Numerical Algorithms (2020) 84:1265–1293
https://doi.org/10.1007/s11075-019-00814-7

ORIGINAL PAPER

The simpler block CMRH method for linear systems

Ilias Abdaoui[1] · **Lakhdar Elbouyahyaoui**[2] · **Mohammed Heyouni**[1]

Received: 9 June 2019 / Accepted: 20 September 2019 / Published online: 11 March 2020

Abstract

The block changing minimal residual method based on the Hessenberg reduction algorithm (in short BCMRH) is a recent block Krylov method that can solve large linear systems with multiple right-hand sides. This method uses the block Hessenberg process with pivoting strategy to construct a trapezoidal Krylov basis and minimizes a quasi-residual norm by solving a least squares problem. In this paper, we describe the simpler BCMRH method which is a new variant that avoids the QR factorization to solve the least-squares problem. Another major difference between the classical and simpler variants of BCMRH is that the simpler one allows to check the convergence within each cycle of the block Hessenberg process by using a recursive relation that updates the residual at each iteration. This is not possible with the classical BCMRH where we can only compute an estimate of the residual norm. Experiments are described to compare the behavior of the new proposed method with that of the classical and simpler versions of the block GMRES method. These numerical experiments show the good performances of the simpler BCMRH method.

Keywords Block Krylov subspace · Block Hessenberg process · Simpler block CMRH method · Simpler block GMRES method

Mathematics Subject Classification (2010) 65F

✉ Mohammed Heyouni
mohammed.heyouni@gmail.com

Ilias Abdaoui
ilias.abdaoui@yahoo.com

Lakhdar Elbouyahyaoui
lakhdarr2000@yahoo.fr

1 Laboratoire LM2N, Equipe MSN ENSA, Université Mohammed Premier, Oujda, Morocco

2 Centre Régional des Métiers de l'Education et de la Formation, Fes, Morocco

1 Introduction

Many scientific applications require efficient iterative methods for solving large nonsymmetric linear systems with multiple right-hand sides of the form

$$A\,X = B, \tag{1}$$

where $A \in \mathbb{R}^{n\times n}$ is a nonsingular matrix and $B,\ X \in \mathbb{R}^{n\times r}$ are rectangular matrices with r largely smaller than n, $(r \ll n)$. Obviously solving (1) is equivalent to solving the r linear systems $A\,x^{(i)} = b^{(i)}$, for $i = 1, \dots, r$, where $x^{(i)},\ b^{(i)}$ are the ith column of X and B, respectively. Problems of type (1) can be encountered in many areas of scientific computing and engineering, such as wave propagation phenomena, computational biology, quantum chromo-dynamics, electromagnetic structure computation, control theory, and dynamics of structures (see for example [11, 21, 25, 29] and the references therein).

Recently, there has been a great interest in developing block Krylov methods to solve systems of the form (1). Indeed, in many situations, these methods are more competitive—compared with others such as global methods or classical methods applied separately to each linear system $A\,x^{(i)} = b^{(i)}$—because block Krylov spaces enlarge the search space and therefore implicitly contain the Krylov subspaces associated with each system. In addition, block Krylov methods allow the use of level 3-BLAS operations in the implementation of these methods [25, 28]. The BCG algorithm described by O'leary is one of the first block methods. This algorithm is used when A is a symmetric positive definite matrix [24]. For non-symmetric matrices, other Lanczos-based methods have been developed to solve (1) such as block quasi-minimal residual (Bl-QMR) algorithm [11], block bi-conjugate gradient stabilized (Bl-BiCGstab) method [9], block Lanczos/Orthodir, and block BIODIR algorithms which are derived from the block Lanczos method [10]. Several other methods based on the Arnoldi process [25] were also proposed such as block FOM, block GMRES (BGMRES), and its variants [22, 25, 29, 34]. For a survey and detailed overview on block Krylov methods, we refer the reader to [16, 17, 25]. It should also be noted that, taking into account the well-known shift invariance property of (single or block) Krylov subspaces [6], many of the methods mentioned above have been adapted to address the resolution of shifted linear systems [30, 32, 33]. We also draw the reader's attention to the fact that currently, preconditioning combined with deflation/augmentation techniques are among the best tools to consider to improve the robustness and efficiency of block Krylov subspace methods. Recent developments on this context can be found in [23, 31, 37] and the references therein.

When solving single linear systems (case $r = 1$) and like the Arnoldi and Lanczos processes [36], the Hessenberg process is another tool that can generate a Krylov basis of the subspace $K_k(A, v) = \mathrm{span}\{v, A\,v, \dots, A^{k-1}\,v\}$, where $v \in \mathbb{R}^n$. In [26], Sadok used this process in order to define the CMRH (changing minimal residual based on the Hessenberg reduction algorithm) method for solving a single linear system. The derivation of this method is similar to that of the QMR method [12], since it uses a quasi-minimal residual condition. Moreover, authors in [27] show that the convergence behavior of the CMRH method is similar to that of the GMRES method. Recently, there has been a growing interest in the Hessenberg process. To solve large and dense linear systems, a new implementation of this process is proposed in [18] and a comparison with the Gaussian elimination and GMRES methods is given in [8]. In [14, 15], variants of the CMRH and Hessenberg methods has been successfully used for the solution of shifted linear systems. To solve matrix equations in general and multiple linear systems in particular, block versions of the Hessenberg process are described in [1, 2].

In this work, we are interested in methods based on the block Hessenberg process. More precisely, we consider the block CMRH (BCMRH) method first introduced in [2] and also described in [3]. This method uses the block Hessenberg process with pivoting strategy, to generate a lower trapezoidal basis for the block Krylov subspace. We recall that block CMRH shares much similarity with block GMRES, but with less arithmetic and storage requirements. Thus, inspired by the elegance of the simpler GMRES (sGMRES) proposed by Walker and Zhou and the performances of the simpler block GMRES (sBGMRES) [22, 35], we propose a new implementation of the BCMRH algorithm that will be called simpler block CMRH (sBCMRH). This new variant is faster and less expensive in terms of computational needs for many considerations. The most important one is that here the block Hessenberg process is reformulated to be applied to the pair (A, AR_0) instead of the pair (A, R_0)—where R_0 is the initial residual—and this leads to a triangular linear system easier to solve. Moreover, the simpler version of the block CMRH method allows to check the convergence within each cycle of the block Hessenberg process by using a recursive relation that updates the residual at each iteration. Similarly, this new implementation avoids the update of the QR factorization that is needed in the solution of the least-squares problem produced by the quasi-minimization condition. We note that an analysis of the numerical behavior of the simpler GMRES method has been described in [19]. This analysis suggested proposing adaptive versions for the simpler GMRES based methods [20, 37]. We think it would be interesting to apply this analysis to the simpler block CMRH method and hope to do it in a future work.

The content of this paper is organized as follows: in the Section 2, we recall the block CMRH method with a brief mention to the Hessenberg process and some of its properties. In Section 3, we describe a simpler variant of the block CMRH method. Then, we establish some theoretical results that link the simpler block CMRH algorithm to the simpler block GMRES algorithm. The last section will be dedicated to numerical examples that show the good behavior of the simpler block CMRH in comparison with the simpler block GMRES or with the classical block CMRH and GMRES methods.

2 The block CMRH method

Of interest in this section is the block changing minimal residual method based on the Hessenberg process (BCMRH) introduced by Addam et al. in [2]. This method can be viewed as a generalization to the block case of the CMRH method proposed by Sadok [26]. The principle underlying the BCMRH method for solving the block linear system (1) is to build a particular Krylov basis via the block Hessenberg process with pivoting strategy.

Before describing the block Hessenberg process, we introduce the notion of a left inverse for a rectangular matrix [26]. Let $k \leq n$ and Z_k, Y_k be two $n \times k$ matrices such that $Y_k^T Z_k$ is nonsingular. We define a left inverse Z_k^L of Z_k by

$$Z_k^L = \left(Y_k^T Z_k\right)^{-1} Y_k^T.$$

As we can see, this left inverse satisfies $Z_k^L Z_k = I_k$. We point out that for a full-column rank matrix Z_k there exists many left inverses—i.e. the left inverse of a matrix is not unique—and generally the favorite one is the so-called pseudo-inverse which is defined by $Z_k^+ = \left(Z_k^T Z_k\right)^{-1} Z_k^T$. But in the following, we will use another left inverse that is subordinate to the PLU decomposition of Z_k. More precisely, let P the $n \times n$ permutation matrix, L the $n \times k$ lower unit trapezoidal matrix, and U the $k \times k$ upper triangular be the factors in the PLU decomposition Z_k, i.e.,

$$P Z_k = L U.$$

Then, letting P_k be the $n \times k$ matrix whose columns are the k first columns of P, we can check that $P_k^T Z_k$ is nonsingular and so $Z_k^\ell := \left(P_k^T Z_k\right)^{-1} P_k^T$ is a left inverse of Z_k.

2.1 The block Hessenberg process

Here, we give a brief description of the block Hessenberg process. Let $V \in \mathbb{R}^{n\times r}$ be a given block column vector and

$$\mathbb{K}_m(A, V) = \text{span}\{V, A V, \ldots, A^{m-1} V\},$$

the block Krylov subspace associated to the pair (A, V) where the span of a sequence of block vectors is simply the span of the columns of all the blocks combined. This means that the block Krylov space $\mathbb{K}_m(A, V)$ is the subset of $\mathbb{R}^{n\times r}$ defined by

$$\mathbb{K}_m(A, V) = \left\{\sum_{k=0}^{m-1} A^k V \Omega_k, \quad \text{where } \Omega_k \in \mathbb{R}^{r\times r} \ \text{ for } k = 1, \ldots, m-1\right\}.$$

The block Hessenberg process with pivoting strategy consists in constructing recursively a sequence of block matrices $V_1, V_2, \ldots, V_k$ such that $V_i \in \mathbb{R}^{n\times r}$ and $\mathbb{V}_k = [V_1, \ldots, V_k] \in \mathbb{R}^{n\times kr}$ is a unit lower trapezoidal basis—up to a permutation matrix—of the block Krylov subspace $\mathbb{K}_m(A, V)$. This process proceeds as follows [2]:

– We compute the first block vector V_1 by

$$V_1 = P_1^T L_1 = V H_{1,0}^{-1},$$

where P_1, L_1, and $H_{1,0}$ are the coefficient matrices that appear in the following LU decomposition with partial pivoting of the block vector V

$$P_1\, V = L_1\, H_{1,0}, \quad \text{(PLU decomposition of } V\text{)}.$$

More precisely, P_1 is the $n \times n$ permutation matrix, L_1 is the $n \times r$ unit lower trapezoidal matrix, and $H_{1,0}$ is the $r \times r$ upper triangular matrix.

– Now, for $j = 1, \ldots, r$, let i_j be the index row of V_1 corresponding to the jth row of L_1 and define the vector $p_1 = [i_1, \ldots, i_r]$ and the $n \times r$ matrix $\widetilde{E}_1 = [e_{i_1}, \ldots, e_{i_r}]$ which corresponds to the r first columns of P_1 with $e_i = [0, \ldots, 0, 1, 0 \ldots, 0]^T$ is the ith vector of the canonical basis of $\mathbb{R}^n$.
– In order to show how to compute V_{k+1}, we suppose that the preceding block matrices $V_2, \ldots, V_k$ have been already computed and the permutation vectors $p_2, \ldots, p_k$ updated. Then, we first take $\widetilde{V}_{k+1}^{(0)} = A\, V_k$ and generate the following block vectors $\widetilde{V}_{k+1}^{(j)}$ via

$$\widetilde{V}_{k+1}^{(j)} = \widetilde{V}_{k+1}^{(j-1)} - V_j\, H_{j,k},$$

where for $j = 1, \ldots, k$ the coefficients $H_{j,k} \in \mathbb{R}^{r \times r}$ are computed by imposing the orthogonality condition

$$\widetilde{V}_{k+1}^{(j)} \perp \widetilde{E}_1, \ldots, \ \widetilde{E}_j.$$

Thus, thanks to the above condition and using Matlab notation, we can check that

$$H_{j,k} = (V_j(p_j, :))^{-1}\, \widetilde{V}_{k+1}^{(j)}(p_j, :), \quad \text{for } j = 1, \ldots, k.$$

Computing again, $P_{k+1}\, \widetilde{V}_{k+1}^{(k)} = L_{k+1}\, H_{k+1,k}$ the PLU decomposition of $\widetilde{V}_{k+1}^{(k)}$, we define V_{k+1} as

$$V_{k+1} = P_{k+1}^T\, L_{k+1} = \widetilde{V}_{k+1}^{(k)}\, H_{k+1,k}^{-1}.$$

– The derivation of the block Hessenberg process is achieved by considering i_j as the index row of V_{k+1} which corresponds to the jth row of L_{k+1} ($j = kr + 1, \ldots, (k+1)r$), $p_{k+1} = [i_{kr+1} \ldots, i_{(k+1)r}]$ and by taking $\widetilde{E}_{k+1} := E_{p_{k+1}} = [e_{i_{kr+1}}, \ldots, e_{i_{(k+1)r}}]$.

