A hybridized iterative algorithm of the BiCORSTAB and GPBiCOR methods for solving non-Hermitian linear systems

Gu, Xian-Ming; Huang, Ting-Zhu; Carpentieri, Bruno; Li, Liang

Published in:
Computers & Mathematics with Applications

DOI:
10.1016/j.camwa.2015.10.012

Publication date:
2015

Computers and Mathematics with Applications 70 (2015) 3019–3031

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

A hybridized iterative algorithm of the BiCORSTAB and GPBiCOR methods for solving non-Hermitian linear systems

Xian-Ming Gu [a,b], Ting-Zhu Huang [a,*], Bruno Carpentieri [b], Liang Li [a], Chun Wen [a]

[a] School of Mathematical Sciences, University of Electronic Science and Technology of China, Chengdu, 611731, PR China

[b] Institute of Mathematics and Computing Science, University of Groningen, Nijenborgh 9, PO Box 407, 9700 AK Groningen, The Netherlands

ARTICLE INFO

Article history:
Received 16 March 2015
Received in revised form 15 October 2015
Accepted 18 October 2015
Available online 6 November 2015

Keywords:
Non-Hermitian linear systems
Krylov subspace method
BiCORSTAB
GPBiCOR
GPBiCG
Residual polynomial

ABSTRACT

In this study, we derive a new iterative algorithm (including its preconditioned version) which is a hybridized variant of the biconjugate A-orthogonal residual stabilized (BiCORSTAB) method and the generalized product-type solvers based on BiCOR (GPBiCOR) method. The proposed method, which is named GPBiCOR(m, ℓ) similarly to the GPBiCG(m, ℓ) method proposed by Fujino (2002), can be regarded as an extension of the BiCORSTAB2 method introduced by Zhao and Huang (2013). Inspired by Fujino's idea for improving the BiCGSTAB2 method, in the established GPBiCOR(m, ℓ) method the parameters computed by the BiCORSTAB method are chosen at successive m iteration steps, and afterwards the parameters of the GPBiCOR method are utilized in the subsequent ℓ iteration steps. Therefore, the proposed method can inherit the low computational cost of BiCORSTAB and the attractive convergence of GPBiCOR. Extensive numerical convergence results on selected real and complex matrices are shown to assess the performances of the proposed GPBiCOR(m, ℓ) method, also against other popular non-Hermitian Krylov subspace methods.

1. Introduction

In many computational science and engineering applications, there is a strong need of fast and efficient solutions of large sparse linear systems

$$A\boldsymbol{x} = \boldsymbol{b}, \tag{1}$$

where A is an $n \times n$ non-Hermitian and possibly indefinite matrix, and $\boldsymbol{b}$ is the right-hand side vector of length n. There are two classes of numerical methods for solving the linear system (1) on modern computers, namely direct and iterative solvers. Sparse direct methods are generally very accurate, robust, and predictable in terms of both storage and algorithmic cost (e.g. see [1]). However, they tend to be very expensive for solving large problems. A large number of researchers are still interested in developing robust iterative solvers, especially the well-known class of Krylov subspace methods, as they only require matrix–vector products; refer to [2–5]. As we already know, if the coefficient matrix A is Hermitian positive definite, the conjugate gradient (CG) method [6] is a very good choice. In addition, if the coefficient matrix A is complex symmetric,

* Corresponding author. Tel.: +86 28 61831608.
E-mail addresses: guxianming@live.cn, x.m.gu@rug.nl (X.-M. Gu), tingzhuhuang@126.com (T.-Z. Huang), bcarpentieri@gmail.com (B. Carpentieri), plum_liliang@uestc.edu.cn (L. Li), wchun17@163.com (C. Wen).

http://dx.doi.org/10.1016/j.camwa.2015.10.012
0898-1221/

i.e. $A = A^T$ and $A \neq A^H$, Krylov subspace methods can exploit symmetry during the iterative process; refer, e.g., to our recent work [7,8] about the SCBiCG class of iterative algorithms. In the case, the coefficient matrix A is substantially non-Hermitian (e.g., see [2,3] and references therein), two popular non-Hermitian iterative solvers are the generalized minimal residual (GMRES) method proposed by Saad and Schultz [9], and the generalized conjugate residual (GCR) method proposed by Eisenstat, Elman and Schultz [10]. Both the GMRES and GCR methods search the *optimal* approximation $\boldsymbol{x}_m$ for which the 2-norm of the residual $\boldsymbol{r}_m$ is minimal over the Krylov space $\mathcal{K}_m(A, \boldsymbol{r}_0) = \text{span}\{\boldsymbol{r}_0, A\boldsymbol{r}_0, \ldots, A^{m-1}\boldsymbol{r}_0\}$ for $m = 1, 2, \ldots$, where $\boldsymbol{r}_0$ denotes the initial residual. However, due to the linearly increasing memory and computational costs with the iteration count, the two algorithms are often restarted after each cycle of, say, m iterations. The restarted GMRES and GCR variants, dubbed by GMRES(m) [2,9, pp. 179–180] and GCR(m) [10], however, do not share the optimality property of the original methods, and in practice they show some difficulties to converge on difficult problems. Some strategies to recover the optimal convergence behavior, while keeping the cost per iteration roughly constant, such as deflation and augmentation accelerating techniques, are described in [4,11–15].

On the other side, a prominent *non-optimal* Krylov subspace method for non-Hermitian systems is the biconjugate gradient (BiCG) method proposed in [16,17]. However, the BiCG method displays irregular convergence behavior in many practical applications and requires two matrix–vector products, one with A and one with the transpose conjugate matrix A^H per iteration step. Several variants of the BiCG method have been proposed to enhance its overall performance, such as the conjugate gradient squared (CGS) method proposed by Sonneveld [18], the generalized CGS (GCGS) method proposed by Fokkema, Sleijpen and van der Vorst [19], Freund and Nachtigal's transpose-free quasi-minimal residual (TFQMR) method [20], van der Vorst's biconjugate gradient stabilized (BiCGSTAB) method [21], the BiCGSTAB2 method by Gutknecht [22], the BiCGSTAB(ℓ) method by Sleijpen and Fokkema [23], and the generalized product-type method based on BiCG (GPBiCG) method introduced by Zhang [24,25]. Moreover, the reborn induced dimension reduction (abbreviated as IDR(s)) method, which is also an efficient short-recurrence family of Krylov subspace methods, was proposed by Sonneveld and van Gijzen in [26]. A different approach to generalize BiCGSTAB was proposed by Yeung and Chan, whose ML(k)BiCGSTAB method [27] is a BiCGSTAB variant based on multiple left Lanczos starting vectors; for a comparative study of the IDR(s) and ML(k)BiCGSTAB methods, see [28].

It is a question to determine the classes of problems for which one algorithm is more numerically stable and memory efficient than others. For example, in many problems the CGS method is significantly faster than the BiCG method, but the convergence behavior is much more irregular, and this can potentially affect the final convergence rate and accuracy of the solution. Instead of squaring the BiCG polynomial [16] as in CGS, the authors of [19] consider products of two nearby BiCG polynomials, which results in the GCGS method. The BiCGSTAB2 and BiCGSTAB(ℓ) methods integrate both first and second (or higher) degree auxiliary polynomials, amending the convergent stability of the BiCGSTAB method on many problems. In contrast, the GPBiCG method utilizes only second degree auxiliary polynomials. Therefore each GPBiCG iteration step is slightly more expensive than other product-type iterative methods; similarly, refer to Table 1 about the cost for computing the scalar parameters. However, the idea behind the GPBiCG method implies that it is further possible to develop hybridized variants which attempt to select the different parameters of the algorithm to minimize its final cost; see [24] for details.

In recent years, we [29,30] proposed a novel class of efficient *non-optimal* iterative methods for solving non-Hermitian linear systems (1), dubbed biconjugate A-orthogonal residual (BiCOR) family. The BiCOR family of solvers requires only $\mathcal{O}(n)$ extra storage in addition to the matrix and performs $\mathcal{O}(n)$ operations per iteration, often shows competitive and sometimes superior performance against other popular Krylov subspace solvers in solving different scientific and engineering applications [31,32]. The first variant, named the BiCOR method, can have much smoother and faster convergence behavior than the BiCG method. The same residual polynomial of the BiCOR method is utilized and squared in the second variant, the CORS method [29]. The convergence of the CORS method may be typically more irregular than that of the BiCOR method when the BiCOR method has a jagged convergence curve. In order to overcome this convergence problem, the BiCORSTAB method has been established exploiting the same philosophy behind the BiCGSTAB method. While the BiCORSTAB method works well in a lot of cases, it still has quite erratic oscillation in some difficult problems with complex spectrum; see e.g., the experiments reported in [29,32] on the representative matrix problems. By applying the ideas behind the BiCORSTAB, CORS, CGS and BiCGSTAB methods, a few generalized methods have been recently proposed such as the generalized product-type solvers based on BiCOR (GPBiCOR) method introduced by Zhao, Huang, Jing et al. [33], the generalized CORS (GCORS) method by Zhang and Dai [34], the QMOR method also by Zhang and Dai [35], the TFQMR-like variant of the CORS (TFQMORS) method by Zhang and Dai [36], and the BiCORSTAB2 method by Zhao and Huang [37]. Again, the GPBiCOR method utilizes always the second degree polynomial only similar to the case of GPBiCG. Therefore, cost of the GPBiCOR method per iteration step is slightly more expensive than those of other product-type iterative methods such as CORS and BiCORSTAB. By the way, it is also noted that almost simultaneously the authors of [38] independently constructed the hybrid BiCR methods for solving nonsymmetric linear systems via replacing the BiCG part in the residual polynomial of the hybrid BiCG variants with BiCR [39], but the comparison of the BiCOR family of solvers [29,30] and the hybrid BiCR variants [38] is not the emphasis of this present study.

A mechanism of the GPBiCOR method has indicated further possibility of constructing a new hybridized variant in which one can vary from the CORS and BiCORSTAB methods to the BiCORSTAB2 method with suitable choice of parameters [33]. In this paper, we investigate this possibility. The research was prompted by the pioneering work about the GPBiCG(m, ℓ) method [40], which makes an attempt to construct a hybridized iterative procedure for reducing the cost of the GPBiCG iterations. The GPBiCOR method is considerably more competitive than the GPBiCG method, see [33] for details, but this

Table 1
Choice of parameters η_n and ζ_n of some product-type iterative methods.

