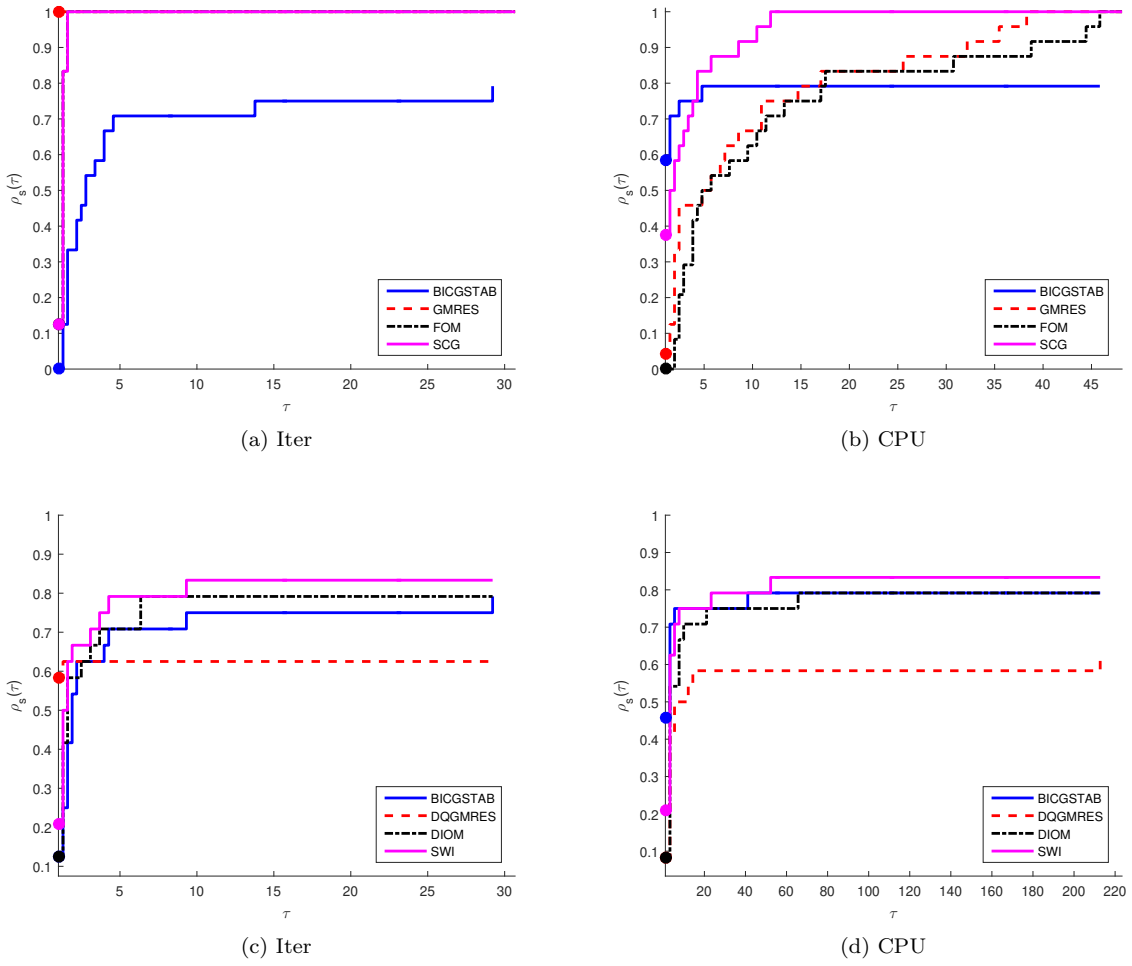
GMRES, FOM and SCG successfully solved all the problems but BICGSTAB, DQGMRES, DIOM and SWI failed in 5, 9, 5 and 4 cases, respectively. In terms of the CPU time, BICGSTAB, SCG and SWI perform best in 10, 7, and 4 cases, respectively. Compared to SCG, SWI requires less CPU time in 14 cases and the improvements are significant. Compared to DQGMRES and DIOM, SWI requires the least CPU time in 16 cases, while DQGMRES and DIOM require the least in only 2 cases. Hence, SWI is the most successful of the sliding window implementations.

In Figure 2, performance profiles[2] [12] indicate that SCG and SWI are more robust than BICGSTAB, and also more efficient than other tested methods; the reduction in CPU time for SWI was often substantial. To see the role of m in SWI's performance, we also plot performance profiles for SWI with different m. From Figure 3 it is apparent that larger m leads to fewer iterations for SWI but more CPU time. Hence, the choice of m to balance iterations and CPU time is crucial for the performance of SWI.