Finally, before summarizing the block Hessenberg process described by Algorithm 1, we note that the use of the **lu** and **max** Matlab functions help us in updating the blocks V_1, V_{k+1} and the vectors p_1, p_{k+1} by

$$\left[V_1, H_{1,0}\right] = \mathbf{lu}(V) \qquad \text{and } [\sim, p_1] = \mathbf{max}(V_1),$$
$$\left[V_{k+1}, H_{k+1,k}\right] = \mathbf{lu}\left(\widetilde{V}_{k+1}^{(k)}\right) \quad \text{and } \left[\sim, p_{k+1}\right] = \mathbf{max}(V_{k+1}).$$

Algorithm 1 The block Hessenberg algorithm (with partial pivoting).

Input: A an $n \times n$ matrix, V an $n \times r$ matrix and m an integer.

1: Compute the PLU decomposition of V and the first permutation vector, i.e., $[V_1,\ H_{1,0}] = \mathbf{lu}(V)$; $[\sim,\ p_1] = \mathbf{max}(V_1)$;
2: **for** $k = 1, \ldots, m$ **do**
3: $\quad \widetilde{V}_{k+1}^{(0)} = A\, V_k$;
4: $\quad$ **for** $j = 1, 2, \ldots, k$ **do**
5: $\quad\quad H_{j,k} = (V_j(p_j, :))^{-1}\, \widetilde{V}_{k+1}^{(j-1)}(p_j, :)$;
6: $\quad\quad \widetilde{V}_{k+1}^{(j)} = \widetilde{V}_{k+1}^{(j-1)} - V_j\, H_{j,k}$;
7: $\quad$ **end for**
8: $\quad$ Compute the PLU decomposition of $\widetilde{V}_{k+1}^{(k)}$ and the $(k+1)$-th permutation vector, i.e., $[V_{k+1}, H_{k+1,k}] = \mathbf{lu}(\widetilde{V}_{k+1}^{(k)})$; $[\sim, p_{k+1}] = \mathbf{max}(V_{k+1})$;
9: **end for**

At the end of the kth iteration, the following typical relations are obtained

$$\begin{aligned} A\, \mathbb{V}_k &= \mathbb{V}_{k+1}\, \widetilde{\mathbb{H}}_k && (2)\\ &= \mathbb{V}_k\, \mathbb{H}_k + V_{k+1}\, H_{k+1,k}\, E_k^T. && (3) \end{aligned}$$

where $E_k = [0_r, 0_r, \ldots, 0_r, I_r]^T$ is the kth block of the identity matrix I_{kr} and $\widetilde{\mathbb{H}}_k$, $\mathbb{H}_k$ are respectively the $(k+1)r \times kr$ and $kr \times kr$ block upper Hessenberg matrices whose non-zero block entries are the $H_{j,k}$ generated by Algorithm 1. Note also that if $\mathbb{P}_k$ is the $n \times kr$ permutation matrix whose block columns are $\widetilde{E}_1, \ldots, \widetilde{E}_k$, i.e.,

$$\mathbb{P}_k = [\widetilde{E}_1, \widetilde{E}_2, \ldots, \widetilde{E}_k],$$

then

$$\mathbb{P}_k^T\, \mathbb{V}_k = \mathbb{L}_k, \tag{4}$$

where $\mathbb{L}_k \in \mathbb{R}^{kr \times kr}$ is a unit lower triangular matrix. Using (4) and the fact that $\mathbb{L}_k$ being nonsingular we let $\mathbb{V}_k^\ell = \mathbb{L}_k^{-1}\, \mathbb{P}_k^T$ be the left inverse of $\mathbb{V}_k$ that is subordinate to the PLU decomposition of $\mathbb{V}_k$. Since $\mathbb{L}_k$ is a lower trapezoidal matrix, then we easily verify that $\mathbb{V}_{k+1}^\ell$ can be partitioned under the form

$$\mathbb{V}_{k+1}^\ell = \begin{bmatrix} \mathbb{V}_k^\ell \\ V_{k+1}^\ell \end{bmatrix}. \tag{5}$$

Now, pre-multiplying (2) and (3) respectively by $\mathbb{V}_{k+1}^{\ell}$ and $\mathbb{V}_{k}^{\ell}$, we obtain

$$\mathbb{V}_{k+1}^{\ell}\, A\, \mathbb{V}_k = \widetilde{\mathbb{H}}_k \quad \text{and} \quad \mathbb{V}_{k}^{\ell}\, A\, \mathbb{V}_k = \mathbb{H}_k.$$

2.2 Description of the block CMRH method

In this subsection, we give a description of the BCMRH method when applied to the block linear system (1). Let X_0 be an initial guess and $R_0 = B - A\,X_0$ the corresponding residual. The block CMRH method builds a sequence of iterates X_k such that

$$X_k - X_0 = \Psi_k \in \mathbb{K}_k(A, R_0), \tag{6}$$

where Ψ_k results from the following constrained minimization problem

$$\Psi_k = \underset{\substack{\mathscr{W}\in\mathbb{R}^{(k+1)r\times r}\\ Z\in\mathbb{K}_k(A,R_0)}}{\operatorname{argmin}} \|\mathscr{W}\|_F, \quad \text{subject to } A\,Z = R_0 - \mathbb{V}_{k+1}\,\mathscr{W}, \tag{7}$$

and $\mathbb{V}_{k+1}$ is the permuted lower trapezoidal basis obtained by Algorithm 1 applied to the pair (A, R_0). Using the left inverse $\mathbb{V}_{k+1}^{\ell}$ of the matrix $\mathbb{V}_{k+1}$, we reformulate the minimization problem (7) as follows:

$$\Psi_k = \underset{Z\in\mathbb{K}_k(A,R_0)}{\operatorname{argmin}} \|\mathbb{V}_{k+1}^{\ell}\,(R_0 - A\,Z)\|_F. \tag{8}$$

Now, writing $\Psi_k = \mathbb{V}_k\,Y_k$ where $Y_k \in \mathbb{R}^{kr\times r}$ and since $\mathbb{V}_{k+1}^{\ell}\,(R_0 - A\,\mathbb{V}_k Y) = E_1\,H_{1,0} - \widetilde{\mathbb{H}}_k\,Y$, with $H_{1,0}$ obtained from the PLU decomposition of R_0 and $E_1 \in \mathbb{R}^{(k+1)r\times r}$ corresponds to the first r columns of the identity matrix $I_{(k+1)\,r}$, then we compute the BCMRH approximate solution by

$$X_k = X_0 + \mathbb{V}_k\,Y_k,$$

such that Y_k solves the least squares problem

$$\min_{Y\in\mathbb{R}^{kr\times r}} \|E_1\,H_{1,0} - \widetilde{\mathbb{H}}_k\,Y\|_F. \tag{9}$$

As the corresponding residual $R_k = B - A\,X_k$ is given by

$$R_k = R_0 - A\,\mathbb{V}_k\,Y_k = V_1\,H_{1,0} - \mathbb{V}_{k+1}\,\widetilde{\mathbb{H}}_k = \mathbb{V}_{k+1}\,(E_1\,H_{1,0} - \widetilde{\mathbb{H}}_k\,Y_k),$$

we easily see that the update of the block CMRH iterates X_k are based on condition (6) and on requiring that the residual vector $R_k = B - A\,X_k$ satisfies the Galerkin-type condition

$$R_k \perp_\ell A\,\mathbb{K}_k(A, R_0), \tag{10}$$

which means that $\mathscr{B}_k^\ell\, R_k = 0$ provided that $\mathscr{B}_k$ is a suitable basis of $A\,\mathbb{K}_k(A, R_0)$.

Remark 1 Let $\mathbb{W}_k = (\mathbb{V}_{k+1}^\ell)^T\, \mathbb{V}_{k+1}^\ell$ and define the semi-norm $|U|_{\mathbb{W}_k} = \sqrt{tr(U^T\, \mathbb{W}_k\, U)}$. Then, the constrained minimization problem (7) can be interpreted as a standard residual minimization using this semi-norm since

$$\rho_k := |R_k|_{\mathbb{W}_k} = \min_{Z \in \mathbb{K}_k(A, R_0)} |R_0 - A\,Z|_{\mathbb{W}_k} = \min_{Y \in \mathbb{R}^{kr \times r}} \|E_1\, H_{1,0} - \widetilde{\mathbb{H}}_k\, Y\|_F.$$

In a similar way to solving the least-squares problems appearing in the classical CMRH, GMRES, or block GMRES methods, the solution of the minimization problem (9) can be obtained recursively for each index k by means of Givens rotations or Householder transformations in order to obtain a QR factorization of $\widetilde{\mathbb{H}}_k$. Note that the standard value of the residual ρ_k ($\rho_k = \|E_1\, H_{1,0} - \widetilde{\mathbb{H}}_k\, Y_k\|_F$) can be calculated even before the solution Y_k is determined. It can then be verified that the value ρ_k is equal to the minimum residual in (8). As with the standard block GMRES and in order to limit the increasing memory and algorithmic costs, one has to use a restarting strategy in the implementation of the BCMRH algorithm (for more details on the restarting strategy, see [25]). Next, we summarize the BCMRH in Algorithm 2.

Algorithm 2 The restarted block CMRH method: BCMRH(m) (simplified version).

Input: A an $n \times n$ matrix, B an $n \times r$ matrix, X_0 an $n \times r$ matrix (initial guess), m an integer and ϵ a desired tolerance.

1: Compute $R_0 = B - AX_0$;
2: Apply Algorithm 1 to the pair (A, R_0) to get $H_{1,0}$, $\mathbb{V}_m$ and $\widetilde{\mathbb{H}}_m$;
3: Determine Y_m as the solution of $\min_{Y \in \mathbb{R}^{m\,r \times r}} \|E_1\, H_{1,0} - \widetilde{\mathbb{H}}_m\, Y\|_F$;
4: Compute the approximate solution $X_m = X_0 + \mathbb{V}_m\, Y_m$;
5: Compute $R_m = B - A\,X_m$;
6: **if** $\|R_m\|_F \leq \epsilon$ **then**
7:     Stop;
8: **else**
9:     $X_0 = X_m$; $R_0 = R_m$; goto line 2;
10: **end if**

We end this section by pointing out that Algorithm 2 describes a simplified implementation of the BCMRH method. This version does not allow to detect the convergence of the BCMRH method when it can take place at an iteration k with $k < m$, i.e., during a restart cycle. Generally, the cost of the additional iterations is accepted especially when the size mr of the block upper Hessenberg matrix $\widetilde{\mathbb{H}}_{m,r}$ is small. On the other hand, if the size of $\mathbb{H}_m$ is relatively large, it is preferable to use Givens rotations when updating the QR factorization of $\widetilde{\mathbb{H}}_m$ and thus be able to detect convergence during a cycle of iterations. Algorithm 3 resuming the BCMRH method and described below takes this remark into account.

Algorithm 3 The restarted block CMRH method: BCMRH(m).

Input: A an $n \times n$ matrix, B an $n \times r$ matrix, X_0 an $n \times r$ matrix (initial guess), m an integer and ϵ a desired tolerance.