Methods	Choice of parameters η_n and ζ_n
CORS	$\eta_n = \frac{\beta_{n-1}}{\alpha_{n-1}}\alpha_n, \quad \zeta_n = \alpha_n$
BiCORSTAB	$\eta_n = 0, \quad \zeta_n = \frac{\langle \boldsymbol{s}_n, \boldsymbol{t}_n\rangle}{\langle \boldsymbol{s}_n, \boldsymbol{s}_n\rangle}$
GPBiCOR	$\eta_n = \frac{\langle \boldsymbol{s}_n, \boldsymbol{s}_n\rangle\langle \boldsymbol{y}_n, \boldsymbol{t}_n\rangle - \langle \boldsymbol{y}_n, \boldsymbol{s}_n\rangle\langle \boldsymbol{s}_n, \boldsymbol{t}_n\rangle}{\langle \boldsymbol{s}_n, \boldsymbol{s}_n\rangle\langle \boldsymbol{y}_n, \boldsymbol{y}_n\rangle - \langle \boldsymbol{y}_n, \boldsymbol{s}_n\rangle\langle \boldsymbol{s}_n, \boldsymbol{y}_n\rangle}$, $\zeta_n = \frac{\langle \boldsymbol{y}_n, \boldsymbol{y}_n\rangle\langle \boldsymbol{s}_n, \boldsymbol{t}_n\rangle - \langle \boldsymbol{y}_n, \boldsymbol{t}_n\rangle\langle \boldsymbol{s}_n, \boldsymbol{y}_n\rangle}{\langle \boldsymbol{s}_n, \boldsymbol{s}_n\rangle\langle \boldsymbol{y}_n, \boldsymbol{y}_n\rangle - \langle \boldsymbol{y}_n, \boldsymbol{s}_n\rangle\langle \boldsymbol{s}_n, \boldsymbol{y}_n\rangle}$
BiCORSTAB2	At even iteration step: $\eta_n = 0, \ \zeta_n = \frac{\langle \boldsymbol{s}_n, \boldsymbol{t}_n\rangle}{\langle \boldsymbol{s}_n, \boldsymbol{s}_n\rangle}$ At odd iteration step: $\eta_n = \frac{\langle \boldsymbol{s}_n, \boldsymbol{s}_n\rangle\langle \boldsymbol{y}_n, \boldsymbol{t}_n\rangle - \langle \boldsymbol{y}_n, \boldsymbol{s}_n\rangle\langle \boldsymbol{s}_n, \boldsymbol{t}_n\rangle}{\langle \boldsymbol{s}_n, \boldsymbol{s}_n\rangle\langle \boldsymbol{y}_n, \boldsymbol{y}_n\rangle - \langle \boldsymbol{y}_n, \boldsymbol{s}_n\rangle\langle \boldsymbol{s}_n, \boldsymbol{y}_n\rangle}$, $\zeta_n = \frac{\langle \boldsymbol{y}_n, \boldsymbol{y}_n\rangle\langle \boldsymbol{s}_n, \boldsymbol{t}_n\rangle - \langle \boldsymbol{y}_n, \boldsymbol{t}_n\rangle\langle \boldsymbol{s}_n, \boldsymbol{y}_n\rangle}{\langle \boldsymbol{s}_n, \boldsymbol{s}_n\rangle\langle \boldsymbol{y}_n, \boldsymbol{y}_n\rangle - \langle \boldsymbol{y}_n, \boldsymbol{s}_n\rangle\langle \boldsymbol{s}_n, \boldsymbol{y}_n\rangle}$
GPBiCOR(m, ℓ)	At consecutive m iteration steps: $\eta_n = 0, \ \zeta_n = \frac{\langle \boldsymbol{s}_n, \boldsymbol{t}_n\rangle}{\langle \boldsymbol{s}_n, \boldsymbol{s}_n\rangle}$ Afterwards at consecutive ℓ iteration steps: $\eta_n = \frac{\langle \boldsymbol{s}_n, \boldsymbol{s}_n\rangle\langle \boldsymbol{y}_n, \boldsymbol{t}_n\rangle - \langle \boldsymbol{y}_n, \boldsymbol{s}_n\rangle\langle \boldsymbol{s}_n, \boldsymbol{t}_n\rangle}{\langle \boldsymbol{s}_n, \boldsymbol{s}_n\rangle\langle \boldsymbol{y}_n, \boldsymbol{y}_n\rangle - \langle \boldsymbol{y}_n, \boldsymbol{s}_n\rangle\langle \boldsymbol{s}_n, \boldsymbol{y}_n\rangle}$, $\zeta_n = \frac{\langle \boldsymbol{y}_n, \boldsymbol{y}_n\rangle\langle \boldsymbol{s}_n, \boldsymbol{t}_n\rangle - \langle \boldsymbol{y}_n, \boldsymbol{t}_n\rangle\langle \boldsymbol{s}_n, \boldsymbol{y}_n\rangle}{\langle \boldsymbol{s}_n, \boldsymbol{s}_n\rangle\langle \boldsymbol{y}_n, \boldsymbol{y}_n\rangle - \langle \boldsymbol{y}_n, \boldsymbol{s}_n\rangle\langle \boldsymbol{s}_n, \boldsymbol{y}_n\rangle}$

algorithm suffers from the slightly more expensive cost during the iteration procedure, as shown in Table 1. We exploit the ingenious idea behind GPBiCG(m, ℓ) to establish a hybridized cost-efficient variant of BiCORSTAB and GPBiCOR. The proposed method not only generalizes earlier established iterative methods as the BiCORSTAB, GPBiCOR and BiCORSTAB2 methods, but can improve their convergence to some extent, as illustrated by our numerical results presented in Section 3.

The remainder of the paper is organized as follows. In Section 2, we briefly review the GPBiCG(m, ℓ) method and the derivations of GPBiCOR and GPBiCG(m, ℓ), then we establish the GPBiCOR(m, ℓ) method and describe the corresponding algorithms and parameters of the BiCORSTAB, BiCORSTAB2 and GPBiCOR methods of the product-type methods. In Section 3, some test problems involving both real and complex matrices are employed to illustrate the effectiveness of the proposed solvers. Finally, the paper closes with conclusions in Section 4.

2. GPBiCG(m, ℓ) method and a new variant of the GPBiCOR method

In this section, we will first briefly introduce the main idea of GPBiCG(m, ℓ) proposed by Fujino, and then give some remarks on this method. Since the GPBiCOR method is closely related to the GPBiCG method, so we can exploit Fujino's trick to derive a new hybridized variant of GPBiCOR.

2.1. The GPBiCG(m, ℓ) method

Let us begin by reviewing the GPBiCG method [24], recall $\boldsymbol{x}_0$ and $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$ denote the initial guess and the corresponding initial residual, respectively. Then, the residual vector $\boldsymbol{r}_k^{\mathrm{BiCG}}$ generated by BiCG iterations can be formed by $\boldsymbol{r}_k^{\mathrm{BiCG}} = R_k(A)\boldsymbol{r}_0$, where $R_k(\lambda)$ with degree k is the residual polynomial of BiCG. It writes as a multiple of the so-called Bi-Lanczos polynomial [3,16,24], which is in accordance with the following recurrence relations

$$\begin{cases} R_0(\lambda) = 1, \qquad R_1(\lambda) = 1 - \alpha_0\lambda, \\ R_{k+1}(\lambda) = \left(1 + \alpha_k\dfrac{\beta_{k-1}}{\alpha_{k-1}} - \alpha_k\lambda\right)R_k(\lambda) - \alpha_k\dfrac{\beta_{k-1}}{\alpha_{k-1}}R_{k-1}(\lambda), \quad k = 1, 2, \ldots \end{cases} \tag{2}$$

for certain coefficients α_k and β_{k-1}. The residual vector $\boldsymbol{r}_k$ of the hybrid BiCG methods is expressed as $M_k(A)\boldsymbol{r}_k^{\mathrm{BiCG}}$ by combining BiCG with an auxiliary polynomial $M_k(\lambda)$ of degree k. The polynomial $M_k(\lambda)$ used in the GPBiCG method is computed from the following recurrence

$$\begin{cases} M_0(\lambda) = 1, \qquad M_1(\lambda) = 1 - \zeta_0\lambda, \\ M_{k+1}(\lambda) = (1 + \eta_k - \zeta_k\lambda)M_k(\lambda) - \eta_k M_{k-1}(\lambda), \quad k = 1, 2, \ldots \end{cases} \tag{3}$$

which can make the residual of BiCG converge to zero faster. Then the GPBiCG method needs to compute the coefficients η_k and ζ_k by minimizing the term

$$\min_{\eta_k,\zeta_k} \|\boldsymbol{r}_k\|_2 = \min_{\eta_k,\zeta_k} \|M_k(A)\boldsymbol{r}_k^{\mathrm{BiCG}}\|_2 = \min_{\eta_k,\zeta_k} \|M_k(A)R_k(A)\boldsymbol{r}_0\|_2. \tag{4}$$

In many applications, the GPBiCG method is indeed more efficient and robust than the BiCGSTAB method. However, the total computational time can be higher than that of the BiCGSTAB method due to the solution of the minimization problem (4). Motivated by this computational problem, Fujino recently proposed to utilize the sectional iteration steps of BiCGSTAB followed by another segmental iteration steps of GPBiCG during the whole iterative procedure. The resulting GPBiCG(m, ℓ)

method is clearly a hybrid variant of GPBiCG and BiCGSTAB and is expected to enjoy both the low computational cost of BiCGSTAB and at the same time the good convergence of GPBiCG, refer to [40,41]. From [40], the original version of GPBiCG(m, ℓ) can be written as Algorithm 1.

Algorithm 1 Algorithm of the GPBiCG(m, ℓ) method

1: Select initial guess $\boldsymbol{x}_0$ and compute $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$
2: Choose $\boldsymbol{r}_0^* = \boldsymbol{r}_0$ such that $\langle \boldsymbol{r}_0^*, \boldsymbol{r}_0 \rangle \neq 0$. Set $\boldsymbol{t}_{-1} = \boldsymbol{w}_{-1} = \boldsymbol{0}$, $\beta_{-1} = 0$.
3: **for** $n = 0, 1, \ldots$, until convergence **do**
4: $\quad \boldsymbol{p}_n = \boldsymbol{r}_n + \beta_{n-1}(\boldsymbol{p}_{n-1} - \boldsymbol{u}_{n-1})$, $\boldsymbol{q}_n = A\boldsymbol{p}_n$
5: $\quad \alpha_n = \frac{\langle \boldsymbol{r}_0^*, \boldsymbol{r}_n \rangle}{\langle \boldsymbol{r}_0^*, \boldsymbol{q}_n \rangle}$
6: $\quad \boldsymbol{t}_n = \boldsymbol{r}_n - \alpha_n \boldsymbol{q}_n$, $\boldsymbol{s}_n = A\boldsymbol{t}_n$
7: $\quad \boldsymbol{y}_n = \boldsymbol{t}_{n-1} - \boldsymbol{t}_n - \alpha_n \boldsymbol{w}_{n-1}$
8: $\quad$ **if** (mod$(n, m + \ell) < m$ or $n = 0$) **then**
9: $\quad\quad \zeta_n = \frac{\langle \boldsymbol{s}_n, \boldsymbol{t}_n \rangle}{\langle \boldsymbol{s}_n, \boldsymbol{s}_n \rangle}$ (Hint: $\eta_n = 0$)
10: $\quad\quad \boldsymbol{u}_n = \zeta_n \boldsymbol{q}_n$, $\boldsymbol{z}_n = \zeta_n \boldsymbol{r}_n - \alpha_n \boldsymbol{u}_n$
11: $\quad\quad \boldsymbol{r}_{n+1} = \boldsymbol{t}_n - \zeta_n \boldsymbol{s}_n$
12: $\quad$ **else**
13: $\quad\quad \zeta_n = \frac{\langle \boldsymbol{y}_n, \boldsymbol{y}_n \rangle \langle \boldsymbol{s}_n, \boldsymbol{t}_n \rangle - \langle \boldsymbol{y}_n, \boldsymbol{t}_n \rangle \langle \boldsymbol{s}_n, \boldsymbol{y}_n \rangle}{\langle \boldsymbol{s}_n, \boldsymbol{s}_n \rangle \langle \boldsymbol{y}_n, \boldsymbol{y}_n \rangle - \langle \boldsymbol{y}_n, \boldsymbol{s}_n \rangle \langle \boldsymbol{s}_n, \boldsymbol{y}_n \rangle}$
14: $\quad\quad \eta_n = \frac{\langle \boldsymbol{s}_n, \boldsymbol{s}_n \rangle \langle \boldsymbol{y}_n, \boldsymbol{t}_n \rangle - \langle \boldsymbol{y}_n, \boldsymbol{s}_n \rangle \langle \boldsymbol{s}_n, \boldsymbol{t}_n \rangle}{\langle \boldsymbol{s}_n, \boldsymbol{s}_n \rangle \langle \boldsymbol{y}_n, \boldsymbol{y}_n \rangle - \langle \boldsymbol{y}_n, \boldsymbol{s}_n \rangle \langle \boldsymbol{s}_n, \boldsymbol{y}_n \rangle}$
15: $\quad\quad \boldsymbol{u}_n = \zeta_n \boldsymbol{q}_n + \eta_n(\boldsymbol{t}_{n-1} - \boldsymbol{r}_n + \beta_{n-1}\boldsymbol{u}_{n-1})$
16: $\quad\quad \boldsymbol{z}_n = \zeta_n \boldsymbol{r}_n + \eta_n \boldsymbol{z}_{n-1} - \alpha_n \boldsymbol{u}_n$
17: $\quad\quad \boldsymbol{r}_{n+1} = \boldsymbol{t}_n - \eta_n \boldsymbol{y}_n - \zeta_n \boldsymbol{s}_n$
18: $\quad$ **end if**
19: $\quad \beta_n = \frac{\alpha_n}{\zeta_n} \cdot \frac{\langle \boldsymbol{r}_0^*, \boldsymbol{r}_{n+1} \rangle}{\langle \boldsymbol{r}_0^*, \boldsymbol{r}_n \rangle}$
20: $\quad \boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n \boldsymbol{p}_n + \boldsymbol{z}_n$
21: $\quad \boldsymbol{w}_n = \boldsymbol{s}_n + \beta_n \boldsymbol{q}_n$
22: **end for**

In addition, the GPBiCG(1, 0), GPBiCG(0, 1) and GPBiCG(1, 1) methods are equivalent to the BiCGSTAB, GPBiCG and BiCGSTAB2 methods, respectively; see [40] for details. Therefore, the GPBiCG(m, ℓ) method can be viewed as a direct extension of the BiCGSTAB2 method. The two parameters η_n and ζ_n at lines 13 and 14 are computed to minimize the residual norm $\|\boldsymbol{r}_{n+1}\|_2 = \|\boldsymbol{t}_n - \eta_n \boldsymbol{y}_n - \zeta_n A\boldsymbol{t}_n\|_2$, also see Eq. (4). The selection of m and ℓ in the GPBiCG(m, ℓ) methods is in general problem dependent.