[2]The performance profile $\varrho_s(\tau)$ is a distribution function for the performance ratio $r_{p,s}$, with

$$r_{p,s} = t_{p,s} / \min\{t_{p,s} : s \in S\} \qquad \text{and} \qquad \varrho_s(\tau) = \big|\{p \in P : r_{p,s} \leq \tau\}\big| / |P|,$$

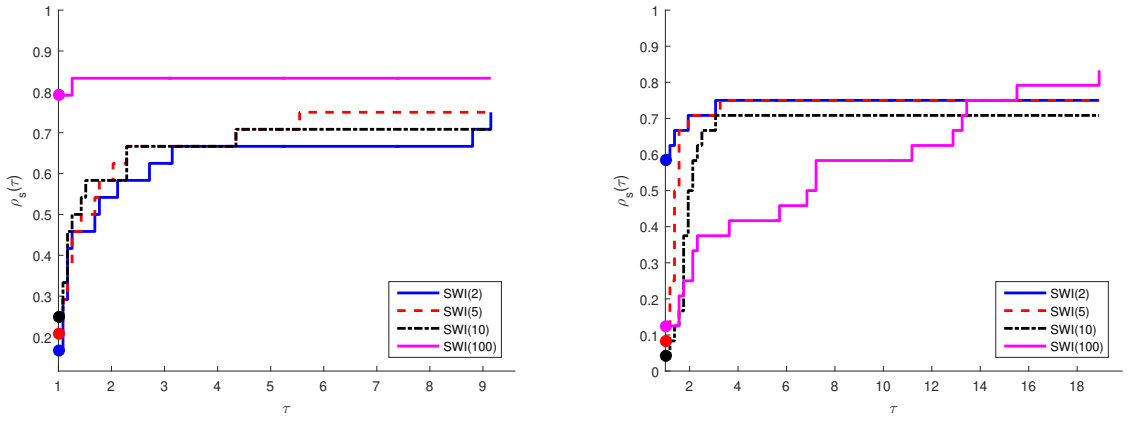
where S is the set of solvers, P is the set of problems, $|\cdot|$ indicates cardinality, and $t_{p,s}$ is the Iter/CPU required to solve problem p with solver s.



(a) Iter (b) CPU

(c) Iter (d) CPU

Figure 2: Performance profiles for all tested methods on Example 2. Limited-memory methods use the value of m stated in Tables 7 to 9



(a) Iter (b) CPU

Figure 3: Performance profiles for SWI with different values of m on Example 2

Table 3: Numerical results for BICGSTAB on Example 2

Problem	Iter	CPU	Res	Problem	Iter	CPU	Res
ACTIVSg10K	-	-	-	fpga_dcop_35	-	-	-
ACTIVSg2000	2608	0.42	3.04E-07	majorbasis	110	0.71	8.20E-07
add20	376	0.053	8.59E-07	pde2961	267	0.021	9.12E-07
add32	72	0.0086	8.44E-07	raefsky2	636	0.32	6.92E-07
adder_dcop_01	1607	0.072	9.96E-07	raefsky4	29	0.086	7.01E-07
cage12	20	0.11	5.71E-07	raefsky5	129	0.036	7.49E-07
cage13	20	0.40	4.76E-07	rajat01	-	-	-
crashbasis	244	1.57	9.74E-07	rajat03	2338	0.56	4.08E-07
ex11	1572	3.56	8.94E-07	rajat13	86	0.014	9.52E-07
ex18	1397	0.25	9.89E-07	rajat16	-	-	-
ex19	4050	2.88	3.79E-07	rajat27	-	-	-
ex35	1438	1.17	8.15E-07	swang1	22	0.0017	6.14E-07

Table 4: Numerical results for GMRES on Example 2

Problem	Iter	CPU	Res	Problem	Iter	CPU	Res
ACTIVSg10K	6130	2690.28	9.98E-07	fpga_dcop_35	214	0.29	9.37E-07
ACTIVSg2000	779	16.03	9.71E-07	majorbasis	97	7.49	9.81E-07
add20	195	0.35	9.92E-07	pde2961	189	0.67	9.11E-07
add32	57	0.091	9.42E-07	raefsky2	332	2.71	8.61E-07
adder_dcop_01	55	0.023	9.39E-07	raefsky4	22	0.085	4.73E-07
cage12	14	0.16	9.57E-07	raefsky5	34	0.048	8.49E-07
cage13	15	0.65	8.33E-07	rajat01	1894	132.19	9.99E-07
crashbasis	175	22.80	9.93E-07	rajat03	172	1.32	8.98E-07
ex11	571	24.24	9.98E-07	rajat13	22	0.022	9.06E-07
ex18	505	8.75	9.99E-07	rajat16	1116	355.88	9.97E-07
ex19	936	48.60	9.31E-07	rajat27	595	30.20	9.89E-07
ex35	607	29.71	9.99E-07	swang1	19	0.0076	7.15E-07