1: Compute $R_0 = B - AX_0$; Compute the PLU decomposition of R_0 and the first permutation vector, i.e., $[V_1,\ H_{1,0}] = \mathbf{lu}(R_0)$; $[\sim,\ p_1] = \mathbf{max}(V_1)$;
2: **for** $k = 1, \ldots, m$ **do**
3: $\quad \widetilde{V}_{k+1}^{(0)} = A\, V_k$;
4: $\quad$ **for** $j = 1, 2, \ldots, k$ **do**
5: $\quad\quad H_{j,k} = (V_j(p_j, :))^{-1}\, \widetilde{V}_{k+1}^{(j-1)}(p_j, :)$;
6: $\quad\quad \widetilde{V}_{k+1}^{(j)} = \widetilde{V}_{k+1}^{(j-1)} - V_j\, H_{j,k}$;
7: $\quad$ **end for**
8: $\quad$ Compute the PLU decomposition of $\widetilde{V}_{k+1}^{(k)}$ and the $(k+1)$-th permutation vector, i.e., $[V_{k+1}, H_{k+1,k}] = \mathbf{lu}(\widetilde{V}_{k+1}^{(k)})$; $[\sim, p_{k+1}] = \mathbf{max}(V_{k+1})$;
9: $\quad$ Update the QR factorization of $\widetilde{\mathbb{H}}_k$ and determine ρ_k an estimate of the norm of the residual R_k;
10: $\quad$ **if** $\rho_k \leq \epsilon$ **then**
11: $\quad\quad$ replace m by k, i.e. $m = k$;
12: $\quad\quad$ go to line (15); (there is no need to determine the solution Y_k yet)
13: $\quad$ **end if**
14: **end for**
15: Determine $Y_m = \underset{Y \in \mathbb{R}^{mr \times r}}{\operatorname{argmin}} \|E_1\, H_{1,0} - \widetilde{\mathbb{H}}_m\, Y\|_F$ by solving the triangular linear system obtained after the QR factorization of $\widetilde{\mathbb{H}}_m$;
16: Compute the approximate solution $X_m = X_0 + \mathbb{V}_m\, Y_m$;
17: **if** $\rho_m \leq \epsilon$ **then**
18: $\quad$ accept X_m and exit;
19: **else**
20: $\quad X_0 = X_m$ and goto line 1;
21: **end if**

3 The simpler BCMRH method

It is clear that the essential ingredient in the implementation of the BCMRH method is the application of m steps of the block Hessenberg process applied to the pair

(A, R_0). This yields a basis $\mathbb{V}_m$ of the block Krylov subspace $\mathbb{K}_m(A, R_0)$ and an upper block Hessenberg matrix $\mathbb{H}_m$ satisfying (2). Then, we have to solve the reduced minimization problem (9), which can be done by updating a recursive QR factorization via Givens rotations or Householder transformations [25]. Following the ideas developed by Liu and Zhong in [22] and in order to avoid the QR decomposition of the block upper Hessenberg matrix $\widetilde{\mathbb{H}}_m$, we describe in this section a simpler implementation of the block CMRH method.

3.1 The simpler block Hessenberg process

Based on the Galerkin-type condition (10) and instead of applying the block Hessenberg process to the pair (A, R_0), we will shift this process to begin with $A\,R_0$ to construct a trapezoidal basis $\mathbb{Q}_m$ of $A\,\mathbb{K}_m(A, R_0)$ and an upper triangular matrix $\mathbb{T}_m$. These modifications on the block Hessenberg process are summarized in the following algorithm.

Algorithm 4 The simpler block Hessenberg process (with partial pivoting).

Input: A an $n \times n$ matrix, V an $n \times r$ matrix and m an integer.

1: Compute the PLU decomposition of $A\,V$ and the first permutation vector, i.e., $[Q_1,\ T_{1,1}] = \mathbf{lu}(A\,V)$; $[\sim,\ p_1] = \mathbf{max}(Q_1)$;

2: **for** $k = 2, \ldots, m$ **do**

3: $\quad \widetilde{Q}_{k+1}^{(0)} = A\,Q_{k-1}$;

4: $\quad$ **for** $j = 1, 2, \ldots, k-1$ **do**

5: $\quad\quad T_{j,k} = (Q_j(p_j, :))^{-1}\, \widetilde{Q}_k^{(j-1)}(p_j, :)$;

6: $\quad\quad \widetilde{Q}_k^{(j)} = \widetilde{Q}_k^{(j-1)} - Q_j\, T_{j,k}$;

7: $\quad$ **end for**

8: $\quad$ Compute the PLU decomposition of $\widetilde{Q}_k^{(0)}$ and the k-th permutation vector, i.e., $[Q_k, T_{k,k}] = \mathbf{lu}(\widetilde{Q}_k^{(0)})$; $[\sim, p_k] = \mathbf{max}(Q_k)$;

9: **end for**

Denotied by $\mathbb{Q}_k = [Q_1, Q_2, \ldots, Q_k]$, the trapezoidal basis of $A\,\mathbb{K}_k(A, R_0)$ obtained after k iterations of Algorithm 4 and letting $\mathbb{Z}_k = [R_0, \mathbb{Q}_{k-1}]$—which is a basis of $\mathbb{K}_k(A, R_0)$—we can check that the following relation is satisfied

$$A\,\mathbb{Z}_k = \mathbb{Q}_k\,\mathbb{T}_k, \tag{11}$$

where $\mathbb{T}_k = (T_{i,j}) \in \mathbb{R}^{kr \times kr}$ is the upper triangular matrix whose non-zero entries are the $T_{i,j}$ computed in lines 1, 5, and 8 of Algorithm 4. We point out that similar to Algorithm 1, we let $\overline{\mathbb{P}}_k$ be the permutation matrix whose block columns are $\widetilde{E}_1, \ldots, \widetilde{E}_k$ where $\widetilde{E}_j = E_{p_j}$ $(j = 1, \ldots, k)$ with $p_1, \ldots, p_k$ computed in lines 1 and 8 of Algorithm 4. In this case, we verify that $\overline{\mathbb{P}}_k^T\,\mathbb{Q}_k = \overline{\mathbb{L}}_k$ where $\overline{\mathbb{L}}_k$ is an unit lower triangular matrix and then we take $\mathbb{Q}_k^\ell = \overline{\mathbb{L}}_k^{-1}\,\overline{\mathbb{P}}_k^T$ as a left inverse of $\mathbb{Q}_k$.

3.2 Derivation of the simpler block CMRH algorithm

We seek for X_k, an approximate solution to (1), belonging to $X_0 + \mathbb{K}_k(A, R_0)$ and such that the corresponding residual $R_k = B - A\,X_k$ satisfies the Galerkin-type condition (10). Thus, there exists $\mathbb{S}_k = [S_1^T, S_2^T, .., S_k^T]^T \in \mathbb{R}^{kr\times r}$ with $S_i \in \mathbb{R}^{r\times r}$ for $i = 1, \dots, k$ and such that the residual R_k is under the form

$$R_k = R_0 - \mathbb{Q}_k\,\mathbb{S}_k.$$

Pre-multiplying the above equation by $\mathbb{Q}_k^\ell$, we get $\mathbb{S}_k = \mathbb{Q}_k^\ell\,R_0$. So, the kth residual is given by

$$R_k = R_0 - \mathbb{Q}_k\,\mathbb{Q}_k^\ell\,R_0 = (I - \mathbb{Q}_k\,\mathbb{Q}_k^\ell)\,R_0. \tag{12}$$

Consequently, as $\mathbb{Q}_k = [\mathbb{Q}_{k-1},\;Q_k]$ and thanks to property (5), the residual R_k can be obtained recursively via

$$R_k = R_{k-1} - Q_k\,S_k, \quad \text{where } S_k = Q_k^\ell\,R_{k-1} = Q_k^\ell\,R_0. \tag{13}$$

Now, since the columns of $\mathbb{Z}_k = [R_0, \mathbb{Q}_{k-1}]$ form a basis of $\mathbb{K}_k(A, R_0)$, the approximate solution is given by

$$X_k = X_0 + [R_0, \mathbb{Q}_{k-1}]\,Y_k, \tag{14}$$

where Y_k is the solution of the block upper triangular system

$$\mathbb{T}_k\,Y_k = \mathbb{S}_k. \tag{15}$$

We point out that the previous upper triangular system is only solved when the Frobenius residual norm $\|R_k\|_F$ is small enough. As a consequence, the recursion formula (13) leads to evaluate the residual of the iterate X_k without computing it at every iteration. Thus, it avoids us to perform the matrix-vector product $A\,X_k$ if we compute the true residual $R_k = B - A\,X_k$ and this means that some floating operations are saved. The previous approach which describes a simpler implementation of the block CMRH method is summarized in the following algorithm.

Algorithm 5 The restarted simpler block CMRH method: sBCMRH(m).

Input: A an $n \times n$ matrix, B an $n \times r$ matrix, X_0 an $n \times r$ matrix (initial guess), m an integer and ϵ a desired tolerance.

1: Compute $R_0 = B - A X_0$; $Z_1 = R_0$; Compute the PLU decomposition of $A R_0$ and the first permutation vector, i.e., $[Q_1,\ T_{1,1}] = \mathbf{lu}(A R_0)$; $[\sim,\ p_1] = \mathbf{max}(Q_1)$;
2: Compute $S_1 = (Q_1(p_1,:))^{-1} R_0(p_1,:)$; $R_1 = R_0 - Q_1 S_1$;
3: **if** $\|R_1\|_F \leq \epsilon$ **then**
4: solve $\mathbb{T}_1 Y_1 = \mathbb{S}_1$; compute $X_1 = X_0 + R_0 Y_1$;
5: exit;
6: **end if**
7: **for** $k = 2, \ldots, m$ **do**
8: $\widetilde{Q}_k^{(0)} = A Q_{k-1}$;
9: **for** $j = 1, 2, \ldots, k-1$ **do**
10: $T_{j,k} = (Q_j(p_j,:))^{-1} \widetilde{Q}_k^{(j-1)}(p_j,:)$;
11: $\widetilde{Q}_k^{(j)} = \widetilde{Q}_k^{(j-1)} - Q_j T_{j,k}$;
12: **end for**
13: Compute the PLU decomposition of $\widetilde{Q}_k^{(k-1)}$ and the k-th permutation vector, i.e., $[Q_k, T_{k,k}] = \mathbf{lu}(\widetilde{Q}_k^{(k-1)})$; $[\sim, p_k] = \mathbf{max}(Q_k)$;
14: Compute $S_k = (Q_k(p_k,:))^{-1} R_k(p_k,:)$; $R_k = R_{k-1} - Q_k S_k$;
15: **if** $\|R_k\|_F \leq \epsilon$ **then**
16: solve $\mathbb{T}_k Y_k = \mathbb{S}_k$; compute $X_k = X_0 + \mathbb{Z}_k Y_k$;
17: exit;
18: **end if**
19: **end for**
20: solve $\mathbb{T}_m Y_m = \mathbb{S}_m$; compute $X_m = X_0 + \mathbb{Z}_m Y_m$;
21: $X_0 = X_m$;
22: go to line 1;

Before ending this section, we give the operation count for the sBCMRH method and compare it with those of the BCMRH [2, 3], sBGMRES [22], and BGMRES [34]. We also point out that the algorithms of the sBGMRES(m) and BGMRES(m) methods are given in the appendix.

In Table 1, we report the algorithmic costs of the different operations occurring during a given cycle of the sBCMRH(m) and sBGMRES(m) methods. Similarly, the algorithmic costs of the classical BCMRH and BGMRES methods are listed in Table 2. Obviously, as in [13], the costs listed are proportional to both n the size of the coefficient matrix A and r the number of columns of the right-hand side B. The difference in costs between the sBCMRH and sBGMRES methods is mainly explained by the fact that—a part from a permutation matrix—the basis $\mathbb{Q}_m$ constructed by the simpler Hessenberg process is unit lower trapezoidal. This means that the operations appearing in lines 2, 8, 10, 11, and 14 of Algorithm 5 are less expensive than their counterparts in Algorithm 7. Note that the same analysis holds when comparing the costs of the BCMRH and BGMRES methods.