2.2. A new variant of the GPBiCOR method

According to the derivation of GPBiCG, it is not hard to find that the GPBiCOR method can be also obtained by utilizing the aforementioned framework (2)–(4), by employing the identities $\boldsymbol{r}_k^{\mathrm{BiCOR}} = R_k(A)\boldsymbol{r}_0$ and defining the residual vector $\boldsymbol{r}_k$ of the hybrid BiCOR methods as $M_k(A)\boldsymbol{r}_k^{\mathrm{BiCOR}}$, where the residual vector $\boldsymbol{r}_k^{\mathrm{BiCOR}}$ generated by BiCOR; refer to [24,25] for details. Similarly to the GPBiCG method, the implementation of minimization problem (4) is slightly more expensive (see the computational formulae of the scalar coefficients ζ_n and η_n in Algorithm 2) than other hybrid methods such as CORS and BiCORSTAB. In order to remedy this difficulty, we exploit Fujino's idea implemented in GPBiCG(m, ℓ) and we derive the hybridized variant of the GPBiCOR method. We refer to the resulting method as dubbed GPBiCOR(m, ℓ). Firstly, we reformulate the algorithm of GPBiCOR proposed in [33] as in Algorithm 2.

Algorithm 2 Algorithm of GPBiCOR

1: Select initial guess $\boldsymbol{x}_0$ and compute $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$
2: Choose $\boldsymbol{r}_0^* = A\boldsymbol{r}_0$ such that $\langle \boldsymbol{r}_0^*, A\boldsymbol{r}_0 \rangle \neq 0$. Set $\boldsymbol{t}_{-1} = \boldsymbol{w}_{-1} = \boldsymbol{0}$, $\beta_{-1} = 0$.
3: **for** $n = 0, 1, \ldots$, until convergence **do**
4: $\quad \boldsymbol{p}_n = \boldsymbol{r}_n + \beta_{n-1}(\boldsymbol{p}_{n-1} - \boldsymbol{u}_{n-1})$
5: $\quad \check{\boldsymbol{p}}_n = \check{\boldsymbol{r}}_n + \beta_{n-1}(\check{\boldsymbol{p}}_{n-1} - \check{\boldsymbol{u}}_{n-1})$,
6: $\quad$ Compute $\boldsymbol{f}_n = A\check{\boldsymbol{p}}_n$ and then $\alpha_n = \frac{\langle \boldsymbol{r}_0^*, \check{\boldsymbol{r}}_n \rangle}{\langle \boldsymbol{r}_0^*, \boldsymbol{f}_n \rangle}$
7: $\quad \boldsymbol{t}_n = \boldsymbol{r}_n - \alpha_n \check{\boldsymbol{p}}_n$
8: $\quad \boldsymbol{y}_n = \boldsymbol{t}_{n-1} - \boldsymbol{t}_n - \alpha_n \boldsymbol{w}_{n-1}$
9: $\quad \boldsymbol{s}_n = \check{\boldsymbol{r}}_n - \alpha_n \boldsymbol{f}_n$, $\quad (\triangleright\ \boldsymbol{s}_n \triangleq A\boldsymbol{t}_n)$
10: $\quad \zeta_n = \frac{\langle \boldsymbol{y}_n, \boldsymbol{y}_n \rangle \langle \boldsymbol{s}_n, \boldsymbol{t}_n \rangle - \langle \boldsymbol{y}_n, \boldsymbol{t}_n \rangle \langle \boldsymbol{s}_n, \boldsymbol{y}_n \rangle}{\langle \boldsymbol{s}_n, \boldsymbol{s}_n \rangle \langle \boldsymbol{y}_n, \boldsymbol{y}_n \rangle - \langle \boldsymbol{y}_n, \boldsymbol{s}_n \rangle \langle \boldsymbol{s}_n, \boldsymbol{y}_n \rangle}$
11: $\quad \eta_n = \frac{\langle \boldsymbol{s}_n, \boldsymbol{s}_n \rangle \langle \boldsymbol{y}_n, \boldsymbol{t}_n \rangle - \langle \boldsymbol{y}_n, \boldsymbol{s}_n \rangle \langle \boldsymbol{s}_n, \boldsymbol{t}_n \rangle}{\langle \boldsymbol{s}_n, \boldsymbol{s}_n \rangle \langle \boldsymbol{y}_n, \boldsymbol{y}_n \rangle - \langle \boldsymbol{y}_n, \boldsymbol{s}_n \rangle \langle \boldsymbol{s}_n, \boldsymbol{y}_n \rangle}$
12: $\quad$ **if** $n = 0$ **then**
13: $\quad\quad \zeta_n = \frac{\langle \boldsymbol{s}_n, \boldsymbol{t}_n \rangle}{\langle \boldsymbol{s}_n, \boldsymbol{s}_n \rangle}$, $\eta_n = 0$
14: $\quad$ **end if**
15: $\quad \boldsymbol{u}_n = \zeta_n \check{\boldsymbol{p}}_n + \eta_n(\boldsymbol{t}_{n-1} - \boldsymbol{r}_n + \beta_{n-1}\boldsymbol{u}_{n-1})$
16: $\quad \check{\boldsymbol{u}}_n = \zeta_n \boldsymbol{f}_n + \eta_n(\boldsymbol{s}_{n-1} - \check{\boldsymbol{r}}_n + \beta_{n-1}\check{\boldsymbol{u}}_{n-1})$
17: $\quad \boldsymbol{z}_n = \zeta_n \boldsymbol{r}_n + \eta_n \boldsymbol{z}_{n-1} - \alpha_n \boldsymbol{u}_n$
18: $\quad \boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n \boldsymbol{p}_n + \boldsymbol{z}_n$
19: $\quad \boldsymbol{r}_{n+1} = \boldsymbol{t}_n - \eta_n \boldsymbol{y}_n - \zeta_n \boldsymbol{s}_n$
20: $\quad$ Compute $\check{\boldsymbol{r}}_{n+1} = A\boldsymbol{r}_{n+1}$
21: $\quad \beta_n = \frac{\alpha_n}{\zeta_n} \cdot \frac{\langle \boldsymbol{r}_0^*, \check{\boldsymbol{r}}_{n+1} \rangle}{\langle \boldsymbol{r}_0^*, \check{\boldsymbol{r}}_n \rangle}$, $\quad \boldsymbol{w}_n = \boldsymbol{s}_n + \beta_n \check{\boldsymbol{p}}_n$
22: **end for**

Here, two recurrences and the variables characterized with symbol "ˇ" at steps 5, 9, and 16 are newly added to alleviate the number of matrix–vector products per iteration step. If we want to obtain the algorithm of BiCORSTAB, we would need to simplify four working variables $\boldsymbol{y}_n$, $\boldsymbol{u}_n$, $\boldsymbol{z}_n$ and $\boldsymbol{w}_n$ given above. In Table 1 we summarize the choices of the two parameters η_n and ζ_n leading to the CORS, BiCORSTAB, GPBiCOR, BiCORSTAB2 and the proposed GPBiCOR(m, ℓ) methods. From this table we can see that the BiCORSTAB, GPBiCOR and BiCORSTAB2 methods correspond to GPBiCOR(1, 0), GPBiCOR(0, 1) and GPBiCOR(1, 1) methods, respectively. In other words, the BiCORSTAB2 method will be extended to the GPBiCOR(m, ℓ)

Table 2
Computation cost of GPBiCG(m, ℓ), GPBiCOR(m, ℓ) and their preconditioned variants.

Method	Vector update				Inner product				MVPs	Preconditioner solve
	H	**if**	**else**	T	H	**if**	**else**	T		
GPBiCG(m, ℓ)	5	3	7	3	1	2	5	1	2	0
GPBiCOR(m, ℓ)	8	4	10	3	1	2	5	1	2	0
PGPBiCG(m, ℓ)	6	3	7	3	1	2	5	1	2	2
PGPBiCOR(m, ℓ)	9	4	10	3	1	2	5	1	2	2

method by this study. Like the GPBiCG(m, ℓ) method, the two parameters η_n and ζ_n are determined so that the residual norm $\|\boldsymbol{r}_{n+1}\|_2 = \|\boldsymbol{t}_n - \eta_n \boldsymbol{y}_n - \zeta_n A\boldsymbol{t}_n\|_2$ is minimized, also see Eq. (4). Combination of both parameters m and ℓ of the GPBiCOR(m, ℓ) method can be decided according to the difficulty of the problem.

For the efficient implementation, we present the preconditioned version of the GPBiCOR(m, ℓ) algorithm (see Algorithm 2[1]) by the derivation of preconditioned GPBiCG method [25]. The pseudo code, here rewritten using the same style of Algorithm 1, reads as:

Algorithm 3 Algorithm of preconditioned GPBiCOR(m, ℓ) (K is the preconditioner)