Table 5: Numerical results for FOM on Example 2

Problem	Iter	CPU	Res	Problem	Iter	CPU	Res
ACTIVSg10K	6376	2762.76	9.48E-07	fpga_dcop_35	276	0.42	5.26E-07
ACTIVSg2000	814	18.55	9.75E-07	majorbasis	109	9.36	8.54E-07
add20	214	0.40	8.59E-07	pde2961	192	0.63	8.36E-07
add32	59	0.095	6.06E-07	raefsky2	334	2.93	9.72E-07
adder_dcop_01	80	0.055	9.41E-07	raefsky4	22	0.16	5.06E-07
cage12	15	0.27	5.12E-07	raefsky5	36	0.077	7.37E-07
cage13	15	0.84	9.47E-07	rajat01	2296	185.61	5.60E-07
crashbasis	196	27.43	9.89E-07	rajat03	173	1.28	7.55E-07
ex11	694	35.94	9.96E-07	rajat13	24	0.025	7.72E-07
ex18	541	9.64	9.62E-07	rajat16	1453	580.32	9.89E-07
ex19	947	47.85	9.41E-07	rajat27	909	68.95	9.45E-07
ex35	847	53.71	9.35E-07	swang1	19	0.0075	8.26E-07

Table 6: Numerical results for SCG on Example 2

Problem	Iter	CPU	Res	Problem	Iter	CPU	Res
ACTIVSg10K	6376	1266.00	9.50E-07	fpga_dcop_35	276	0.13	5.26E-07
ACTIVSg2000	814	4.34	9.75E-07	majorbasis	109	2.50	8.54E-07
add20	214	0.084	8.59E-07	pde2961	192	0.12	8.36E-07
add32	59	0.020	6.06E-07	raefsky2	334	0.57	9.72E-07
adder_dcop_01	80	0.016	9.41E-07	raefsky4	22	0.094	5.06E-07
cage12	15	0.14	5.12E-07	raefsky5	36	0.027	7.37E-07
cage13	15	0.45	9.47E-07	rajat01	2294	70.42	9.75E-07
crashbasis	196	6.14	9.89E-07	rajat03	173	0.24	7.60E-07
ex11	694	8.97	9.96E-07	rajat13	24	0.013	7.72E-07
ex18	541	2.08	9.62E-07	rajat16	1453	158.62	9.98E-07
ex19	947	12.28	9.41E-07	rajat27	912	16.63	7.96E-07
ex35	847	13.47	9.33E-07	swang1	19	0.0056	8.26E-07

Table 7: Numerical results for DQGMRES on Example 2

Problem	Iter	CPU	Res	m	Problem	Iter	CPU	Res	m
ACTIVSg10K	-	-	-	-	fpga_dcop_35	97	7.35	9.81E-07	100
ACTIVSg2000	-	-	-	-	majorbasis	-	-	-	-
add20	206	0.46	9.98E-07	100	pde2961	-	-	-	-
add32	57	0.094	9.42E-07	100	raefsky2	-	-	-	-
adder_dcop_01	55	0.022	9.39E-07	100	raefsky4	22	0.093	4.73E-07	5
cage12	15	0.25	5.65E-07	5	raefsky5	34	0.045	8.49E-07	100
cage13	16	0.79	7.25E-07	5	rajat01	-	-	-	-
crashbasis	-	-	-	-	rajat03	257	0.17	9.86E-07	5
ex11	893	4.87	9.87E-07	5	rajat13	45	0.071	8.34E-07	10
ex18	798	0.53	9.90E-07	5	rajat16	-	-	-	-
ex19	1897	3.54	9.78E-07	5	rajat27	-	-	-	-
ex35	347	0.029	9.83E-07	5	swang1	19	0.0080	7.44E-07	10