Table 1 Computational cost during a cycle of the simpler BCMRH and BGMRES methods

Step	sBCMRH	sBGMRES
$A\,R_0$	$2\,n^2\,r$	$2\,n^2\,r$
LU factorization of $A\,R_0$	$n\,r^2 - \frac{r^3}{3}$	–
QR factorization of $A\,R_0$	–	$2\,n\,r^2 + n\,r$
$\widetilde{Q}_k^{(0)} = A\,Q_{k-1}$	$2\,n^2\,r - 2\,n\,k\,r^2 + \frac{r\,(r+1)}{2}$	$2\,n^2\,r$
$T_{j,k}$ for $j = 1, \ldots, k-1$	$(k-1)\,r^3$	$2\,n\,(k-1)\,r^2$
$\widetilde{Q}_k^{(j)}$ for $j = 1, \ldots, k-1$	$2\,n\,(k-1)\,r^2 + n\,r - (k-1)^2\,r^3 + (k-1)\,r^2$	$2\,n\,(k-1)\,r^2 + n\,r$
LU factorization of $\widetilde{Q}_k^{(k-1)}$	$n\,r^2 - k\,r^3 - \frac{r^3}{3}$	–
QR factorization of $\widetilde{Q}_k^{(k-1)}$	–	$2\,n\,r^2 + n\,r$
S_k	r^3	$2\,n\,r^2$
$R_k = R_{k-1} - Q_k\,S_k$	$2\,n\,r^2 + n\,r - k\,r^3$	$2\,n\,r^2 + n\,r$
Y_m	$m^2\,r^3$	$m^2\,r^3$

According to Tables 1 and 2, we find that the leading terms in the total number of operations that are necessary to execute one cycle of length **m** *of the simpler or classical block CMRH and GMRES methods* grows as $\alpha_1\,n^2\,r + \alpha_2\,n\,r^2 + \alpha_3\,n\,r$ where α_1, α_2 and α_3 are given in Table 3 *for each of the four methods*.

Finally, we conclude this section by a comparison between the leading terms that shows that the simpler block CMRH method is slightly less expensive than three other methods.

4 Relations between the simpler BCMRH and BGMRES methods

Following the ideas developed in [27], we derive in this section some theoretical results that links the simpler variants of the block GMRES and block CMRH algorithms.

Table 2 Computational cost during a cycle of the classical BCMRH and BGMRES methods

Step	BCMRH	BGMRES
LU factorization of R_0	$n\,r^2 - \frac{r^3}{3}$	–
QR factorization of R_0	–	$2\,n\,r^2 + n\,r$
$\widetilde{V}_{k+1}^{(0)} = A\,V_k$	$2\,n^2\,r - 2\,n\,k\,r^2 + \frac{r\,(r+1)}{2}$	$2\,n^2\,r$
$H_{j,k}$ for $j = 1, \ldots, k$	$k\,r^3$	$2\,n\,k\,r^2$
$\widetilde{V}_{k+1}^{(j)}$ for $j = 1, \ldots, k$	$2\,n\,k\,r^2 + n\,r - k^2\,r^3 + k\,r^2$	$2\,n\,k\,r^2 + n\,r$
LU factorization of $\widetilde{V}_{k+1}^{(k)}$	$n\,r^2 - k\,r^3 - \frac{r^3}{3}$	–
QR factorization of $\widetilde{V}_{k+1}^{(k)}$	–	$2\,n\,r^2 + n\,r$
Y_m	$(2\,m^3 + 3\,m^2)\,r^3$	$(2\,m^3 + 3\,m^2)\,r^3$

Table 3 Comparison of the leading terms appearing in the operating cost of the sBCMRH, sBGMRES, BCMRH, and BGMRES

Process \ α_i	α_1	α_2	α_3
sBCMRH	$2\,m$	$-(m-2)$	$m-1$
sBGMRES	$2\,m$	$2\,m^2+2\,m$	$2\,m-1$
BCMRH	$2\,m$	$2\,m+1$	$2\,m$
BGMRES	$2\,m$	$2\,(m+1)^2$	$\frac{m\,(m+3)}{2}$

Thus and in order to distinguish the different iterates produced by the simpler block CMRH and GMRES algorithms, we will add the superscripts $^{\mathrm{sC}}$ and $^{\mathrm{sG}}$. We also suppose that the initial guesses in the simpler block CMRH and simpler block GMRES methods are equal, i.e., $X_0^{\mathrm{sC}} = X_0^{\mathrm{sG}} = X_0$.

Before establishing some relations between the simpler block CMRH and GMRES methods, we need to describe briefly the simpler block GMRES method which is a cheaper implementation of the well-known block GMRES algorithm [22]. Indeed, let $\mathbb{V}_k$ be an orthonormal basis of the $A\,\mathbb{K}_k(A, R_0)$ which satisfies $A\,\mathbb{Z}_k^{\mathrm{sG}} = \mathbb{V}_k\,\mathbb{T}_k^{\mathrm{sG}}$, where $\mathbb{Z}_k^{\mathrm{sG}} = [R_0, \mathbb{V}_{k-1}]$ is a basis of $\mathbb{K}_k(A, R_0)$ and $\mathbb{T}_k^{\mathrm{sG}}$ is an upper triangular matrix [22, 37]. The approximate solution X_k^{sG} produced by the simpler block GMRES method is given by

$$X_k^{\mathrm{sG}} = X_0 + \mathbb{Z}_k^{\mathrm{sG}}\,Y_k^{\mathrm{sG}},$$

such that the following orthogonality condition is satisfied

$$R_k^{\mathrm{sG}} \perp_F A\,\mathbb{K}_k(A, R_0). \tag{16}$$

This allows to obtain Y_k^{sG} as the solution of the upper triangular system

$$\mathbb{T}_k^{\mathrm{sG}}\,Y_k^{\mathrm{sG}} = \mathbb{V}_k^T\,R_0.$$

Moreover, the residuals can be computed recursively via $R_k^{\mathrm{sG}} = R_{k-1}^{\mathrm{sG}} - V_k\,S_k^{\mathrm{sG}}$ with $S_k^{\mathrm{sG}} = V_k^T\,R_0$.

Now, let $\overline{\mathbb{K}}_k = [R_0, A\,R_0, \ldots, A^{k-1}\,R_0]$ be the block Krylov matrix and consider the following QR decomposition of the $n \times kr$ matrix $A\,\overline{\mathbb{K}}_k$

$$A\,\overline{\mathbb{K}}_k = \mathbb{V}_k\,\widetilde{\mathscr{R}}_k,$$

where $\mathbb{V}_k$ is an orthonormal basis of $A\,\mathbb{K}_k(A, R_0)$ and $\widetilde{\mathscr{R}}_k$ is an upper triangular matrix. As $\overline{\mathbb{K}}_{k+1} = [R_0, A\,\overline{\mathbb{K}}_k]$, we can write

$$\overline{\mathbb{K}}_{k+1} = [R_0, \mathbb{V}_k\widetilde{\mathscr{R}}_k] = [R_0, \mathbb{V}_k] \begin{bmatrix} I_k & 0_{k\times kr} \\ 0_{kr\times k} & \widetilde{\mathscr{R}}_k \end{bmatrix}.$$

Multiplying on the left by A, we get

$$A\,\overline{\mathbb{K}}_{k+1} = \mathbb{V}_{k+1}\,\widetilde{\mathscr{R}}_{k+1} = [A\,R_0, A\,\mathbb{V}_k] \begin{bmatrix} I_k & 0_{k\times kr} \\ 0_{kr\times k} & \widetilde{\mathscr{R}}_k \end{bmatrix}.$$

Now, from the simpler block GMRES we have

$$A\,[R_0, \mathbb{V}_k] = \mathbb{V}_{k+1}\,\mathbb{T}_{k+1}^{\mathrm{sG}},$$

where $\mathbb{T}^{\mathrm{sG}}_{k+1} \in \mathbb{R}^{kr\times kr}$ is the upper triangular matrix generated within the simpler block Arnoldi process and so

$$\mathbb{V}_{k+1}\,\mathbb{T}^{\mathrm{sG}}_{k+1}\begin{bmatrix} I_k & 0_{k\times kr} \\ 0_{kr\times k} & \widetilde{\mathscr{R}}_k \end{bmatrix} = \mathbb{V}_{k+1}\,\widetilde{\mathscr{R}}_{k+1}.$$

Next, multiplying on the left by $\mathbb{V}^T_{k+1}$, we obtain

$$\mathbb{T}^{\mathrm{sG}}_{k+1} = \widetilde{\mathscr{R}}_{k+1}\begin{bmatrix} I_k & 0_{k\times kr} \\ 0_{kr\times k} & \widetilde{\mathscr{R}}_k^{-1} \end{bmatrix}. \tag{17}$$

Analogously, using the simpler implementation of the block CMRH algorithm and considering the LU decomposition of the $n \times kr$ Krylov matrix $A\,\overline{\mathbb{K}}_k$, we have

$$A\,\overline{\mathbb{K}}_k = \mathbb{Q}_k\,\widetilde{\mathscr{U}}_k,$$

where $\mathbb{Q}_k$ is the lower trapezoidal basis of $A\,\overline{\mathbb{K}}_k(A, R_0)$ and $\widetilde{\mathscr{U}}_k$ is the upper triangular matrix generated by the simpler block Hessenberg process. Then

$$\overline{\mathbb{K}}_{k+1} = [R_0, A\,\overline{\mathbb{K}}_k] = [R_0, \mathbb{Q}_k\,\widetilde{\mathscr{U}}_k] = [R_0, \mathbb{Q}_k]\begin{bmatrix} I_k & 0_{k\times kr} \\ 0_{kr\times k} & \widetilde{\mathscr{U}}_k \end{bmatrix}.$$

Or from (11), we have $A\,\mathbb{Z}_{k+1} = A\,[R_0, \mathbb{Q}_k] = \mathbb{Q}_{k+1}\,\mathbb{T}^{\mathrm{sC}}_{k+1}$. Hence, by considering the LU decomposition of $A\,\overline{\mathbb{K}}_{k+1}$ we obtain

$$\mathbb{Q}_{k+1}\,\mathbb{T}^{\mathrm{sC}}_{k+1}\begin{bmatrix} I_k & 0_{k\times kr} \\ 0_{kr\times k} & \widetilde{\mathscr{U}}_k \end{bmatrix} = \mathbb{Q}_{k+1}\,\widetilde{\mathscr{U}}_{k+1}.$$

Multiplying on the left by $\mathbb{Q}^{\ell}_{k+1}$, we get

$$\mathbb{T}^{\mathrm{sC}}_{k+1} = \widetilde{\mathscr{U}}_{k+1}\begin{bmatrix} I_k & 0_{k\times kr} \\ 0_{kr\times k} & \widetilde{\mathscr{U}}_k^{-1} \end{bmatrix}. \tag{18}$$

Now, from the QR and LU decompositions of $A\,\overline{\mathbb{K}}_k$, we have

$$\mathbb{Q}_k\,\widetilde{\mathscr{U}}_k = \mathbb{V}_k\,\widetilde{\mathscr{R}}_k.$$

This implies that

$$\mathbb{Q}_k = \mathbb{V}_k\widetilde{\mathscr{R}}_k\,\widetilde{\mathscr{U}}_k^{-1} = \mathbb{V}_k\,\mathscr{R}_k, \tag{19}$$

which is the QR factorization of $\mathbb{Q}_k$. The previous decompositions enable us to give a relation between the triangular matrices $\mathbb{T}^{\mathrm{sG}}_{k+1}$ and $\mathbb{T}^{\mathrm{sC}}_{k+1}$. Indeed, observing that

$$\mathbb{V}_{k+1}\,\widetilde{\mathscr{R}}_{k+1} = \mathbb{Q}_{k+1}\,\widetilde{\mathscr{U}}_{k+1},$$

then using (19), (18), and (17), we get

$$\mathbb{T}^{\mathrm{sG}}_{k+1}\begin{bmatrix} I_k & 0_{k\times kr} \\ 0_{kr\times k} & \widetilde{\mathscr{R}}_k \end{bmatrix} = \mathscr{R}_{k+1}\,\mathbb{T}^{\mathrm{sC}}_{k+1}\begin{bmatrix} I_k & 0_{k\times kr} \\ 0_{kr\times k} & \widetilde{\mathscr{U}}_k \end{bmatrix}.$$

Finally, the upper triangular matrices $\mathbb{T}^{\mathrm{sG}}_{k+1}$, $\mathbb{T}^{\mathrm{sC}}_{k+1}$ generated with the simpler Arnoldi and Hessenberg processes respectively are such that

$$\mathbb{T}^{\mathrm{sG}}_{k+1}\begin{bmatrix} I_k & 0_{k\times kr} \\ 0_{kr\times k} & \mathscr{R}_k \end{bmatrix} = \mathscr{R}_{k+1}\mathbb{T}^{\mathrm{sC}}_{k+1},$$

or equivalently

$$\mathbb{T}^{\mathrm{sC}}_{k+1} = \mathscr{R}^{-1}_{k+1}\, \mathbb{T}^{\mathrm{sG}}_{k+1} \begin{bmatrix} I_k & 0_{k\times kr} \\ 0_{kr\times k} & \mathscr{R}_k \end{bmatrix}. \tag{20}$$