Select initial guess $\boldsymbol{x}_0$ and $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$
Choose $\boldsymbol{r}_0^* = AK^{-1}\boldsymbol{r}_0$ s.t. $\langle \boldsymbol{r}_0^*, AK^{-1}\boldsymbol{r}_0\rangle \neq 0$. Set $\boldsymbol{t}_{-1} = \boldsymbol{w}_{-1} = \boldsymbol{u}_{-1} = \hat{\boldsymbol{u}}_{-1} = \boldsymbol{0}$, $\beta_{-1} = 0$, $\boldsymbol{e}_0 = K^{-1}\boldsymbol{r}_0$, $\hat{\boldsymbol{e}}_0 = A\boldsymbol{e}_0$.
for $n = 0, 1, \ldots,$ until convergence **do**
  $\boldsymbol{p}_n = \boldsymbol{e}_n + \beta_{n-1}(\boldsymbol{p}_{n-1} - \boldsymbol{u}_{n-1})$
  $\hat{\boldsymbol{p}}_n = \hat{\boldsymbol{e}}_n + \beta_{n-1}(\hat{\boldsymbol{p}}_{n-1} - \hat{\boldsymbol{u}}_{n-1})$
  Solve $K\boldsymbol{h}_n = \hat{\boldsymbol{p}}_n$
  Compute $\boldsymbol{g}_n = A\boldsymbol{h}_n$ and then $\alpha_n = \frac{\langle \boldsymbol{r}_0^*, \hat{\boldsymbol{e}}_n\rangle}{\langle \boldsymbol{r}_0^*, \boldsymbol{g}_n\rangle}$
  $\boldsymbol{t}_n = \boldsymbol{r}_n - \alpha_n \hat{\boldsymbol{p}}_n$
  $\boldsymbol{y}_n = \boldsymbol{t}_{n-1} - \boldsymbol{t}_n - \alpha_n \boldsymbol{q}_{n-1}$
  $\boldsymbol{s}_n = \boldsymbol{e}_n - \alpha_n \boldsymbol{h}_n$ ($\triangleright$ $\boldsymbol{s}_n \triangleq K^{-1}\boldsymbol{t}_n$)
  $\boldsymbol{w}_n = \hat{\boldsymbol{e}}_n - \alpha_n \boldsymbol{g}_n$ ($\triangleright$ $\boldsymbol{w}_n \triangleq A\boldsymbol{s}_n$)
  if $(\mathrm{mod}(n, m+\ell) < m$ or $n = 0)$ **then**
    $\zeta_n = \frac{\langle \boldsymbol{w}_n, \boldsymbol{t}_n\rangle}{\langle \boldsymbol{w}_n, \boldsymbol{w}_n\rangle}$ (Hint : $\eta_n = 0$)
    $\boldsymbol{u}_n = \zeta_n \boldsymbol{h}_n$, $\hat{\boldsymbol{u}}_n = \zeta_n \boldsymbol{g}_n$
    $\boldsymbol{z}_n = \zeta_n \boldsymbol{e}_n - \alpha_n \boldsymbol{u}_n$
    $\boldsymbol{r}_{n+1} = \boldsymbol{t}_n - \zeta_n \boldsymbol{w}_n$
  else
    $\zeta_n = \frac{\langle \boldsymbol{y}_n, \boldsymbol{y}_n\rangle\langle \boldsymbol{w}_n, \boldsymbol{t}_n\rangle - \langle \boldsymbol{y}_n, \boldsymbol{t}_n\rangle\langle \boldsymbol{w}_n, \boldsymbol{y}_n\rangle}{\langle \boldsymbol{w}_n, \boldsymbol{w}_n\rangle\langle \boldsymbol{y}_n, \boldsymbol{y}_n\rangle - \langle \boldsymbol{y}_n, \boldsymbol{w}_n\rangle\langle \boldsymbol{w}_n, \boldsymbol{y}_n\rangle}$
    $\eta_n = \frac{\langle \boldsymbol{w}_n, \boldsymbol{w}_n\rangle\langle \boldsymbol{y}_n, \boldsymbol{t}_n\rangle - \langle \boldsymbol{y}_n, \boldsymbol{w}_n\rangle\langle \boldsymbol{w}_n, \boldsymbol{t}_n\rangle}{\langle \boldsymbol{w}_n, \boldsymbol{w}_n\rangle\langle \boldsymbol{y}_n, \boldsymbol{y}_n\rangle - \langle \boldsymbol{y}_n, \boldsymbol{w}_n\rangle\langle \boldsymbol{w}_n, \boldsymbol{y}_n\rangle}$
    $\boldsymbol{u}_n = \zeta_n \boldsymbol{h}_n + \eta_n(\boldsymbol{s}_{n-1} - \boldsymbol{e}_n + \beta_{n-1}\boldsymbol{u}_{n-1})$
    $\hat{\boldsymbol{u}}_n = \zeta_n \boldsymbol{g}_n + \eta_n(\boldsymbol{w}_{n-1} - \hat{\boldsymbol{e}}_n + \beta_{n-1}\hat{\boldsymbol{u}}_{n-1})$
    $\boldsymbol{z}_n = \zeta_n \boldsymbol{e}_n + \eta_n \boldsymbol{z}_{n-1} - \alpha_n \boldsymbol{u}_n$
    $\boldsymbol{r}_{n+1} = \boldsymbol{t}_n - \eta_n \boldsymbol{y}_n - \zeta_n \boldsymbol{w}_n$
  end if
  Solve $K\boldsymbol{e}_{n+1} = \boldsymbol{r}_{n+1}$
  Compute $\hat{\boldsymbol{e}}_{n+1} = A\boldsymbol{e}_{n+1}$
  $\beta_n = \frac{\alpha_n}{\zeta_n} \cdot \frac{\langle \boldsymbol{r}_0^*, \hat{\boldsymbol{e}}_{n+1}\rangle}{\langle \boldsymbol{r}_0^*, \hat{\boldsymbol{e}}_n\rangle}$
  $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n \boldsymbol{p}_n + \boldsymbol{z}_n$
  $\boldsymbol{q}_n = \boldsymbol{w}_n + \beta_n \hat{\boldsymbol{p}}_n$
end for

At the end of this section, we analyze the computational cost for GPBiCG(m, ℓ), GPBiCOR(m, ℓ) and their preconditioned variants (abbreviated as PGPBiCG(m, ℓ) and PGPBiCOR(m, ℓ), respectively) in Table 1 by using the idea of Ref. [42]. The **for** loops in Algorithms 1 and 3 are divided into 4 parts: the part before **if** and after **else** denoted as the head (H) part and the tail (T) respectively. As seen from Table 2, the computational cost of the GPBiCOR(m, ℓ) and PGPBiCOR(m, ℓ) methods is slightly more expensive than the GPBiCG(m, ℓ) and PGPBiCG(m, ℓ) methods, respectively, if the number of iterations of the GPBiCOR(m, ℓ) and PGPBiCOR(m, ℓ) methods is greatly close to that of the GPBiCG(m, ℓ) and PGPBiCG(m, ℓ) methods. However, the GPBiCOR(m, ℓ) and PGPBiCOR(m, ℓ) methods require less number of iteration steps than the GPBiCG(m, ℓ) and PGPBiCG(m, ℓ) methods in some cases, respectively. Then it implies that the GPBiCOR(m, ℓ) and PGPBiCOR(m, ℓ) methods can still converge faster than the GPBiCG(m, ℓ) and PGPBiCG(m, ℓ) methods in terms of the elapsed CPU time for some test problems, respectively; also refer to Section 3 for further discussion.

3. Numerical experiments

Far from being exhaustive, in this section the following experiments are reported to investigate some performance of the GPBiCOR(m, ℓ) methods when applied to solving five sets of test problems. For the sake of comparison, we are concerned with the GPBiCG(m, ℓ), BiCORSTAB, GPBiCOR, BiCORSTAB2, CORS, and other popular Krylov subspace methods (such as CGS, GMRES(50), IDR(s),[2] and TFQMR). Here we also need to specify that the BiCGSTAB, GPBiCG and BiCGSTAB2 methods correspond to the GPBiCG(1, 0), GPBiCG(0, 1) and GPBiCG(1, 1) methods, respectively, refer to [40] for details. Meanwhile, our numerical experiments focus on both real and complex test matrices, and also consider the preconditioning benefits of

[1] In order to reduce the number of *MVPS*, some notations should be defined: $\hat{\boldsymbol{p}}_n := A\boldsymbol{p}_n$, $\hat{\boldsymbol{e}}_n := A\boldsymbol{e}_n$, $\hat{\boldsymbol{u}}_{n-1} := A\boldsymbol{u}_{n-1}$. Then it shows that Algorithm 2 (like the GPBiCG(m, ℓ) method) needs two *MVPS* in each iteration step.

[2] We directly use the MATLAB codes from http://ta.twi.tudelft.nl/nw/users/gijzen/IDR.html.

Table 3
Set and characteristics of test matrices in Section 3 (listed in increasing matrix size).

Matrix problem	Size	Field	$nnz(A)$
circuit_1	2,624	Circuit simulation problem	35,823
ex31	3,909	Computational fluid dynamics	91,223
circuit_2	4,510	Circuit simulation problem	21,199
sherman3	5,005	Computational fluid dynamics	20,033
Poisson3Da	13,514	Computational fluid dynamics	352,762
memplus	17,758	Circuit simulation problem	99,147
bcircuit	68,902	Circuit simulation problem	375,558
hcircuit	105,676	Circuit simulation problem	513,072

iterative solvers. The experiments have been carried out in double precision floating point arithmetic with MATLAB 2014a (64 bit) on PC-Intel(R) Core(TM) i5-3470 CPU 3.20 GHz, 8 GB of RAM. Here we use *Iters* and *TRR* to denote the number of iterations and $\log_{10}$ of the final true relative residual 2-norm defined as $\log_{10}(\|\boldsymbol{b} - A\boldsymbol{x}_{\text{final}}\|_2/\|\boldsymbol{r}_0\|_2)$, respectively. In our experiments we take the zero vector as initial guess. The stopping criterion used here is the reduction of the 2-norm of the initial residual by eight orders of magnitude, i.e., $\|\boldsymbol{r}_k\|_2/\|\boldsymbol{r}_0\|_2 \leq \text{Tol} = 10^{-8}$. The convergence histories show *MVPS* (on the horizontal axis) versus 2-norms of the approximate relative residuals (on the vertical axis) in all figures. The notations "OSTAB", "STAB", "GPO" and "GP" correspond to the BiCORSTAB, BiCGSTAB, GPBiCOR and GPBiCG methods, respectively. Similarly, the symbols "GPO$(\cdot,\cdot)$" and "GP$(\cdot,\cdot)$" denote each sub-solvers of the GPBiCOR(m,ℓ) and GPBiCG(m,ℓ) methods, respectively. For the sake of clarity, the non-Hermitian test matrices from University of Florida Sparse Matrix Collection [43] in this section are listed in Table 3.

Example 1. First, we consider two publicly available linear systems with the two coefficient matrices "circuit_1" and "circuit_2", which arise from circuit simulations modeled as a system of differential algebraic equations. Upon solving with a BDF method, and Newton's method, a linear system $A\boldsymbol{x} = \boldsymbol{b}$ is required to be solved at each time step. The numerical results are reported in Table 4. Here, the symbol "†" indicates that the method did not converge, and this specified symbol is also available for similar cases in the next numerical examples.

We can see that the GPBiCOR(m,ℓ) class of methods are more efficient than the GPBiCG(m,ℓ) solvers in terms of the less number of *Iters*, except the GPBiCOR$(2,1)$ method for the first problem. Meanwhile, the level of accuracy of all the converged approximate solutions (in terms of *TRR*) was almost of the same order of magnitude of the tolerance value used in the given stopping criterion, except the CGS and TFQMR methods for the second test problem. We found that when the number of *Iters* for these two families of iterative solvers are similar, the GPBiCOR(m,ℓ) methods may be slightly more expensive than the GPBiCG(m,ℓ) methods. It is because the GPBiCOR(m,ℓ) methods require some additional BLAS-1 operations in each iteration step, see the lines 5, 10, 13 and 20 of Algorithm 3 and also Table 3. Both the GPBiCG(m,ℓ) and GPBiCOR(m,ℓ) methods indeed improve the convergence of the BiCGSTAB and BiCORSTAB methods in terms of both the number of *Iters* and elapsed CPU time. Roughly, the GPBiCG(m,ℓ) method also alleviates the elapsed CPU time of GPBiCG, but the GPBiCOR(m,ℓ) method fails to reduce the elapsed CPU time of GPBiCOR dramatically in Example 1. Meanwhile, although the restarted GMRES method is close to the *optimal* Krylov subspace methods, it is still more expensive than the other iterative solvers. For simplicity, Fig. 1 only shows typical plots of the convergence behavior of different iterative solvers; the GPBiCOR(m,ℓ) methods showed fairly attractive convergence behaviors compared to those of GPBiCG(m,ℓ) in the last phase. In conclusion, the GPBiCOR$(1,3)$ and GPBiCOR$(1,2)$ methods are the best choices for the number of *Iters* and elapsed CPU time for solving the first and second problems, respectively. From the point of view of solution time, the CORS method can be used as an alternative for the first test problem, and the IDR(4) and IDR(6) methods can be regarded as two good alternatives for the second test problem. By the way, the results for both CGS and TFQMR methods listed in Table 4 indicate that these two methods may suffer from serious breakdowns on the circuit_2 problem. In the interest of counteraction against such breakdowns, oneself can make use of remedies such as the so-called look-ahead strategies [44] which can enhance stability while increasing cost modestly or others. Since this is outside the scope of this study, we shall not pursue that here.