Table 8: Numerical results for DIOM on Example 2

Problem	Iter	CPU	Res	m	Problem	Iter	CPU	Res	m
ACTIVSg10K	-	-	-	-	fpga_dcop_35	587	0.035	6.07E-07	2
ACTIVSg2000	-	-	-	-	majorbasis	268	3.98	9.83E-07	2
add20	224	0.038	9.96E-07	5	pde2961	384	0.18	9.58E-07	10
add32	59	0.015	6.26E-07	2	raefsky2	3883	6.05	9.63E-07	10
adder_dcop_01	80	0.049	9.41E-07	100	raefsky4	22	0.091	5.06E-07	2
cage12	15	0.17	5.86E-07	2	raefsky5	54	0.068	8.50E-07	10
cage13	16	0.52	4.63E-07	2	rajat01	-	-	-	-
crashbasis	721	10.91	9.96E-07	2	rajat03	281	0.12	6.78E-07	2
ex11	1237	6.04	9.58E-07	2	rajat13	55	0.10	9.54E-07	10
ex18	943	0.45	9.80E-07	2	rajat16	-	-	-	-
ex19	2181	2.97	8.87E-07	2	rajat27	-	-	-	-
ex35	1235	1.90	8.03E-07	2	swang1	19	0.0049	9.46E-07	5

Table 9: Numerical results for SWI on Example 2

Problem	Iter	CPU	Res	m	Problem	Iter	CPU	Res	m
ACTIVSg10K	-	-	-	-	fpga_dcop_35	903	0.053	4.54E-07	2
ACTIVSg2000	1764	9.35	9.94E-07	100	majorbasis	130	1.44	9.78E-07	2
add20	234	0.033	9.46E-07	2	pde2961	255	0.076	9.21E-07	10
add32	59	0.014	6.23E-07	2	raefsky2	2683	1.95	9.95E-07	5
adder_dcop_01	80	0.037	9.41E-07	100	raefsky4	22	0.059	5.06E-07	2
cage12	15	0.13	5.28E-07	2	raefsky5	95	0.046	8.29E-07	2
cage13	15	0.40	9.60E-07	2	rajat01	-	-	-	-
crashbasis	428	7.52	9.72E-07	5	rajat03	341	0.12	7.50E-07	2
ex11	1235	3.38	9.05E-07	2	rajat13	53	0.038	9.86E-07	10
ex18	908	0.36	9.64E-07	2	rajat16	-	-	-	-
ex19	2188	2.22	8.52E-07	2	rajat27	-	-	-	-
ex35	1242	1.48	9.36E-07	2	swang1	20	0.0038	9.31E-07	2

5 Conclusions and future work

We introduced the semi-conjugate gradient method (SCG) and its sliding window implementation (SWI) to solve unsymmetric positive definite linear systems. Both theoretical and numerical studies of SCG and SWI were conducted. SCG is theoretically equivalent to FOM, but a counter-example illustrates that their sliding window implementations differ. The numerical results presented are highly encouraging, though the performance of SWI naturally depends on the window width m.

Future work should aim to develop efficient algorithms for adaptive selection of the window width m. A possibly feasible approach is changing the value of m dynamically. Another way to improve the performance of SCG and SWI is to incorporate preconditioning into them and to develop practical and effective preconditioners. It is also interesting and challenging to extend SCG and SWI to solve nonlinear problems as has been done for CG [8].

References

[1] W. E. Arnoldi. The principle of minimized iterations in the solution of the matrix eigenvalue problem. Quarterly Appl. Math., 9(1):17–29, 1951.

[2] O. Axelsson. Conjugate gradient type methods for unsymmetric and inconsistent systems of linear equations. Linear Algebra and its Applications, 29:1–16, 1980.

[3] O. Axelsson. A generalized conjugate direction method and its application on a singular perturbation problem. In Numerical Analysis, pages 1–11. Springer, Berlin, Heidelberg, 1980.

[4] O. Axelsson. Iterative Solution Methods. Cambridge University Press, 1996.

[5] R. E. Bank and T. F. Chan. An analysis of the composite step biconjugate gradient method. Numer. Math., 66(1):295–319, 1993.

[6] O. Burdakov. Conjugate direction methods for solving systems of equations and finding saddle points. part 1. Engineering Cybernetics, 20(3):13–19, 1982.

[7] O. Burdakov. Conjugate direction methods for solving systems of equations and finding saddle points. part 2. Engineering Cybernetics, 20(4):23–31, 1982.