Proposition 1 *Suppose that the initial guesses in the simpler block CMRH and GMRES methods are equal, i.e.,* $X_0^{sC} = X_0^{sG} = X_0$, *then the following relations hold*

1. *The pseudo-inverse of the matrix* $\mathbb{Q}_k$ *satisfies* $\mathbb{Q}_k^{\dagger} = \mathscr{R}_k^{-1}\mathbb{V}_k^T$ *and so* $\mathscr{R}_k^{-1} = \mathbb{Q}_k^{\dagger}\mathbb{V}_k$.
2. *The kth residual* R_k^{sG} *of the simpler block GMRES method is given by* $R_k^{sG} = (I_n - \mathbb{V}_k\,\mathbb{V}_k^T)\,R_0$.
3. *The kth iterate* X_k^{sC} *of the simpler block CMRH method and its corresponding residual* R_k^{sC} *are given by*

   (i) $X_k^{sC} = X_0 + [R_0, \mathbb{V}_{k-1}]\,(\mathbb{T}_k^{sG})^{-1}\,\mathscr{R}_k\,\mathbb{Q}^{\ell}\,R_0 = X_0 + [R_0, \mathbb{V}_{k-1}]\,(\mathbb{T}_k^{sG})^{-1}\,\mathscr{R}_k\,\overline{\mathbb{L}}_k^{-1}\,\mathbb{P}_k^T\,R_0$,

   (ii) $R_k^{sC} = (I_n - \mathbb{V}_k\,\mathscr{R}_k\,\overline{\mathbb{L}}_k^{-1}\,\mathbb{P}_k^T)\,R_0$.

4. *The kth iterates* X_k^{sC} *and* X_k^{sG} *of the simpler block CMRH and GMRES methods and their corresponding residuals* R_k^{sC}, R_k^{sG} *satisfy*

   (i) $\frac{\|X_k^{sC}-X_k^{sG}\|_F}{\|R_0\|_F} \le \|[R_0, \mathbb{V}_{k-1}]\|_F\,\|(\mathbb{T}_k^{sG})^{-1}\|_F\,\|\mathscr{R}_k\|_F\,\|(\mathbb{Q}_k^{\ell} - \mathbb{Q}_k^{\dagger})\|_F$.

   (ii) $\frac{\|R_k^{sC}-R_k^{sG}\|_F}{\|R_0\|_F} \le \|\mathscr{R}_k\|_F\,\|\mathbb{Q}_k^{\dagger} - \mathbb{Q}_k^{\ell}\|_F$.

Proof 1. Thanks to the orthogonality of the basis $\mathbb{V}_k$ and by incorporating (19) into the definition of the pseudo-inverse of $\mathbb{Q}_k$, we easily get the first equality $\mathbb{Q}_k^{\dagger} = \mathscr{R}_k^{-1}\mathbb{V}_k^T$. Then, we also get the second equality $\mathscr{R}_k^{-1} = \mathbb{Q}_k^{\dagger}\mathbb{V}_k$.

2. The second item is a classical result. Indeed, as $R_k^{\mathrm{sG}} \in R_0 - A\,\mathbb{K}_k(A, R_0)$, then there exists $\mathbb{S}_k^{\mathrm{sG}} = [S_1^{\mathrm{sG}^T}, \ldots, S_k^{\mathrm{sG}^T}]^T \in \mathbb{R}^{kr\times r}$ such that $R_k^{\mathrm{sG}} = R_0 - \mathbb{V}_k\,\mathbb{S}_k^{\mathrm{sG}}$. Using the orthogonality condition (16) we obtain $\mathbb{S}_k^{\mathrm{sG}} = \mathbb{V}_k^T\,R_0$ and so $R_k^{\mathrm{sG}} = (I_n - \mathbb{V}_k\,\mathbb{V}_k^T)\,R_0$.
3. We use (14) and (15) to obtain $X_k^{\mathrm{sC}} = X_0 + [R_0, \mathbb{Q}_{k-1}]\,(\mathbb{T}_k^{\mathrm{sC}})^{-1}\,\mathbb{Q}_k^l\,R_0$. Then, invoking (19) and (20), we get

$$\begin{aligned} X_k &= X_0 + [R_0, \mathbb{V}_{k-1}] \begin{bmatrix} I & 0 \\ 0 & \mathscr{R}_{k-1} \end{bmatrix} (\mathbb{T}_k^{\mathrm{sC}})^{-1}\,\mathbb{L}_k^{-1}\,\mathbb{P}_k^T\,R_0 \\ &= X_0 + [R_0, \mathbb{V}_{k-1}]\,(\mathbb{T}_k^{\mathrm{sG}})^{-1}\,\mathscr{R}_k\,\mathbb{L}_k^{-1}\,\mathbb{P}_k^T\,R_0. \end{aligned}$$

   To proof *(ii)*, it suffices to use (12) and (19).
4. To proof *(i)*, we use the results given in items 1, 2, and 3. *(i)*, this gives

$$\begin{aligned} X_k^{\mathrm{sC}} - X_k^{\mathrm{sG}} &= [R_0, \mathbb{V}_{k-1}]\,(\mathbb{T}_k^{\mathrm{sG}})^{-1}\,\mathscr{R}_k\,\mathbb{Q}_k^l\,R_0 - [R_0, \mathbb{V}_{k-1}]\,(\mathbb{T}_k^{\mathrm{sG}})^{-1}\,\mathbb{V}_k^T\,R_0 \\ &= [R_0, \mathbb{V}_{k-1}]\,(\mathbb{T}_k^{\mathrm{sG}})^{-1}\,(\mathscr{R}_k\,\mathbb{Q}_k^{\ell} - \mathbb{V}_k^T)\,R_0 \\ &= [R_0, \mathbb{V}_{k-1}]\,(\mathbb{T}_k^{\mathrm{sG}})^{-1}\,\mathscr{R}_k\,(\mathbb{Q}_k^{\ell} - \mathbb{Q}_k^{\dagger})\,R_0. \end{aligned}$$

Taking the Frobenius norm, we get

$$\frac{\|X_k^{\rm sC}-X_k^{\rm sG}\|_F}{\|R_0\|_F} \leq \|[R_0,\mathbb{V}_{k-1}]\|_F\,\|(\mathbb{T}_k^{\rm sG})^{-1}\|_F\,\|\mathscr{R}_k\|_F\,\|(\mathbb{Q}_k^{\ell}-\mathbb{Q}_k^{\dagger})\|_F.$$

On the other hand from the results of 3 *(i)* and 2, we have

$$R_k^{\rm sC}-R_k^{\rm sG}=\mathbb{V}_k\,\mathbb{V}_k^T\,R_0-\mathbb{V}_k\,\mathscr{R}_k\,\mathbb{L}_k^{-1}\,\mathbb{P}_k^T\,R_0=\mathbb{V}_k\,(\mathscr{R}_k\,\mathbb{Q}_k^{\dagger}-\mathscr{R}_k\,\mathbb{L}_k^{-1}\,\mathbb{P}_k^T)\,R_0.$$

Again, using the norm we get

$$\|R_k^{\rm sC}-R_k^{\rm sG}\|_F\leq\|\mathscr{R}_k\left(\mathbb{Q}_k^{\dagger}-\mathbb{L}_k^{-1}\,\mathbb{P}_k^T\right)\|_F\,\|R_0\|_F,$$

which implies that

$$\frac{\|R_k^{\rm sC}-R_k^{\rm sG}\|_F}{\|R_0\|_F}\leq\|\mathscr{R}_k\|_F\,\|\mathbb{Q}_k^{\dagger}-\mathbb{L}_k^{-1}\mathbb{P}_k^T\|_F.$$

□

5 Numerical experiments

To evaluate the efficiency of the proposed algorithm, we describe in this section several numerical examples. We compare the behavior of the simpler block CMRH method with the classical block CMRH and also with the classical and simpler versions of the block GMRES method. All the algorithms were coded in Matlab version R2015a. The numerical tests were run on a laptop with a 64-bit Intel(R)-Core(TM) i5 processor at 2.59 GHz with 6 GO RAM. In all numerical examples, we take $X_0=0_{n\times r}$ as the initial guess. In addition we set itermax $=501$ as a maximum number of restarts and consider that a good approximate solution X_m has been obtained when the corresponding residual norm R_m satisfies $\|R_m\|_F\leq\epsilon\,\|R_0\|_F$ where ϵ is a chosen tolerance that will be specified for each example. We note that in the tables of results, we give the number of iterations within a restart (denoted by # iter.), the number of restarts (denoted by # rest.), the number of matrix vector products (denoted by # mv), the CPU time (denoted by Time), the final residual norm (denoted by res. norm), and the final error norm (denoted by err. norm).

Example 1 In this first example, we compare the norm of the true residual $R_k^t=B-A\,X_k$ with that of the recursive residual $R_k^r=R_{k-1}^r-Q_k\,S_k$. The plots correspond to results obtained when using the non-restarted sBCMRH and sBGMRES methods for solving $A_i\,X=B$, $(i=1,2)$ where the matrices $A_1=$ gallery(‘Poisson’,n_0) and $A_2=$ gallery(‘tridiag’,n,c,d,e) are from the Matlab gallery. For the matrix $A_1\in\mathbb{R}^{n_0^2\times n_0^2}$, we considered the following parameters $(n_0,r)=(50,2)$ or $(n_0,r)=(70,5)$, while for the matrix $A_2\in\mathbb{R}^{n\times n}$, we took $c=-5$, $d=10$, $e=5$ and $r=10$ or $r=20$. In all the experiments of this first example, we took $\epsilon=10^{-12}$ and the right-hand side B is such that $B=A_i\,X^*$, where X^* is generated randomly using the Matlab function **rand** which produces entries uniformly distributed in $[0,\ 1]$. This choice allows us to compare the error norm $e_k=\|X_k-X^*\|_F$. The obtained plots and results are given in Fig. 1 and Table 4, respectively.

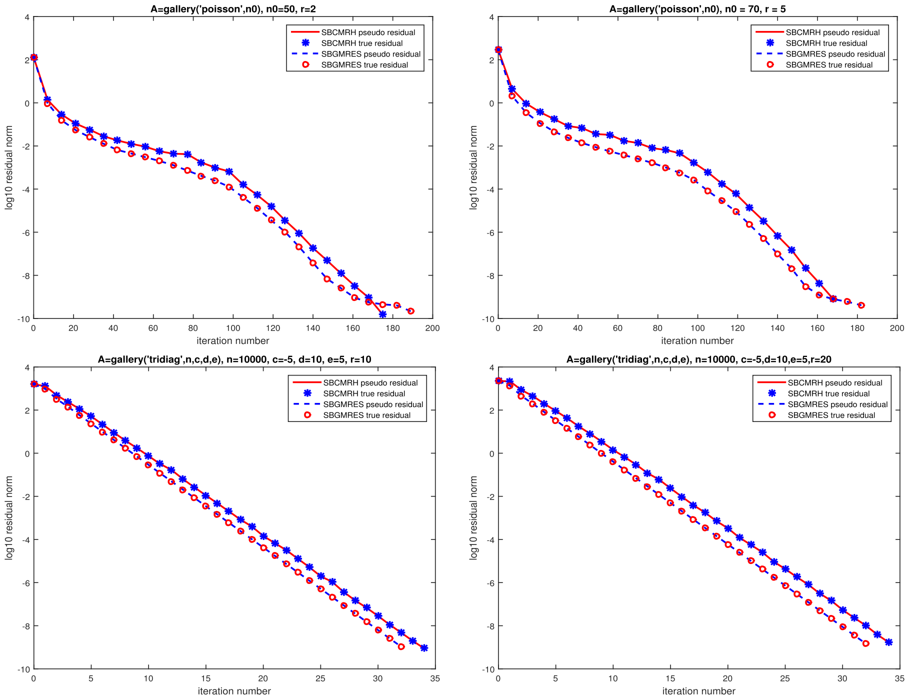

Fig. 1 Comparison of the true and recursive residual norms produced by sBCMRH and sBGMRES with $A_1 = $ gallery('Poisson',n_0) and $A_2 = $ gallery('tridiag',n, c, d, e)