Example 2. Here we choose other two linear systems from University of Florida Sparse Matrix Collection, arising from computational fluid dynamics (CFD) applications. More precisely, the nonsymmetric matrix "sherman3" comes from the black oil model simulations. Test matrix "Poisson3Da" from FEMLAB, which is a finite-element method toolbox for MATLAB, developed by Comsol, Inc. (see http://www.femlab.com for more details). We let $\boldsymbol{b} = A\boldsymbol{e}$, where $\boldsymbol{e}$ was chosen as a vector with random entries from -1 to 1, such that $\boldsymbol{e}$ is the exact solution. The numerical results of different iterative solvers are reported in Table 5. A symbol "*max*" is used to indicate that the method did not meet the required *Tol* before $MAXIT = 5000$.

As seen from Table 5, since the number of successful sub-solvers of the GPBiCOR(m,ℓ) methods for solving the first test problem is higher compared to GPBiCG(m,ℓ), we conclude that the GPBiCOR(m,ℓ) methods are more efficient than the GPBiCG(m,ℓ) methods on this problem. Moreover, the GPBiCG(m,ℓ) method does not improve the convergence of the BiCGSTAB and GPBiCG methods at all, whereas the GPBiCOR(m,ℓ) method improves the convergence of BiCORSTAB and partially the convergence of GPBiCOR in the first test problem. For the second test problem, the GPBiCG(m,ℓ) method

Table 4
Numerical results of different iterative solvers for Example 1.

Method	circuit_1			circuit_2		
	Iters	*TRR*	Time (s)	*Iters*	*TRR*	Time (s)
OSTAB/STAB	498/824	−8.0026/−8.0161	0.1640/0.2197	342/390	−8.0325/−8.1821	0.1467/0.1526
GPO/GP	136/170	−8.2221/−8.0101	0.0748/0.0964	295/339	−8.2942/−8.0677	0.1513/0.1659
GPO(1, 1)/GP(1, 1)	158/189	−8.0588/−8.0196	0.0810/0.0872	268/320	−8.1521/−8.1293	0.1411/0.1487
GPO(1, 2)/GP(1, 2)	132/163	−8.2126/−8.0026	0.0741/0.0814	**257**/319	−8.0200/−8.1943	**0.1317**/0.1509
GPO(1, 3)/GP(1, 3)	**120**/180	−8.0791/−8.0398	**0.0687**/0.0833	294/346	−8.0071/−8.1511	0.1574/0.1634
GPO(1, 4)/GP(1, 4)	136/181	−8.0612/−8.0073	0.0784/0.0816	288/321	−8.0119/−8.0542	0.1505/0.1545
GPO(1, 5)/GP(1, 5)	182/171	−8.0263/−8.1147	0.0784/0.0816	301/325	−8.0253/−8.0387	0.1633/0.1610
GPO(2, 1)/GP(2, 1)	214/202	−8.0108/−8.0410	0.0967/0.0887	295/325	−8.0388/−8.0348	0.1508/0.1491
GPO(3, 1)/GP(3, 1)	208/245	−8.0795/−8.0033	0.0941/0.0992	329/334	−8.2955/−8.0477	0.1605/0.1498
GPO(4, 1)/GP(4, 1)	182/219	−8.0026/−8.0096	0.0807/0.0914	288/380	−8.3542/−8.0066	0.1549/0.1544
GPO(5, 1)/GP(5, 1)	204/240	−8.0204/−8.0712	0.0877/0.0895	310/317	−8.0981/−8.0206	0.1587/0.1441
CORS/CGS	151/137	−8.1720/−8.2129	0.0698/0.0898	469/†	−8.0914/†	0.1467/†
IDR(2)	622	−8.0476	0.1607	645	−8.3012	0.1629
IDR(4)	379	−8.1426	0.1297	366	−8.1850	0.1369
IDR(6)	623	−8.0722	0.1844	325	−8.1151	0.1383
IDR(8)	268	−8.0670	0.1146	306	−8.4450	0.1414
TFQMR	131	−8.0051	0.1098	†	†	†
GMRES(50)	139	−8.0670	0.2327	270	−8.0127	0.4994

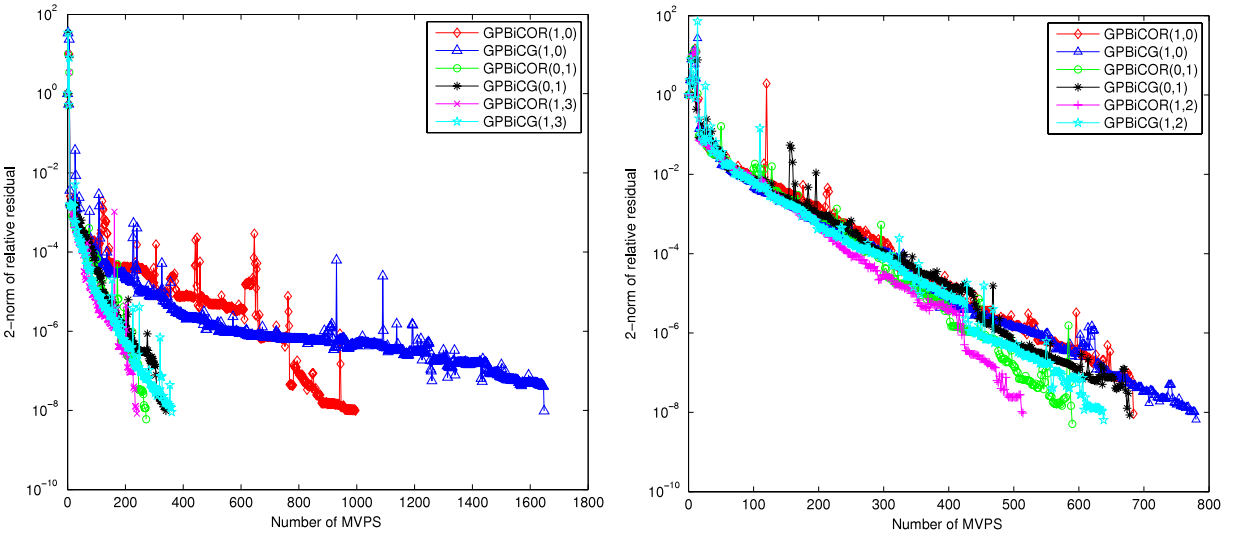

Fig. 1. Convergence histories of different iterative methods for Example 1. Left: `circuit_1`; Right: `circuit_2`.

indeed improves the convergence of the BiCGSTAB method in terms of both number of *Iters* and elapsed CPU time. Roughly, the GPBiCG(m, ℓ) method also alleviates the elapsed CPU time of GPBiCG, but the GPBiCOR(m, ℓ) method fails to improve the performance of the BiCORSTAB and GPBiCOR methods dramatically. Meanwhile, the level of accuracy of all the converged approximate solutions (in terms of *TRR*) was almost of the same order of magnitude of the tolerance value used in the given stopping criterion. At the same time, when the number of *Iters* required to converge is similar, the GPBiCOR(m, ℓ) methods may be slightly more expensive than the GPBiCG(m, ℓ) methods. Once again, although the restarted GMRES method gets close to the *optimal* Krylov subspace methods, it is still more expensive than the other iterative solvers. From Table 5 we can conclude that the GPBiCOR(5, 1) method is the best choice among these listed iterative solvers in terms of the less number of *Iters* and elapsed CPU time for the first test problem. Even the TRR of the GPBiCOR(5, 1) method is the best one among these iterative solvers for the first test problem. The BiCORSTAB method is the fastest solver on the second test problem. In addition, the performance of both the GPBiCOR(3, 1) and GPBiCOR(4, 1) methods are not disappointing and they can be considered as two alternatives for this problem. By the way, the results of testing the first problem also indicate that application specific preconditioners may be mandatory to use to speed up the iterative procedure.

Example 3. In this example, we solve the test problems adopted from [45], for which the coefficient matrix of the linear system has the form

$$A = \begin{pmatrix} D & K \\ -K^T & \mu I \end{pmatrix} \in \mathbb{R}^{2n\times 2n}, \quad K = (t_{i,j}) = \frac{1}{\sqrt{2\pi}}\sigma e^{\frac{-|i-j|^2}{2\sigma^2}} \text{ is a Toeplitz matrix,}$$

Table 5
Numerical results of different iterative solvers for Example 2.

Method	sherman3			Poisson3Da		
	Iters	*TRR*	Time (s)	*Iters*	*TRR*	Time (s)
OSTAB/STAB	*max*/*max*	−7.1836/−5.8986	1.6602/1.3644	**81**/93	−8.1803/−8.1077	**0.2102**/0.2402
GPO/GP	4357/*max*	−8.1954/−6.2042	1.8392/1.7978	82/89	−8.1629/−8.1142	0.2267/0.2377
GPO(1, 1)/GP(1, 1)	4611/*max*	−8,1446/−5.4949	1.7633/1.5874	95/86	−8.2449/−8.0148	0.2538/0.2276
GPO(1, 2)/GP(1, 2)	*max*/*max*	−7.6632/−4.8486	1.9452/1.6455	81/86	−8.2744/−8.1585	0.2201/0.2287
GPO(1, 3)/GP(1, 3)	*max*/*max*	−4.9216/−6.2510	1.9827/1.6811	93/88	−8.4584/−8.4727	0.2507/0.2371
GPO(1, 4)/GP(1, 4)	4206/*max*	−8.0509/−6.3787	1.6896/1.6980	91/87	−8.0041/−8.4751	0.2479/0.2329
GPO(1, 5)/GP(1, 5)	4829/*max*	−8.0100/−6.4886	1.9849/1.7083	96/87	−8.0376/−8.2911	0.2647/0.2487
GPO(2, 1)/GP(2, 1)	4317/*max*	−8.1330/−5.2159	1.5649/1.5291	92/86	−8.2067/−8.0555	0.2414/0.2292
GPO(3, 1)/GP(3, 1)	4127/*max*	−8.0615/−7.0136	1.4605/1.5043	81/87	−8.3735/−8.4569	0.2118/0.2331
GPO(4, 1)/GP(4, 1)	4135/*max*	−8.1391/−5.7639	1.4462/1.4712	81/87	−8.3015/−8.2211	0.2108/0.2265
GPO(5, 1)/GP(5, 1)	**4034**/*max*	−8.2421/−6.8179	**1.3598**/1.4609	92/86	−8.1581/−8.0401	0.2399/0.2198
CORS/CGS	4940/†	−8.0780/†	1.4946/†	96/99	−8.0509/−8.2469	0.2336/0.2538
IDR(2)	†	†	†	178	−8.2200	0.2814
IDR(4)	†	†	†	173	−8.1231	0.2785
IDR(6)	*max*	−4.3192	1.4080	162	−8.1489	0.3012
IDR(8)	*max*	−5.9427	1.6074	161	−8.0416	0.3411
TFQMR	†	†	†	99	−8.0495	0.3117
GMRES(50)	†	†	†	159	−8.0003	0.7326

Table 6
Numerical results of different iterative solvers for Example 3.