[8] O. Burdakov, Y. H. Dai, and N. Huang. On solving saddle-point problems and nonlinear monotone equations. http://stanford.edu/group/SOL/classics/18oleg-SCG-ismp-bordeaux.pdf. Presentation at ISMP 2018, Bordeaux, France.

[9] Y. H. Dai and J. Y. Yuan. Study on semi-conjugate direction methods for non-symmetric systems. International J. Numer. Meth. Eng., 60(8):1383–1399, 2004.

[10] T. A. Davis and Y. F. Hu. The University of Florida sparse matrix collection. ACM Trans. Math. Softw., 38(1):1–25, 2011.

[11] T. A. Davis, Y. Hu, and S. Kolodziej. The SuiteSparse matrix collection. https://sparse.tamu.edu/, 2015–present.

[12] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. Math. Program., 91(2):201–213, 2002.

[13] H. C. Elman, D. J. Silvester, and A. J. Wathen. Finite Elements and Fast Iterative Solvers: with Applications in Incompressible Fluid Dynamics. Oxford University Press, Oxford, UK, 2 edition, 2006.

[14] H. C. Elman, A. Ramage, and D. J. Silvester. Algorithm 866: IFISS, a Matlab toolbox for modelling incompressible flow. ACM Trans. Math. Softw., 33(2):14–es, 2007.

[15] D. C.-L. Fong and M. Saunders. LSMR: An iterative algorithm for least-squares problems. SIAM J. Sci. Comput., 33(5):2950–2971, 2011. doi: https://doi.org/10.1137/10079687X.

[16] R. W. Freund and N. M. Nachtigal. QMR: A quasi-minimal residual method for non-Hermitian linear systems. Numer. Math., 60(1):315–339, 1991.

[17] G. H. Golub and C. Van Loan. Unsymmetric positive definite linear systems. Linear Algebra and its Applications, 28:85–97, 1979.

[18] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. J. Res. Nat. Bur. Standards, 49(6):409–435, 1952.

[19] T. A. Manteuffel. The Tchebychev iteration for nonsymmetric linear systems. Numer. Math., 28(3): 307–327, 1977.

[20] T. A. Manteuffel. Adaptive procedure for estimating parameters for the nonsymmetric Tchebychev iteration. Numer. Math., 31(2):183–208, 1978.

[21] A. Montoison and D. Orban. BiLQ: An iterative method for nonsymmetric linear systems with a quasi-minimum error property. SIAM J. Matrix Anal. Appl., 41(3):1145–1166, 2020.

[22] C. C. Paige and M. A. Saunders. LSQR: An algorithm for sparse linear equations and sparse least squares. ACM Trans. Math. Softw., 8(1):43–71, 1982.

[23] Y. Saad. Iterative Methods for Sparse Linear Systems. SIAM, Philadelphia, 2003.

[24] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM J. Sci. and Statist. Comput., 7(3):856–869, 1986.

[25] Y. Saad and K. S. Wu. DQGMRES: A direct quasi-minimal residual algorithm based on incomplete orthogonalization. Numer. Linear Algebra Appl., 3(4):329–343, 1996.

[26] M. A. Saunders, H. D. Simon, and E. L. Yip. Two conjugate-gradient-type methods for unsymmetric linear equations. SIAM J. Numer. Anal., 25(4):927–940, 1988.

[27] P. Sonneveld. CGS, a fast Lanczos-type solver for nonsymmetric linear systems. SIAM J. Sci. and Statist. Comput., 10(1):36–52, 1989.

[28] C. Tong and Q. Ye. Analysis of the finite precision bi-conjugate gradient algorithm for nonsymmetric linear systems. Math. Comp., 69(232):1559–1575, 2000.

[29] H. A. Van der Vorst. Iterative solution methods for certain sparse linear systems with a non-symmetric matrix arising from PDE-problems. J. Comp. Physics, 44(1):1–19, 1981.

[30] H. A. Van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. SIAM J. Sci. and Statist. Comput., 13(2):631–644, 1992.

[31] P. K. Vinsome. Orthomin, an iterative method for solving sparse sets of simultaneous linear equations. In SPE Symposium on Numerical Simulation of Reservoir Performance. OnePetro, 1976.

[32] D. M. Young and K. C. Jea. Generalized conjugate-gradient acceleration of nonsymmetrizable iterative methods. Linear Algebra and its Applications, 34:159–194, 1980.

[33] J. Y. Yuan, G. H. Golub, R. J. Plemmons, and W. Cecilio. Semi-conjugate direction methods for real positive definite systems. BIT Numer. Math., 44(1):189–207, 2004.