The analysis of the results in Table 4 shows that for case $A = A_1$, the sBCMRH method converged in fewer iterations than the sBGMRES method. On the other hand, for matrix $A = A_2$, the opposite occurred. But in the four reported tests, the sBCMRH method returned the best CPU time. Similarly, by observing the four plots

Table 4 Results for Example 1 obtained with non restarted sBCMRH and sBGMRES for matrices $A_1 = $ gallery('Poisson',n_0) and $A_2 = $ gallery('tridiag',n, c, d, e)

Test problem	Method	# iter./mv	Time	res. norm	err. norm
$A = A_1, n_0 = 50$	sBCMRH	**176/356**	**3.15**	$1.29\,10^{-10}$	$5.56\,10^{-10}$
$n = 2500, r = 2$	sBGMRES	193/390	3.79	$1.07\,10^{-10}$	$7.04\,10^{-10}$
$A = A_1, n_0 = 70$	sBCMRH	**172/870**	**13.37**	$3.10\,10^{-10}$	$1.50\,10^{-9}$
$n = 4900, r = 5$	sBGMRES	185/935	16.89	$2.46\,10^{-10}$	$9.77\,10^{-10}$
$A = A_2, n = 10000, r = 10$	sBCMRH	34/360	**3.10**	$9.06\,10^{-10}$	$3.91\,10^{-11}$
$c = -5, d = 10, e = 5$	sBGMRES	**32/340**	3.14	$1.08\,10^{-09}$	$3.07\,10^{-11}$
$A = A_2, n = 10000, r = 20$	sBCMRH	34/720	**8.34**	$1.72\,10^{-09}$	$6.88\,10^{-11}$
$c = -5, d = 10, e = 5$	sBGMRES	**32/680**	9.18	$1.50\,10^{-09}$	$3.06\,10^{-11}$

in Fig. 1, we can see that for each test and for each of the two compared methods, the curve representing the recursive residual R_k^r merges perfectly with that representing the true residual R_k^t.

Example 2 In this example, we continue to compare the full (non-restarted) versions of the sBCMRH and sBGMRES algorithms. For this purpose, we consider some matrices coming from the University of Florida Sparse Matrix Collection [7]. The characteristics and properties: name, size (n), number of nonzero elements (nnz), and symmetric or not (sym), of the matrices are listed in Table 5. Here again, and in order to compare the error norm, we consider linear systems whose solution is known. Thus, the right-hand side B is equal to $AY/\|AY\|_F$, where Y is generated randomly using the Matlab function **rand**. We also point out that in all this set of experiments we choose $\epsilon = 10^{-10}$ as a tolerance in the stopping criterion.

The results of the sBCMRH algorithm are compared to those of three other methods which are sBGMRES, BCMRH, and BGMRES methods with use of Givens rotations when solving the minimization problem. The obtained results for $r = 2$ or $r = 5$ are given in Table 6.

We note that with the exception of the three cases: (A=memplus, $r = 2$), (A=bodyy5, $r = 2$) and (A=af23560, $r = 2$), the simpler variants of BCMRH and BGMRES are faster than the classical ones. Indeed, in the simpler methods, the solution of the triangular systems associated with the minimization problem is done directly without using Givens rotations that are necessary to determine the QR factorization of the Hessenberg matrix in the case of the classical methods. In addition, we see that the two variants of the BGMRES generally require fewer iterations to converge compared with the variants of the BCMRH method. However, from a time point of view, it is the sBCMRH that provides the best CPU time. We think this is due to the use of LU factorization in the block Hessenberg process instead of the QR factorization in the block Arnoldi process.

We end this example by considering the five largest matrices described in Table 6 to which we apply an ILU0 preconditioning except for the matrix af23560 to which an ILUC preconditioning is applied [25]. The results obtained are reported in Table 7.

The results reported in Table 7 show that with the exception of the case (A=appu), it is the simpler block CMRH method that returns the best CPU time.

Table 5 Properties of the matrices in Example 2

Name	Size n	nnz	Sym	Name	Size n	nnz	Sym
sherman1	1000	3750	No	appu	14000	1853104	No
bcsstm12	1473	19659	Yes	memplus	17758	99147	No
pde2961	2961	14585	No	FEM_3D_thermal1	17880	430740	No
psmigr_3	3140	543160	No	bodyy5	18589	128853	Yes
add32	4960	23884	No	af23560	23560	484256	No

Table 6 Results for Example 2. The compared methods are BCMRH, sBCMRH, BGMRES, and sBGMRES. The matrices are from the University of Florida Sparse Matrix Collection

A	Method	$r = 2$			$r = 5$		
		# iter./mv	Time	Error	# iter./mv	Time	Error
	BCMRH	254/512	0.718	$5.22\,10^{-9}$	133/675	0.609	$2.05\,10^{-9}$
sherman1	sBCMRH	253/510	**0.453**	$9.62\,10^{-9}$	130/660	**0.281**	$2.29\,10^{-8}$
$n = 1000$	BGMRES	**246/496**	1.641	$1.69\,10^{-8}$	**128/650**	1.375	$7.51\,10^{-9}$
	sBGMRES	**246/496**	1.078	$1.73\,10^{-8}$	**128/650**	0.750	$7.50\,10^{-9}$
	BCMRH	558/1120	3.781	$1.11\,10^{-6}$	272/1370	6.047	$4.63\,10^{-7}$
bcsstm12	sBCMRH	557/1118	**2.813**	$8.92\,10^{-7}$	272/1370	**3.328**	$5.10\,10^{-7}$
$n = 1473$	BGMRES	**539/1082**	10.08	$2.17\,10^{-6}$	**268/1350**	7.375	$1.16\,10^{-6}$
	sBGMRES	584/1172	8.391	$2.08\,10^{-6}$	**268/1350**	4.484	$1.37\,10^{-6}$
	BCMRH	202/408	0.796	$1.53\,10^{-10}$	158/800	2.797	$1.02\,10^{-10}$
pde2961	sBCMRH	196/396	**0.640**	$5.35\,10^{-10}$	156/790	**1.781**	$1.75\,10^{-10}$
$n = 2961$	BGMRES	**191/386**	2.047	$6.20\,10^{-10}$	**151/765**	3.250	$2.99\,10^{-10}$
	sBGMRES	**191/386**	1.734	$6.20\,10^{-10}$	**151/765**	2.359	$3.00\,10^{-10}$
	BCMRH	40/84	0.203	$2.46\,10^{-11}$	35/185	0.375	$2.42\,10^{-11}$
psmigr_3	sBCMRH	37/78	**0.171**	$2.83\,10^{-10}$	33/175	**0.265**	$1.17\,10^{-10}$
$n = 3140$	BGMRES	**36/76**	0.203	$5.82\,10^{-10}$	**31/165**	0.390	$2.74\,10^{-10}$
	sBGMRES	**36/76**	0.312	$5.82\,10^{-10}$	**31/165**	0.281	$2.74\,10^{-10}$
	BCMRH	107/218	0.843	$4.96\,10^{-8}$	108/550	1.797	$2.71\,10^{-8}$
add32	sBCMRH	103/210	**0.734**	$6.37\,10^{-8}$	103/525	**1.172**	$4.83\,10^{-8}$
$n = 4960$	BGMRES	**98/200**	0.812	$1.13\,10^{-7}$	**97/495**	1.938	$9.25\,10^{-8}$
	sBGMRES	**98/200**	0.734	$1.13\,10^{-7}$	**97/495**	1.484	$9.25\,10^{-8}$
	BCMRH	98/200	2.141	$2.06\,10^{-10}$	99/505	5.656	$1.31\,10^{-10}$
appu	sBCMRH	96/196	**2.000**	$2.64\,10^{-10}$	96/490	**5.109**	$1.76\,10^{-10}$
$n = 14000$	BGMRES	**91/186**	2.172	$4.89\,10^{-10}$	**89/455**	6.250	$4.10\,10^{-10}$
	sBGMRES	**91/186**	2.078	$4.89\,10^{-10}$	**89/455**	5.781	$4.10\,10^{-10}$
	BCMRH	699/1402	**67.11**	$6.36\,10^{-6}$	511/2565	95.81	$2.23\,10^{-6}$
memplus	sBCMRH	722/1448	67.22	$2.23\,10^{-6}$	512/2570	**88.30**	$2.06\,10^{-6}$
$n = 17758$	BGMRES	**654/1312**	72.61	$4.82\,10^{-6}$	**473/2375**	128.3	$3.82\,10^{-6}$
	sBGMRES	751/1506	89.38	$5.65\,10^{-6}$	529/2655	150.2	$3.98\,10^{-6}$
	BCMRH	238/480	8.531	$1.66\,10^{-9}$	208/1050	16.59	$1.61\,10^{-9}$
FEM_3D_thermal1	sBCMRH	239/482	**7.969**	$1.33\,10^{-9}$	209/1055	**15.70**	$1.10\,10^{-9}$
$n = 17880$	BGMRES	**219/442**	8.563	$5.99\,10^{-9}$	**190/960**	21.42	$4.46\,10^{-9}$
	sBGMRES	**219/442**	8.172	$5.99\,10^{-9}$	**190/960**	20.14	$4.46\,10^{-9}$
	BCMRH	487/978	**51.39**	$8.56\,10^{-11}$	404/2030	62.66	$4.35\,10^{-11}$
bodyy5	sBCMRH	516/1036	54.08	$1.49\,10^{-11}$	422/2120	**60.97**	$1.49\,10^{-11}$
$n = 18589$	BGMRES	**484/972**	59.44	$4.60\,10^{-11}$	**390/1960**	91.09	$3.73\,10^{-11}$
	sBGMRES	559/1122	75.31	$5.26\,10^{-11}$	443/2225	110.1	$5.07\,10^{-11}$
	BCMRH	1073/2150	**287.2**	$5.77\,10^{-10}$	692/3470	230.0	$2.61\,10^{-10}$

Table 6 (continued)

A	method	$r=2$			$r=5$		
		# iter./mv	Time	error	# iter./mv	Time	error
af23560	sBCMRH	1082/2168	289.3	$1.35\,10^{-10}$	700/3510	**220.7**	$7.27\,10^{-11}$
$n=23560$	BGMRES	**1050/2104**	338.8	$3.32\,10^{-10}$	**676/3390**	355.5	$1.24\,10^{-10}$
	sBGMRES	1053/2110	323.0	$2.90\,10^{-10}$	678/3400	340.1	$1.27\,10^{-10}$

Example 3 Here, we provide some experimental results of using the restarted sBCMRH(m) method and compare its performance with those of restarted BCMRH(m), restarted simpler BGMRES, and restarted classical block GMRES(m). In all this set of experiments, the tolerance is $\epsilon = 10^{-12}$ and the right-hand side is

Table 7 Results for Example 2. The compared methods are the preconditioned BCMRH, SBCMRH, BGMRES, and SBGMRES. The matrices are from the University of Florida Sparse Matrix Collection