Method	$\mu = 0.04, n = 1024$			$\mu = 0.05, n = 2048$		
	Iters	*TRR*	Time (s)	*Iters*	*TRR*	Time (s)
OSTAB/STAB	577/†	−8.0546/†	1.1474/†	370/312	−8.3614/−8.0636	2.1064/1.7390
GPO/GP	213/299	−8.1234/−8.0187	0.4573/0.6214	125/133	−8.0566/−8.0895	0.7403/0.7773
GPO(1, 1)/GP(1, 1)	217/247	−8.2488/−8.1655	0.4607/0.5114	128/137	−8.0283/−8.1372	0.7531/0.8038
GPO(1, 2)/GP(1, 2)	215/273	−8.0106/−8.1546	0.4590/0.5580	**125**/142	−8.0859/−8.3192	0.7361/0.8306
GPO(1, 3)/GP(1, 3)	214/270	−8.0074/−8.0001	0.4601/0.5522	128/137	−8.0243/−8.0736	0.7597/0.8011
GPO(1, 4)/GP(1, 4)	203/261	−8.2817/−8.0971	0.4305/0.5338	143/144	−8.0213/−8.4762	0.8472/0.8448
GPO(1, 5)/GP(1, 5)	**201**/241	−8.2871/−8.2194	**0.4256**/0.4963	129/140	**−8.7243**/−8.1677	0.7608/0.8235
GPO(2, 1)/GP(2, 1)	221/285	−8.0116/−8.0184	0.4609/0.5671	130/138	−8.4843/−8.0296	0.7487/0.8102
GPO(3, 1)/GP(3, 1)	225/286	−8.1022/−8.0602	0.4630/0.5646	128/157	−8.0722/−8.2626	0.7384/0.8986
GPO(4, 1)/GP(4, 1)	245/320	−8.1949/−8.3759	0.5048/0.6226	135/154	−8.2429/−8.0873	0.7753/0.8742
GPO(5, 1)/GP(5, 1)	277/349	**−8.4751**/−8.0953	0.5767/0.6749	210/173	−8.0567/−8.0402	1.1874/0.9803
CORS/CGS	202/210	−8.0732/−8.0127	0.4258/0.4287	120/135	−8.6516/−8.5601	0.6812/0.7641
IDR(2)	539	−8.0166	0.5609	466	−8.0242	1.3613
IDR(4)	459	−8.2629	0.5468	349	−8.2373	1.0492
IDR(6)	394	−8.0333	0.5027	297	−8.0971	0.9211
IDR(8)	361	−8.0372	0.4775	266	−8.0014	0.8431
TFQMR	209	−8.0145	0.4578	135	−8.0196	0.8058
GMRES(50)	761	−8.0025	1.2231	222	−8.0071	0.9699

where $D \in \mathbb{R}^{n\times n}$ is a positive diagonal matrix,[3] and $I \in \mathbb{R}^{n\times n}$ is a identity matrix. Taking $\sigma = 2$, a very ill-conditioned Toeplitz matrix with rapidly decaying singular values is obtained. This family of linear systems often arises in the weighted Toeplitz regularized least squares computation for image restoration problems, refer to [46] for details. For simplicity, we let $\boldsymbol{b} = A\boldsymbol{e}$, where $\boldsymbol{e}$ was chosen as a vector with random entries from −1 to 1, such that $\boldsymbol{e}$ is the exact solution. The numerical results of different iterative solvers are reported in Table 6.

As seen from Table 6, we find that the GPBiCOR(m, ℓ) methods are more efficient than GPBiCG(m, ℓ) methods in terms of the less number of *Iters* and the elapsed CPU time (except the GPBiCOR(1, 4) method for the second problem). Meanwhile, the level of accuracy of all the converged approximate solutions (in terms of *TRR*) was almost of the same order of magnitude of the tolerance value used in the given stopping criterion. As we already mentioned and explained, when the number of *Iters* are similar, the GPBiCOR(m, ℓ) methods may still be more expensive than the GPBiCG(m, ℓ) methods. Both the GPBiCG(m, ℓ) and GPBiCOR(m, ℓ) methods indeed improve the convergence of the BiCGSTAB and BiCORSTAB methods in terms of the number of *Iters* and elapsed CPU time. Roughly, the GPBiCG(m, ℓ) methods also alleviate the elapsed CPU time of GPBiCG, but the GPBiCOR(m, ℓ) methods fail to reduce the elapsed CPU time of GPBiCOR on the first test problem. Moreover, both the GPBiCOR(m, ℓ) and GPBiCG(m, ℓ) methods fail to reduce the elapsed CPU time of GPBiCOR and GPBiCG for the second test problem in Example 3. Meanwhile, the restarted GMRES method still remains more expensive than the other iterative solvers. For simplicity, Fig. 2 only illustrates typical plots of the convergence behavior of some different iterative solvers;

[3] For the sake of convenience, we employ the MATLAB codes "`d = 3 + [rand(n-400,1); randn(400,1)];`" to generate its diagonal entries.

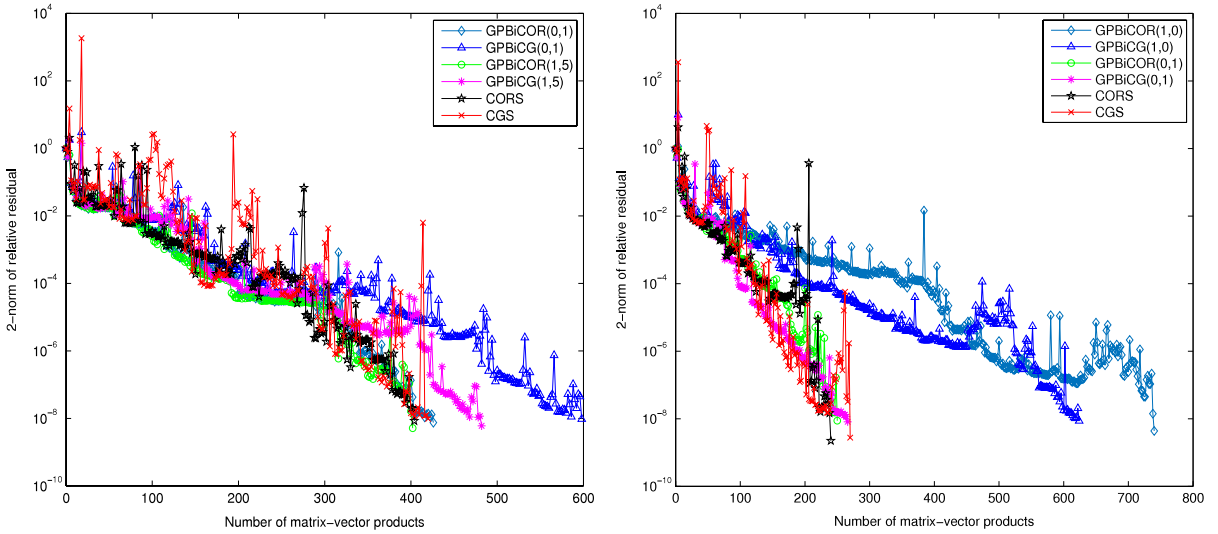

Fig. 2. Convergence histories of the different iterative methods for Example 3. Left: $\mu = 0.04$, $n = 1024$; Right: $\mu = 0.05$, $n = 2048$.

Table 7
Numerical results of different iterative solvers with ILU(0) preconditioning for Example 4.

Method	ex31			memplus		
	Iters	*TRR*	Time (s)	*Iters*	*TRR*	Time (s)
OSTAB/STAB	108/116	−8.2714/−9.1031	0.1207/0.1443	254/189	−8.1232/−8.0034	0.6562/0.4712
GPO/GP	111/139	−8.0746/−8.0504	0.1301/0.1727	142/163	−8.0212/−8.0456	0.4328/0.4606
GPO(1, 1)/GP(1, 1)	100/115	−8.5182/−8.1318	0.1212/0.1456	191/169	−8.0136/−8.0053	0.5537/0.4648
GPO(1, 2)/GP(1, 2)	111/122	−8.2211/−8.0089	0.1338/0.1521	162/169	−8.0427/−8.0229	0.4598/0.4612
GPO(1, 3)/GP(1, 3)	100/127	−8.0471/−8.7276	0.1276/0.1574	139/146	−8.2043/−8.0404	0.4154/0.4285
GPO(1, 4)/GP(1, 4)	111/120	−8.0904/−8.7315	0.1381/0.1529	137/154	−8.0393/−8.0324	0.4084/0.4324
GPO(1, 5)/GP(1, 5)	107/130	−8.4882/−8.3741	0.1356/0.1636	**134**/158	−8.1217/−8.3357	**0.4056**/0.4435
GPO(2, 1)/GP(2, 1)	110/118	−8.5139/−8.0136	0.1240/0.1514	175/165	−8.0078/−8.1747	0.5106/0.4314
GPO(3, 1)/GP(3, 1)	103/117	−8.4296/−8.0549	0.1192/0.1461	221/175	−8.0501/−8.0483	0.6192/0.4608
GPO(4, 1)/GP(4, 1)	102/118	−8.1995/−8.0860	**0.1185**/0.1463	151/171	−8.1220/−8.0007	0.4297/0.4478
GPO(5, 1)/GP(5, 1)	104/117	−8.2862/−8.4862	0.1212/0.1459	188/187	−8.3474/−8.0580	0.5188/0.4946
CORS/CGS	113/135	−8.6617/−8.8585	0.1548/0.1741	152/175	−8.4215/−8.3671	0.4189/0.4841
IDR(2)	137	−8.0958	0.1224	483	−8.0599	0.6627
IDR(4)	130	−8.3549	0.1212	314	−8.0073	0.5464
IDR(6)	116	−8.0992	0.1226	328	−8.0628	0.6576
IDR(8)	115	−8.6055	0.1283	303	−8.0766	0.7054
TFQMR	132	−8.1805	0.1823	180	−8.1355	0.6009
GMRES(50)	215	−8.4268	0.4423	188	−7.3883	1.1046

we can see that GPBiCOR(m, ℓ) methods showed fairly attractive convergence behaviors compared to those of GPBiCG(m, ℓ) methods in the last phase. Note that the CORS and CGS methods show their typically irregular convergence behaviors. For the first test problem, the GPBiCOR(1, 5) method outperforms all other iterative solvers in terms of the number of *Iters* and elapsed CPU time. Meanwhile, the performance of the CORS method is not bad in terms of the number of *Iters* and elapsed CPU time and it can be an efficient alternative for the first test problem. Then the CORS method outperforms the other iterative methods in terms of the less number of *Iters* and elapsed CPU time for solving the second problems. The GPBiCOR(1, 2) method can be used an alternative for this problem. By the way, the results in bold print of Table 6 indicate that the GPBiCOR(5, 1) and GPBiCOR(1, 5) methods can reach the most accurate final solutions in terms of *TRR* in Example 3, respectively.

Example 4. Moreover, we consider two other publicly available linear systems with the coefficient matrices "`ex31`" and "`memplus`", which arise from CFD and circuit simulations, respectively. Whenever no right-hand side is available for the original linear system $A\boldsymbol{x} = \boldsymbol{b}$, we let $\boldsymbol{b} = A\boldsymbol{e}$, where $\boldsymbol{e}$ was chosen as a vector with random entries from -1 to 1, such that $\boldsymbol{e}$ is the exact solution. Then we want to evaluate the performance of GPBiCOR(m, ℓ) and GPBiCG(m, ℓ) with ILU(0) preconditioning [2, pp. 307–310] for solving the test linear systems. For stability reasons we compute an ILU(0) factorization of $A + \sigma I$, where $\sigma = 10^{-12}$ if all diagonal elements of A are zero, or $\sigma = 10^{-12}\max\{|a_{ii}|\}$ if some but not all diagonal elements a_{ii} of A zero, or $\sigma = 0$ otherwise. This procedure follows the recommendations proposed in [47]. The numerical results of different preconditioned iterative solvers are reported in Table 7.