A	Method	$r=2$			$r=5$		
		# iter./mv	Time	Error	# iter./mv	Time	Error
	pBCMRH	49/102	**1.656**	$1.04\,10^{-10}$	50/260	3.984	$7.75\,10^{-11}$
appu	psBCMRH	48/100	1.813	$2.44\,10^{-10}$	48/250	3.984	$1.40\,10^{-10}$
$n=14000$	pBGMRES	**46/96**	1.984	$2.52\,10^{-10}$	**45/235**	4.641	$2.82\,10^{-10}$
	psBGMRES	**46/96**	1.719	$2.52\,10^{-10}$	**45/235**	**3.953**	$2.82\,10^{-10}$
	pBCMRH	263/530	10.98	$5.00\,10^{-6}$	242/1220	26.38	$1.94\,10^{-6}$
memplus	psBCMRH	268/540	**10.53**	$2.10\,10^{-6}$	229/1155	**22.36**	$1.82\,10^{-6}$
$n=17758$	pBGMRES	**252/508**	12.14	$3.28\,10^{-6}$	**213/1075**	29.50	$3.82\,10^{-6}$
	psBGMRES	259/522	11.94	$2.14\,10^{-6}$	216/1090	28.94	$3.03\,10^{-6}$
	pBCMRH	11/26	0.140	$1.15\,10^{-10}$	11/65	0.343	$1.05\,10^{-10}$
FEM_3D_thermal1	psBCMRH	11/26	**0.171**	$1.09\,10^{-10}$	11/65	**0.281**	$1.02\,10^{-10}$
$n=17880$	pBGMRES	11/26	**0.171**	$6.96\,10^{-11}$	**10/60**	0.375	$4.14\,10^{-10}$
	psBGMRES	11/26	0.187	$6.96\,10^{-11}$	**10/60**	0.328	$4.14\,10^{-10}$
	pBCMRH	50/104	0.921	$3.26\,10^{-12}$	30/160	0.812	$2.65\,10^{-12}$
bodyy5	psBCMRH	48/100	**0.765**	$1.54\,10^{-11}$	29/155	**0.718**	$1.13\,10^{-11}$
$n=18589$	pBGMRES	**44/92**	0.796	$2.18\,10^{-11}$	**28/150**	1.047	$1.67\,10^{-11}$
	psBGMRES	**44/92**	0.781	$2.18\,10^{-11}$	**28/150**	0.921	$1.67\,10^{-11}$
	pBCMRH	7/18	0.281	$1.12\,10^{-10}$	7/45	0.546	$6.15\,10^{-11}$
af23560	psBCMRH	7/18	**0.250**	$1.14\,10^{-10}$	7/45	**0.531**	$5.25\,10^{-11}$
$n=23560$	pBGMRES	7/18	0.390	$1.18\,10^{-10}$	7/45	0.921	$1.97\,10^{-11}$
	psBGMRES	7/18	0.343	$1.18\,10^{-10}$	7/45	0.578	$1.97\,10^{-11}$

computed such that $X^* = I_{n\times r}$ is the exact solution, i.e., $B = A_{:,1:r}$. The coefficient matrix A is generated from the central finite difference discretization of the operator

$$L(u) = \Delta u - x\,\cos(x+y)\,\frac{\partial u}{\partial x} - y\,\sin(x-y)\,\frac{\partial u}{\partial y} - x\,y\,u$$

on the unit square $[0, 1] \times [0, 1]$ with homogeneous Dirichlet boundary conditions. The number of inner grid points in each direction is n_0. Therefore, the dimension of the matrix A is $n = n_0^2$. The results reported in Table 8 are those obtained for different values of n_0, m, and r.

In this example, all the compared methods give similar results in terms of CPU time except when r is relatively large. Indeed, for the cases ($n_0 = 50$, $m = 30$, $r = 20$) and ($n_0 = 150$, $m = 100$, $r = 10$), we observe that the simpler variants are more efficient than the classical ones. In addition, the sBCMRH method is faster

Table 8 Results for Example 3. The restarted BCMRH(m), sBCMRH(m), BGMRES(m), and sBGMRES(m) are compared when applied to matrices obtained from the discritization of operator $L(u)$

n_0, n	m, r		# rest./mv	Time	res. norm	err. norm
		BCMRH	24/976	0.375	$3.14\,10^{-9}$	$1.31\,10^{-10}$
	$m = 20$	sBCMRH	**19/800**	**0.265**	$1.52\,10^{-8}$	$3.41\,10^{-10}$
	$r = 2$	BGMRES	23/952	0.437	$1.18\,10^{-8}$	$4.98\,10^{-10}$
$n_0 = 50$		sBGMRES	23/952	0.375	$1.52\,10^{-8}$	$7.07\,10^{-10}$
$n = 2500$		BCMRH	14/8300	29.57	$1.22\,10^{-8}$	$4.84\,10^{-10}$
	$m = 30$	sBCMRH	10/6180	3.609	$4.80\,10^{-8}$	$2.94\,10^{-10}$
	$r = 20$	BGMRES	**9/5200**	19.25	$4.90\,10^{-8}$	$2.05\,10^{-9}$
		sBGMRES	9/5280	**3.593**	$4.95\,10^{-8}$	$1.90\,10^{-9}$
		BCMRH	170/3742	3.593	$3.01\,10^{-8}$	$1.48\,10^{-9}$
	$m = 10$	sBCMRH	**139/3064**	**3.140**	$5.83\,10^{-8}$	$1.97\,10^{-9}$
	$r = 2$	BGMRES	204/4476	4.343	$6.19\,10^{-8}$	$3.04\,10^{-9}$
$n_0 = 100$		sBGMRES	204/4476	4.765	$6.18\,10^{-8}$	$3.04\,10^{-9}$
$n = 10000$		BCMRH	55/2312	**2.937**	$2.68\,10^{-8}$	$8.29\,10^{-9}$
	$m = 20$	sBCMRH	**51/2136**	2.968	$5.93\,10^{-8}$	$1.75\,10^{-9}$
	$r = 2$	BGMRES	56/2342	3.390	$6.18\,10^{-8}$	$2.99\,10^{-9}$
		sBGMRES	56/2344	3.578	$6.15\,10^{-8}$	$2.96\,10^{-9}$
		BCMRH	9/3268	**29.37**	$9.77\,10^{-8}$	$2.56\,10^{-9}$
	$m = 100$	sBCMRH	9/3256	29.42	$1.87\,10^{-7}$	$1.94\,10^{-9}$
	$r = 4$	BGMRES	7/2636	34.34	$1.96\,10^{-7}$	$6.85\,10^{-9}$
$n_0 = 150$		sBGMRES	**7/2484**	32.79	$1.94\,10^{-7}$	$5.49\,10^{-9}$
$n = 22500$		BCMRH	6/5180	157.0	$1.89\,10^{-7}$	$6.75\,10^{-9}$
	$m = 100$	sBCMRH	4/3160	**90.32**	$2.91\,10^{-7}$	$3.53\,10^{-9}$
	$r = 10$	BGMRES	4/3780	165.9	$3.11\,10^{-7}$	$1.11\,10^{-8}$
		sBGMRES	**3/2290**	94.75	$3.08\,10^{-7}$	$6.88\,10^{-9}$

than the sBGMRES method in five tests and the latter only does better for the test ($n_0 = 50, m = 30, r = 20$).

Example 4 In this last example, we consider the following matrix

$$A = I_n \otimes I_n \otimes A_1 + I_n \otimes A_2 \otimes I_n + A_3 \otimes I_n \otimes I_n,$$

where the matrices A_i ($i = 1, 2, 3$) are as follows:

$$A_i = \frac{\nu}{h^2}\begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix} + \frac{c_i}{4h}\begin{bmatrix} 3 & -5 & 1 & & \\ 1 & 3 & -5 & \ddots & \\ & \ddots & \ddots & \ddots & 1 \\ & & 1 & 3 & -5 \\ & & & 1 & 3 \end{bmatrix}.$$

Here, we mention that the matrices A_i ($i = 1,\ 2,\ 3$) are obtained when a standard finite-difference discretization on equidistant nodes with mesh size $h = 1/(n+1)$ is employed in the discretization on all of the three directions, and a second-order convergent scheme for the convection term is applied to the convection-diffusion equation

$$\begin{aligned} -\nu\, \Delta u + (c_1, c_2, c_3)^T\, \nabla u &= f \text{ in } \Omega = [0, 1] \times [0, 1] \times [0, 1], \\ u &= 0 \text{ on } \partial\Omega. \end{aligned}$$

For more details on this convection diffusion equation and its connection with Sylvester tensor equations, we refer to [4] and [5]. In Tables 9, 10, 11, and 12, we report the results obtained for four cases:

- **case 1.** $\mu = 1, c_1 = c_2 = c_3 = 1$
- **case 2.** $\mu = 1, c_1 = c_2 = c_3 = 10$
- **case 3.** $\mu = 100, c_1 = c_2 = c_3 = 1$
- **case 4.** $\mu = 100, c_1 = c_2 = c_3 = 10$

For each case, we consider the following values $n = 30,\ 50$, $m = 30$, and $r = 1,\ 3,\ 10$. In all this set of experiments, the exact solution is again $X^* = I_{n\times r}$ and the tolerance is $\epsilon = 10^{-10}$.

By comparing the different results reported in Tables 9, 10, 11, and 12, it appears that the sBGMRES method returns the best results in terms of number of restarts and number of matrix-vector products. However, if we look at CPU time, we notice that it is rather the sBCMRH method that is the fastest. Indeed, the latter returned 3/6 (3 times out of 6) the best time for case 1, 5/6 the best time for cases 2 and 3, and 4/6 the best time for case 4. On the other hand, the sBGMRES method only returned the best times 2/6 for case 1, and 1/6 for case 4.

Table 9 Results for Example 4 obtained for case 1 with $\mu = 1$ $c_1 = c_2 = c_3 = 1$

n_0, n	Method	$r = 1$		$r = 3$		$r = 10$	
		# rest./mv	Time	# rest./mv	Time	# rest./mv	Time
	BCMRH	6/181	0.828	5/453	**2.078**	6/1810	2.347
$n_0 = 30$	sBCMRH	5/142	**0.687**	5/447	2.203	5/1510	2.025
$n = 27000$	BGMRES	4/121	0.765	5/453	2.859	4/1210	2.320
	sBGMRES	**4/120**	0.984	**4/366**	2.500	**4/1110**	**1.950**
	BCMRH	6/181	3.078	8/723	4.511	9/1810	16.95
$n_0 = 50$	sBCMRH	5/150	**2.828**	7/624	**3.913**	8/1510	15.37
$n = 125000$	BGMRES	5/151	4.734	6/543	4.852	6/1210	16.11
	sBGMRES	**5/145**	4.703	**6/477**	4.145	**6/1110**	**14.61**

Table 10 Results for Example 4 obtained for case 2 with $\mu = 1$ $c_1 = c_2 = c_3 = 10$

n_0, n	Method	$r = 1$		$r = 3$		$r = 10$	
		# rest./mv	Time	# rest./mv	Time	# rest./mv	Time
	BCMRH	3/91	0.390	3/273	1.672	3/910	1.173
$n_0 = 30$	sBCMRH	2/62	**0.296**	**2/183**	**0.890**	2/610	**0.790**
$n = 27000$	BGMRES	2/61	0.390	**2/183**	1.141	2/610	1.116
	sBGMRES	**2/60**	0.421	**2/183**	1.219	**2/590**	1.039
	BCMRH	3/91	1.500	4/363	2.230	5/1510	9.547
$n_0 = 50$	sBCMRH	**3/67**	**1.094**	3/273	**1.697**	6/1530	8.820
$n = 125000$	BGMRES	3/91	3.000	3/273	2.394	**3/910**	**7.603**
	sBGMRES	3/82	2.375	**3/252**	2.120	**3/910**	7.623

Table 11 Results for Example 4 obtained for case 3 with $\mu = 100$ $c_1 = c_2 = c_3 = 1$

n_0, n	Method	$r = 1$		$r = 3$		$r = 10$	
		# rest./mv	Time	# rest./mv	Time	# rest./mv	Time
	BCMRH	5/151	**0.640**	6/543	2.516	6/1810	23.61
$n_0 = 30$	sBCMRH	6/154	0.765	5/441	**2.203**	6/1630	**20.78**
$n = 27000$	BGMRES	5/151	1.047	5/453	2.813	5/1510	27.69
	sBGMRES	**5/133**	0.875	**5/408**	2.656	**4/1190**	21.78
	BCMRH	7/211	3.828	8/723	43.56	9/2710	157.0
$n_0 = 50$	sBCMRH	7/183	**3.047**	7/600	**36.30**	9/2570	**148.4**
$n = 125000$	BGMRES	6/181	5.484	6/543	48.69	7/2110	184.0
	sBGMRES	**6/161**	4.969	**6/528**	45.81	**6/1740**	152.6

Table 12 Results for Example 4 obtained for case 4 with $\mu = 100$ $c_1 = c_2 = c_3 = 10$

n_0, n	Method	$r = 1$		$r = 3$		$r = 10$	
		# rest./mv	Time	# rest./mv	Time	# rest./mv	Time
	BCMRH	5/151	**0.656**	6/543	2.578	5/1510	19.41
$n_0 = 30$	sBCMRH	5/152	0.750	5/453	**2.188**	5/1420	**17.80**
$n = 27000$	BGMRES	5/151	0.906	5/453	2.750	5/1510	27.75
	sBGMRES	**5/132**	0.906	**5/405**	2.641	**4/1190**	21.81
	BCMRH	7/211	3.875	8/723	44.78	9/2710	153.7
$n_0 = 50$	sBCMRH	6/180	**3.109**	7/594	**36.47**	9/2540	134.5
$n = 125000$	BGMRES	6/181	5.500	6/543	48.77	7/2110	158.1
	sBGMRES	**6/159**	5.000	**6/522**	46.14	**6/1730**	**129.0**

6 Conclusion

By reformulating the block Hessenberg process to start with $A\,R_0$ instead of R_0, we have described the simpler block CMRH method which is a new implementation of the BCMRH that avoids the update of the QR factorization of the upper block Hessenberg matrix. Moreover, the simpler block CMRH allows to check the convergence within each cycle of the block Hessenberg process by using a recursive relation that updates the residual at each iteration. This is an important difference with the classical BCMRH method in which only an estimate of the residual norm can be obtained. Since the solution of the triangular systems associated with the minimization problem is done directly without using the QR factorization of the Hessenberg matrix, the simpler BCMRH is more economical and needs less arithmetic requirements than the classical BCMRH and BGMRES and also less than the simpler BGMRES methods. The various numerical tests we have performed show the good behavior of the new proposed method. Comparing the CPU time, and thanks to the use of the LU factorization in the block Hessenberg process instead of the QR factorization in the block Arnoldi process, we see that the simpler BCMRH method needs in many examples less time than the simpler BGMRES method.