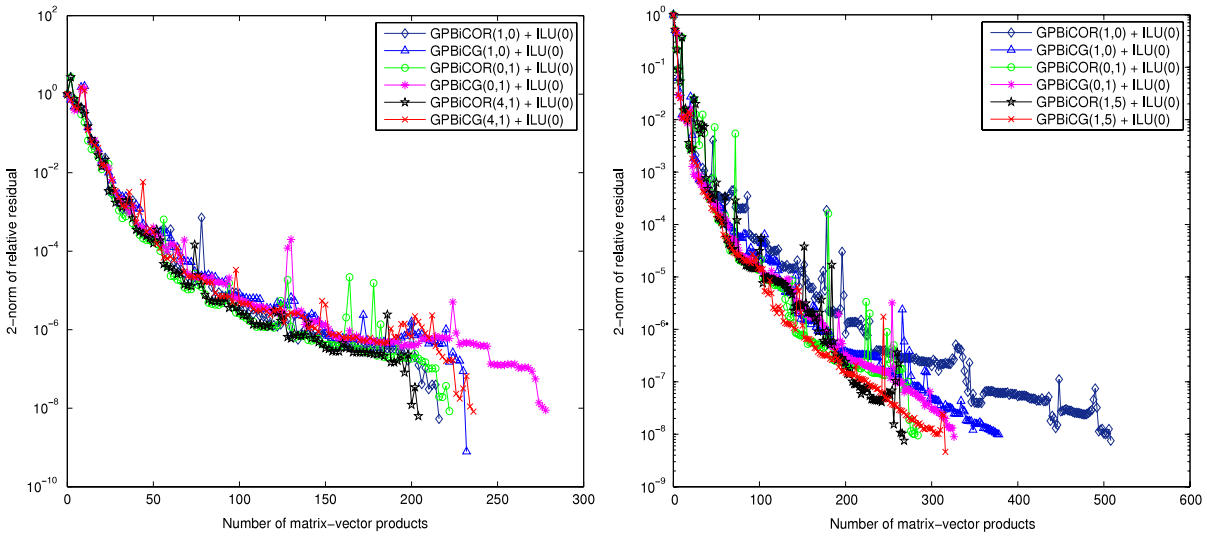

Fig. 3. Convergence histories of the different iterative methods with ILU(0) preconditioning for Example 4. Left: `ex31`; Right: `memplus`.

We clearly see from the results of Table 7 and from the convergence histories shown in Fig. 3 the good potential of the hybridized strategy and selected preconditioning can improve the performance of the GPBiCOR(m, ℓ) and GPBiCG(m, ℓ) methods to some extent. For the first problem, the PGPBiCOR(m, ℓ) methods are more efficient than the PGPBiCG(m, ℓ) methods in terms of the number of *Iters* and elapsed CPU time. Moreover, the PGPBiCOR(4, 1) method outperforms the other preconditioned iterative solvers in aspects of elapsed CPU time. However, the PGPBiCOR(4, 1) method still needs slightly more *Iters* than the PGPBiCOR(1, 1) and PGPBiCOR(1, 3) methods. All in all, we conclude that the PGPBiCOR(4, 1) method is the best choice for solving the first test problem, the PGPBiCOR(3, 1) method being an efficient alternative. Then for the second problem, the PGPBiCOR(1, 5) method outperforms all other preconditioned iterative solvers in terms of the number of *Iters* and elapsed CPU time. Meanwhile, the convergence performance of PGPBiCOR(1, 4) is not disappointing for the second problem. In addition, the level of accuracy of the approximations computed by these mentioned iterative methods (in terms of *TRR*) was almost of the same order of magnitude of the tolerance value used in the given stopping criterion, except the preconditioned GMRES(50) method on the second test problem. Roughly, the PGPBiCOR(m, ℓ) and PGPBiCG(m, ℓ) methods indeed improve the convergence of the preconditioned BiCORSTAB and preconditioned BiCGSTAB methods for these two test problems in terms of the number of *Iters*, except that the preconditioned GPBiCG(m, ℓ) method almost fails to improve the convergence of the preconditioned BiCGSTAB method for the first test problems in terms of the number of *Iters* and elapsed CPU time. Also, the PGPBiCG(m, ℓ) method also alleviates the elapsed CPU time of the preconditioned GPBiCG method, but the PGPBiCOR(m, ℓ) method fails to reduce the CPU time cost of the preconditioned GPBiCOR method dramatically on Example 4. The restarted preconditioned GMRES method still remains more expensive than the other preconditioned iterative solvers. For simplicity, Fig. 3 only shows typical plots of the convergence behavior of some different preconditioned iterative solvers; we see that the PGPBiCOR(m, ℓ) methods had fairly attractive convergence behaviors compared to the PGPBiCG(m, ℓ) methods in the last phase.

Example 5. At last but not least, we still consider two other large-scale linear systems with the coefficient matrices "`bcircuit`" and "`hcircuit`", which are from the circuit simulation problems, for illustrating the effectiveness of our proposed methods. For these two problems, we set $\boldsymbol{b} = A\boldsymbol{e}$ where $\boldsymbol{e}$ is the $n \times 1$ vector whose elements are all equal to unity, such that $\boldsymbol{e} = (1, \ldots, 1)^T$ is the exact solution. Then we want to evaluate the performance of GPBiCOR(m, ℓ) and GPBiCG(m, ℓ) with ILUT(`droptol`) preconditioning [2, pp. 321–327] for solving the test linear systems. The numerical results of different iterative solvers with ILUT(`droptol`) preconditioning are reported in Table 8.

As observed from the numerical results of Table 8 and from the convergence curves demonstrated in Fig. 4 the good potential of the hybridized strategy and selected preconditioning can enhance the performance of the GPBiCOR(m, ℓ) and GPBiCG(m, ℓ) methods to some extent. For these two problems, the PGPBiCOR(m, ℓ) methods are more efficient than the PGPBiCG(m, ℓ) methods in terms of *Iters* and elapsed CPU time. Moreover, the PGPBiCOR(5, 1) method outperforms the other preconditioned iterative solvers in aspects of elapsed CPU time for solving the first problem. However, the PGPBiCOR(5, 1) method still requires slightly more *Iters* than the PGPBiCOR(1, 1) method. All in all, we conclude that the PGPBiCOR(5, 1) method is the best choice for solving the first test problem, the PGPBiCOR(1, 1) method being an efficient alternative. Then for the second problem, the PGPBiCOR method outperforms all other preconditioned iterative solvers in terms of the number of *Iters* and elapsed CPU time. Meanwhile, the convergence performance of PGPBiCOR(4, 1) and PGPBiCOR(5, 1) is not disappointing for the second problem. In addition, the level of accuracy of the approximations computed by these mentioned iterative methods (in aspects of *TRR*) was almost of the same order of magnitude of the tolerance value

Table 8
Numerical results of different iterative solvers with ILUT(`droptol`) preconditioning for Example 5.

Method	bcircuit, `droptol = 0.01`			hcircuit, `droptol = 0.025`		
	Iters	*TRR*	Time (s)	*Iters*	*TRR*	Time (s)
OSTAB/STAB	210/458	−8.0348/−8.0130	2.5495/5.2273	34/45	−8.0745/−8.0331	2.4070/3.1008
GPO/GP	211/294	−8.0513/−8.0137	2.7157/3.5526	**32**/45	−8.3050/−8.1156	**2.3026**/3.1359
GPO(1, 1)/GP(1, 1)	**163**/347	−8.0184/−8.0789	2.0395/4.1056	34/43	−8.3455/−8.0444	2.4238/2.9862
GPO(1, 2)/GP(1, 2)	182/429	−8.0609/−8.2450	2.3046/5.0766	34/42	−8.2854/−8.0705	2.4318/2.9195
GPO(1, 3)/GP(1, 3)	174/294	−8.0105/−8.0029	2.2065/3.4806	33/42	−8.1891/−8.1431	2.3336/2.9146
GPO(1, 4)/GP(1, 4)	250/300	−8.0464/−8.0450	3.1757/3.5878	33/44	−8.2895/−8.0958	2.3577/3.0664
GPO(1, 5)/GP(1, 5)	192/244	−8.0129/−8.1348	2.4482/2.9235	33/40	−8.1724/−8.0593	2.3535/2.7955
GPO(2, 1)/GP(2, 1)	182/249	−8.0035/−8.1568	2.2714/2.9183	33/41	−8.0450/−8.0801	2.3505/2.8371
GPO(3, 1)/GP(3, 1)	177/285	−8.0360/−8.0952	2.1837/3.2849	36/50	−8.0899/−8.3091	2.5428/3.4503
GPO(4, 1)/GP(4, 1)	181/275	−8.0262/−8.0378	2.2351/3.1786	33/42	−8.2124/−8.1421	2.3315/2.9043
GPO(5, 1)/GP(5, 1)	164/286	−8.0021/−8.4862	**2.0142**/3.3061	33/43	−8.0592/−8.0156	2.3251/2.9598
CORS/CGS	171/253	−8.2548/−8.7087	2.1138/3.0651	48/47	−8.1907/−8.5568	3.2666/3.2134
IDR(2)	576	−8.1313	3.9856	76	−8.0475	2.7006
IDR(4)	345	−8.0031	2.8035	66	−8.0257	2.4739
IDR(6)	276	−8.0647	2.5821	64	−8.2077	2.5217
IDR(8)	297	−8.1607	3.1113	64	−8.2038	2.6528
TFQMR	250	−8.0153	3.6938	45	−8.3607	3.2786
GMRES(50)	465	−8.4728	11.8152	54	−7.8927	3.3677

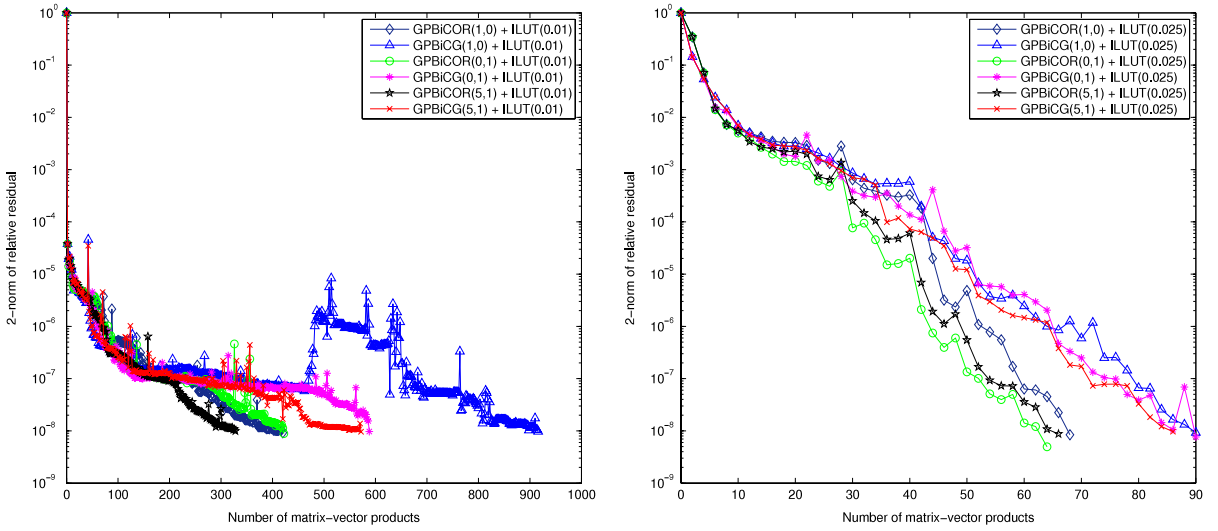

Fig. 4. Convergence histories of the different iterative methods with ILUT(`droptol`) preconditioning for Example 5. Left: `bcircuit, droptol = 0.01`; Right: `hcircuit, droptol = 0.025`.

used in the given stopping criterion, except the preconditioned GMRES(50) method on the second test problem. Roughly, the PGPBiCOR(m, ℓ) and PGPBiCG(m, ℓ) methods indeed improve the convergence of the preconditioned BiCORSTAB and preconditioned BiCGSTAB method for these two test problems in terms of the number of *Iters* and elapsed CPU time. Likewise, the PGPBiCOR(m, ℓ) methods also alleviate the elapsed CPU time of preconditioned GPBiCOR method for the first test problem, not the second test problem. In addition, the PGPBiCG(m, ℓ) method seems to fail to reduce the CPU time elapsed of preconditioned GPBiCG method dramatically on Example 5. The restarted preconditioned GMRES method still remains more expensive than the other preconditioned iterative solvers. For the sake of simplicity, Fig. 4 only illustrates representative plots of the convergence behaviors of some different preconditioned iterative methods; we see that the PGPBiCOR(m, ℓ) methods had fairly preferable convergence behaviors compared to the PGPBiCG(m, ℓ) methods in the last phase.

4. Conclusions

In this paper we present the development steps of the GPBiCOR(m, ℓ) method, which can be regarded as a hybridized variant of the BiCORSTAB and GPBiCOR methods, along with comprehensive convergence results. The numerical experiments indicate that the GPBiCOR(m, ℓ) method may be overall more efficient than the GPBiCG(m, ℓ) method recently proposed by Fujino and can be a significant method for solving the non-Hermitian systems of linear equations with real and complex matrices. However, further research is still necessary to find an optimal parameter selection strategy for the values of m and ℓ in the GPBiCOR(m, ℓ) methods. In our numerical experiments, the performance of the IDR(s) methods were also

competitive. As mentioned by the author of [48], recently, the relations between IDR(s) and BiCGStab(ℓ) and combinations of both methods have been investigated, refer to [49,50]. So we think it should be possible to combine some of the ideas behind GPBiCG(m, ℓ) and GPBiCOR(m, ℓ) with the IDR philosophy to derive novel cost-efficient iterative solvers for non-Hermitian linear systems in future work. Finally, we mention that the parallel strategies for the GPBiCG(m, ℓ) methods presented in [42] can be investigated for the proposed GPBiCOR(m, ℓ) methods. This will be the subject of a future study.

Acknowledgments

We are grateful to the anonymous referees and editor Dr. Leszek Feliks Demkowicz for their useful suggestions and comments that improved the presentation of this paper. This research is supported by 973 Program (2013CB329404), NSFC (61370147, 61170311, 61402082, 11301057, and 61170309), the Fundamental Research Funds for the Central Universities (ZYGX2013J106, ZYGX2013Z005, and ZYGX2014J084).

References

[1] T.A. Davis, Direct Methods for Sparse Linear Systems, in: SIAM Series on the Fundamentals of Algorithms, SIAM, Philadelphia, USA, 2006.
[2] Y. Saad, Iterative Methods for Sparse Linear Systems, second ed., SIAM, Philadelphia, USA, 2003.
[3] M.H. Gutknecht, Lanczos-type solvers for nonsymmetric linear systems of equation, Acta Numer. 6 (1997) 271–397.
[4] V. Simoncini, D.B. Szyld, Recent computational developments in Krylov subspace methods for linear systems, Numer. Linear Algebra Appl. 14 (2007) 1–59.
[5] Z.-Z. Bai, Motivations and realizations of Krylov subspace methods for large sparse linear systems, J. Comput. Appl. Math. 283 (2015) 71–78.
[6] M.R. Hestenes, E. Stiefel, Methods of conjugate gradients for solving linear systems, J. Res. Natl. Bur. Stand. 49 (1952) 409–436.
[7] X.-M. Gu, M. Clemens, T.-Z. Huang, L. Li, The SCBiCG class of algorithms for complex symmetric linear systems with applications in several electromagnetic model problems, Comput. Phys. Comm. 191 (2015) 52–64.
[8] X.-M. Gu, T.-Z. Huang, L. Li, H.-B. Li, T. Sogabe, M. Clemens, Quasi-minimal residual variants of the COCG and COCR methods for complex symmetric linear systems in electromagnetic simulations, IEEE Trans. Microw. Theory Tech. 62 (12) (2014) 2859–2867.
[9] Y. Saad, M.H. Schultz, GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 7 (1986) 856–869.
[10] S.C. Eisenstat, H.C. Elman, M.H. Schultz, Variational iterative methods for nonsymmetric systems of linear equations, SIAM J. Numer. Anal. 20 (1983) 345–357.
[11] P. Jiránek, M. Rozložník, M.H. Gutknecht, How to make simpler GMRES and GCR more stable, SIAM J. Matrix Anal. Appl. 30 (2008) 1483–1499.
[12] E. de Sturler, Truncation strategies for optimal Krylov subspace methods, SIAM J. Numer. Anal. 36 (1999) 864–889.
[13] R.B. Morgan, GMRES with deflated restarting, SIAM J. Sci. Comput. 24 (2002) 20–37.
[14] L.M. Carvalho, S. Gratton, R. Lago, X. Vasseur, A flexible generalized conjugate residual method with inner orthogonalization and deflated restarting, SIAM J. Matrix Anal. Appl. 32 (2011) 1212–1235.
[15] A. Gaul, M.H. Gutknecht, J. Liesen, R. Nabben, A framework for deflated and augmented Krylov subspace methods, SIAM J. Matrix Anal. Appl. 34 (2013) 495–518.
[16] C. Lanczos, Solution of systems of linear equations by minimized iterations, J. Res. Natl. Bur. Stand. 49 (1952) 33–53.
[17] R. Fletcher, Conjugate gradient methods for indefinite systems, in: G.A. Watson (Ed.), Numerical Analysis Dundee 1975, in: Lecture Notes in Mathematics, vol. 506, Springer-Verlag, Berlin, Germany, 1976, pp. 73–89.
[18] P. Sonneveld, CGS: A fast Lanczos-type solver for nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 10 (1989) 36–52.
[19] D.R. Fokkema, G.L.G. Sleijpen, H.A. van der Vorst, Generalized conjugate gradient squared, J. Comput. Appl. Math. 71 (1996) 125–146.
[20] R.W. Freund, A transpose-free quasi-minimum residual algorithm for non-Hermitian linear systems, SIAM J. Sci. Comput. 14 (1993) 470–482.
[21] 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. Stat. Comput. 13 (1992) 631–644.
[22] M.H. Gutknecht, Variants of BiCGSTAB for matrix with complex spectrum, SIAM J. Sci. Comput. 14 (1993) 1020–1033.
[23] G.L.G. Sleijpen, D.R. Fokkema, BiCGstab(ℓ) for linear equations involving matrices with complex spectrum, Electron. Trans. Numer. Anal. 1 (1993) 11–32.
[24] S.-L. Zhang, GPBi-CG: Generalized product-type methods based on Bi-CG for solving nonsymmetric linear systems, SIAM J. Sci. Comput. 18 (1997) 537–551.
[25] S.-L. Zhang, M. Natori, C.-H. Jin, Product-type Krylov-subspace methods for solving nonsymmetric linear systems, RIMS Kôkyûroku 989 (1997) 92–102. Available online at: www.kurims.kyoto-u.ac.jp/~kyodo/kokyuroku/contents/pdf/0989-9.pdf.
[26] P. Sonneveld, M.B. van Gijzen, IDR(s): A family of simple and fast algorithms for solving large nonsymmetric linear systems, SIAM J. Sci. Comput. 31 (2008) 1035–1062.
[27] M.-C. Yeung, T.F. Chan, ML(k)BiCGSTAB: A BiCGSTAB variant based on multiple Lanczos starting vectors, SIAM J. Sci. Comput. 21 (1999) 1263–1290.
[28] M.H. Gutknecht, IDR explained, Electron. Trans. Numer. Anal. 36 (2009) 126–148.
[29] Y.-F. Jing, T.-Z. Huang, Y. Zhang, L. Li, G.-H. Cheng, Z.-G. Ren, Y. Duan, T. Sogabe, B. Carpentieri, Lanczos-type variants of the COCR method for complex nonsymmetric linear systems, J. Comput. Phys. 228 (2009) 6376–6394.
[30] B. Carpentieri, Y.-F. Jing, T.-Z. Huang, The BiCOR and CORS iterative algorithms for solving nonsymmetric linear systems, SIAM J. Sci. Comput. 33 (2011) 3020–3036.
[31] Y.-F. Jing, B. Carpentieri, T.-Z. Huang, Experiments with Lanczos biconjugate A-orthonormalization methods for MoM discretizations of Maxwell's equations, Prog. Electromagn. Res. (PIER) 99 (2009) 427–451.
[32] Y.-F. Jing, T.-Z. Huang, Y. Duan, B. Carpentieri, A comparative study of iterative solutions to linear systems arising in quantum mechanics, J. Comput. Phys. 229 (2010) 8511–8520.
[33] L. Zhao, T.-Z. Huang, Y.-F. Jing, L.-J. Deng, A generalized product-type BiCOR method and its application in signal deconvolution, Comput. Math. Appl. 66 (2013) 1372–1388.
[34] J. Zhang, H. Dai, Generalized conjugate A-orthogonal residual squared method for complex non-Hermitian linear systems, J. Comput. Math. 32 (2014) 248–265.
[35] J. Zhang, H. Dai, A new quasi-minimal residual method based on a biconjugate A-orthonormalization procedure and coupled two-term recurrences, Numer. Algorithms (2015) in press. Avaiable online at: https://dx.doi.org/10.1007/s11075-015-9978-5.
[36] J. Zhang, H. Dai, A transpose-free quasi-minimal residual variant of the CORS method for solving non-Hermitian linear systems, J. Comput. Phys. 291 (2015) 20–33.
[37] L. Zhao, T.-Z. Huang, A hybrid variant of the BiCOR method for a nonsymmetric linear system with a complex spectrum, Appl. Math. Lett. 26 (2013) 457–462.
[38] K. Abe, G.L.G. Sleijpen, BiCR variants of the hybrid BiCG methods for solving linear systems with nonsymmetric matrices, J. Comput. Appl. Math. 234 (2010) 985–994.

[39] T. Sogabe, M. Sugihara, S.-L. Zhang, An extension of the conjugate residual method to nonsymmetric linear systems, J. Comput. Appl. Math. 226 (2009) 103–113.
[40] S. Fujino, GPBiCG(m, ℓ): A hybrid of BiCGSTAB and GPBiCG methods with efficiency and robustness, Appl. Numer. Math. 41 (2002) 107–117.
[41] J. Tang, Y. Shen, Y. Zheng, D. Qiu, An efficient and flexible computational model for solving the mild slope equation, Coastal Eng. 51 (2004) 143–154.
[42] S.-X. Zhu, T.-X. Gu, X.-P. Liu, Minimizing synchronizations in sparse iterative solvers for distributed supercomputers, Comput. Math. Appl. 67 (2014) 199–209.
[43] T.A. Davis, Y. Hu, The University of Florida sparse matrix collection, ACM Trans. Math. Software 38 (1) (2011) Article No. 1. Available online at: http://www.cise.ufl.edu/research/sparse/matrices/.
[44] R.W. Freund, M.H. Gutknecht, N. Nachtigal, An implementation of the look-ahead Lanczos algorithm for non-Hermitian matrices, SIAM J. Sci. Stat. Comput. 14 (1993) 137–158.
[45] Y.-M. Huang, On m-step Hermitian and skew-Hermitian splitting preconditioning methods, J. Eng. Math. 93 (2015) 77–86.
[46] M.K. Ng, J. Pan, Weighted Toeplitz regularized least squares computation for image restoration, SIAM J. Sci. Comput. 36 (2014) B94–B121.
[47] E. Chow, Y. Saad, Experimental study of ILU preconditioners for indefinite matrices, J. Comput. Appl. Math. 86 (1997) 387–414.
[48] J.-P.M. Zemke, IDR(s) and IDR(s)Eig in parallel computing, Supercomput. News 12 (2) (2010) 31–48. Avaiable online at: http://www.tuhh.de/~matjz/papers/journals/IPC_bw.pdf.
[49] K. Abe, G.L.G. Sleijpen, A BiCGStab2 variant of the IDR(s) method for solving linear equations, AIP Conf. Proc. 1479 (2012) 741–744. Avaiable online at: http://dx.doi.org/10.1063/1.4756242.
[50] G.L.G. Sleijpen, M.B. van Gijzen, Exploiting BiCGstab(ℓ) strategies to induce dimension reduction, SIAM J. Sci. Comput. 32 (2010) 2687–2709.