Acknowledgments The authors are grateful to the anonymous referees for their valuable comments and suggestions which helped to improve the quality of this paper.

Appendix

In this section, we give Algorithms 6 and 7 a brief description of the restarted standard block GMRES and restarted simpler block GMRES methods.

Algorithm 6 The restarted block GMRES method: BGMRES(m).

Input: A an $n \times n$ matrix, B an $n \times r$ matrix, X_0 an $n \times r$ matrix (initial guess), m an integer and ϵ a desired tolerance.

1: Compute $R_0 = B - AX_0$; Compute the QR decomposition of R_0, i.e., $[V_1,\ H_{1,0}] = \mathbf{qr}(R_0)$;
2: **for** $k = 1, \ldots, m$ **do**
3: $\quad \widetilde{V}_{k+1}^{(0)} = A\, V_k$;
4: $\quad$ **for** $j = 1, 2, \ldots, k$ **do**
5: $\quad\quad H_{j,k} = V_j^T\, \widetilde{V}_{k+1}^{(j-1)}$;
6: $\quad\quad \widetilde{V}_{k+1}^{(j)} = \widetilde{V}_{k+1}^{(j-1)} - V_j\, H_{j,k}$;
7: $\quad$ **end for**
8: $\quad$ Compute the QR decomposition of $V_{k+1}^{(k)}$, i.e., $[V_{k+1},\ H_{k+1,k}] = \mathbf{qr}(\widetilde{V}_{k+1}^{(k)})$;
9: $\quad$ Update the QR factorization of $\widetilde{\mathbb{H}}_k$ and determine ρ_k the norm of the residual R_k;
10: $\quad$ **if** $\rho_k \leq \epsilon$ **then**
11: $\quad\quad$ replace m by k, i.e. $m = k$;
12: $\quad\quad$ go to line (15); (there is no need to determine the solution Y_k yet)
13: $\quad$ **end if**
14: **end for**
15: Determine $Y_m = \underset{Y \in \mathbb{R}^{m\,r \times r}}{\operatorname{argmin}} \| E_1\, H_{1,0} - \widetilde{\mathbb{H}}_m\, Y \|_F$ by solving the triangular linear system obtained after the QR factorization of $\widetilde{\mathbb{H}}_m$;
16: Compute the approximate solution $X_m = X_0 + \mathbb{V}_m\, Y_m$;
17: **if** $\rho_m \leq \epsilon$ **then**
18: $\quad$ accept X_m and exit;
19: **else**
20: $\quad X_0 = X_m$ and go to line 1;
21: **end if**

Algorithm 7 The restarted simpler block GMRES method: sBGMRES(m).

Input: A an $n \times n$ matrix, B an $n \times r$ matrix, X_0 an $n \times r$ matrix (initial guess), m an integer and ϵ a desired tolerance.

```
Compute R_0 = B − A X_0; Z_1 = R_0; Compute the QR decomposition of A R_0,
    i.e., [Q_1, T_{1,1}] = qr(A R_0);
Compute S_1 = Q_1^T R_0; R_1 = R_0 − Q_1 S_1;
if ||R_1||_F ≤ ε then
    solve 𝕋_1 Y_1 = 𝕊_1; compute X_1 = X_0 + R_0 Y_1;
    exit;
end if
for k = 2, ..., m do
    Q̃_k^{(0)} = A Q_{k−1};
    for j = 1, 2, ..., k − 1 do
        T_{j,k} = Q_j^T Q̃_k^{(j−1)};
        Q̃_k^{(j)} = Q̃_k^{(j−1)} − Q_j T_{j,k};
    end for
    Compute the QR decomposition of Q̃_k^{(k−1)}, i.e., [Q_k, T_{k,k}] = qr(Q̃_k^{(k−1)});
    Compute S_k = Q_k^T R_k; R_k = R_{k−1} − Q_k S_k;
    if ||R_k||_F ≤ ε then
        solve 𝕋_k Y_k = 𝕊_k; compute X_k = X_0 + ℤ_k Y_k;
        exit;
    end if
end for
solve 𝕋_m Y_m = 𝕊_m; compute X_m = X_0 + ℤ_m Y_m;
X_0 = X_m;
go to line 1;
```

References

1. Addam, M., Elbouyahyaoui, L., Heyouni, M.: On Hessenberg type methods for low-rank Lyapunov matrix equations. Applicationes Mathematicae **45**, 255–273 (2018)
2. Addam, M., Heyouni, M., Sadok, H.: The block Hessenberg process for matrix equations. Electron. Trans. Numer. Anal. **46**, 460–473 (2017)
3. Amini, S., Toutounian, F., Gachpazan, M.: The block CMRH method for solving nonsymmetric linear systems with multiple right-hand sides. J. Comput. Appl. Math. **337**, 166–174 (2018)
4. Ballani, J., Grasedyck, L.: A projection method to solve linear systems in tensor format. Numerical Linear Algebra with Applications **20**(1), 27–43 (2013)
5. Beik, F.P.A., Saberi-Movahed, F., Ahmadi-Asl, S.: On the Krylov subspace methods based on tensor format for positive definite Sylvester tensor equations. Numerical Linear Algebra with Applications **23**(3), 444–466 (2016)
6. Datta, B.N., Saad, Y.: Arnoldi methods for large Sylvester-like observer matrix equations, and an associated algorithm for partial spectrum assignment. Linear Algebra Appl. **154-156**, 225–244 (1991)
7. Davis, T.A., Hu, Y.: The University of Florida sparse matrix collection. ACM Trans. Math. Softw. **38**(1), 1,1–1,25 (2011)

8. Duminil, S., Heyouni, M., Marion, P., Sadok, H.: Algorithms for the CMRH method for dense linear systems. Numerical Algorithms **71**(2), 383–394 (2016)
9. El Guennouni, A., Jbilou, K., Sadok, H.: A block version of BiCGSTAB for linear systems with multiple right-hand sides. Electron. Trans. Numer. Anal. **16**, 129–142 (2003)
10. El Guennouni, A., Jbilou, K., Sadok, H.: The block Lanczos method for linear systems with multiple right-hand sides. Appl. Numer. Math. **51**(2), 243–256 (2004)
11. Freund, R.W., Malhotra, M.: A block QMR algorithm for non-hermitian linear systems with multiple right-hand sides. Linear Algebra and its Applications **254**(1), 119–157 (1997). Proceeding of the Fifth Conference of the International Linear Algebra Society
12. Freund, R.W., Nachtigal, N.M.: QMR: a quasi-minimal residual method for non-Hermitian linear systems. Numer. Math. **60**(1), 315–339 (1991)
13. Golub, G.H., Van Loan, C.F. Matrix computations, 4th edn. The Johns Hopkins University Press, Baltimore (2013)
14. Gu, X.-M., Huang, T.-Z., Carpentieri, B., Imakura, A., Zhang, K., Du, L.: Effecient variants of the CMRH method for solving multi-shifted non-Hermitian linear systems. arXiv (2018)
15. Gu, X.-M., Huang, T.-Z., Yin, G., Carpentieri, B., Wen, C., Du, L.: Restarted Hessenberg method for solving shifted nonsymmetric linear systems. J. Comput. Appl. Math. **331**, 166–177 (2018)
16. Gutknecht, M.H.: Block Krylov Space Methods for Linear Systems with Multiple Right-Hand Sides: An Introduction (2006)
17. Gutknecht, M.H., Schmelzer, T.: The block grade of a block Krylov space. Linear Algebra Appl. **430**(1), 174–185 (2009)
18. Heyouni, M., Sadok, H.: A new implementation of the CMRH method for solving dense linear systems. J. Comput. Appl. Math. **213**(2), 387–399 (2008)
19. Jiránek, P., Rozloznik, M., Gutknecht, M.: How to make simpler GMRES and GCR more stable. SIAM Journal on Matrix Analysis and Applications **30**(4), 1483–1499 (2009)
20. Jiránek, P., Rozožní, M.: Adaptive version of simpler GMRES. Numerical Algorithms **53**(1), 93 (2009)
21. Kress, R. Linear Integral Equations, 3rd edn. Springer, New York (1989)
22. Liu, H., Zhong, B.: Simpler Block GMRES for nonsymmetric systems with multiple right-hand sides. Electron. Trans. Numer. Anal. **30**, 1–9 (2008)
23. Morgan, R.B.: Restarted block-GMRES with deflation of eigenvalues. Appl. Numer. Math. **54**(2), 222–236 (2005)
24. O'Leary, D.P.: The block conjugate gradient algorithm and related methods. Linear Algebra Appl. **29**, 293–322 (1980)
25. Saad, Y.: Iterative methods for sparse linear systems. Society for Industrial and Applied Mathematics, second edition (2003)
26. Sadok, H.: CMRH: A new method for solving nonsymmetric linear systems based on the Hessenberg reduction algorithm. Numerical Algorithms **20**(4), 303–321 (1999)
27. Sadok, H., Szyld, D.B.: A new look at CMRH and its relation to GMRES. BIT Numer. Math. **52**(2), 485–501 (2012)
28. Simoncini, V., Gallopoulos, E.: An iterative method for nonsymmetric systems with multiple right-hand sides. SIAM J. Sci. Comput. **16**(4), 917–933 (1995)
29. Simoncini, V., Gallopoulos, E.: Convergence properties of block GMRES and matrix polynomials. Linear Algebra Appl. **247**, 97–119 (1996)
30. Soodhalter, K.: Block Krylov subspace recycling for shifted systems with unrelated right-hand sides. SIAM J. Sci. Comput. **38**(1), A302–A324 (2016)
31. Sun, D.-L., Carpentieri, B., Huang, T.-Z., Jing, Y.-F.: A spectrally preconditioned and initially deflated variant of the restarted block GMRES method for solving multiple right-hand sides linear systems. Int. J. Mech. Sci. **144**, 775–787 (2018)
32. Sun, D.-L., Huang, T.-Z., Carpentieri, B., Jing, Y.-F.: A new shifted block GMRES method with inexact breakdowns for solving multi-shifted and multiple right-hand sides linear systems. J. Sci. Comput. **78**(2), 746–769 (2019)
33. Sun, D.-L., Huang, T.-Z., Jing, Y.-F., Carpentieri, B.: A block GMRES method with deflated restarting for solving linear systems with multiple shifts and multiple right-hand sides. Numerical Linear Algebra with Applications **25**(5), e2148 (2018). e2148 nla.2148
34. Vital, B.: Etude de quelques méthodes de résolution de problèmes linéaires de grande taille sur multiprocesseur. PhD thesis, Université, Rennes, 1 (1990)

35. Walker, H.F., Zhou, L.: A simpler GMRES. Numerical Linear Algebra with Applications **1**(6), 571–581 (1994)
36. Wilkinson, J.H.: The Algebraic Eigenvalue Problem. Oxford University Press, Inc., New York (1988)
37. Xiu Zhong, H., Wu, G., Liang Chen, G.: A flexible and adaptive simpler block GMRES with deflated restarting for linear systems with multiple right-hand sides. J. Comput. Appl. Math. **282**, 139–156 (2015)