Numer. Math. Theor. Meth. Appl.
doi: 10.4208/nmtma.2012.m1035

Vol. **5**, No. 3, pp. 447-492
August 2012

ML(n)BiCGStab: Reformulation, Analysis and Implementation*

Man-Chung Yeung*

Department of Mathematics, University of Wyoming, Laramie, WY 82071, USA.

Received 21 May 2011; Accepted (in revised version) 1 September 2011

Available online 3 July 2012

Abstract. With the aid of index functions, we re-derive the ML(n)BiCGStab algorithm in [Yeung and Chan, SIAM J. Sci. Comput., 21 (1999), pp. 1263-1290] systematically. There are n ways to define the ML(n)BiCGStab residual vector. Each definition leads to a different ML(n)BiCGStab algorithm. We demonstrate this by presenting a second algorithm which requires less storage. In theory, this second algorithm serves as a bridge that connects the Lanczos-based BiCGStab and the Arnoldi-based FOM while ML(n)BiCG is a bridge connecting BiCG and FOM. We also analyze the breakdown situation from the probabilistic point of view and summarize some useful properties of ML(n)BiCGStab. Implementation issues are also addressed.

AMS subject classifications: 65F10, 65F15, 65F25, 65F30

Key words: CGS, BiCGStab, ML(n)BiCGStab, multiple starting Lanczos, Krylov subspace, iterative methods, linear systems.

1. Introduction

Consider the solution of the linear system

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{1.1}$$

where $\mathbf{A} \in \mathbb{C}^{N\times N}$ and $\mathbf{b} \in \mathbb{C}^N$. If we express the BiCG [4, 15] residual as $\mathbf{r}_k^{BiCG} = p_k(\mathbf{A})\mathbf{r}_0$ in terms of a polynomial $p_k(\lambda)$ of degree k and the initial residual $\mathbf{r}_0$, the residual vector $\mathbf{r}_k$ of a Lanczos-type product method† based on BiCG is defined to be $\mathbf{r}_k = \phi_k(\mathbf{A})p_k(\mathbf{A})\mathbf{r}_0$, where $\phi_k(\lambda)$ is some polynomial of degree k with $\phi_k(0) = 1$. In CGS [28], $\phi_k = p_k$. Since, in every iteration, CGS searches for an approximate solution in a larger Krylov subspace, it

*Dedicated to the Memory of Prof. Gene Golub. This paper was presented in Gene Golub Memorial Conference, Feb. 29-Mar. 1, 2008, at University of Massachusetts. This research was supported by 2008 Flittie Sabbatical Augmentation Award, University of Wyoming.

*Corresponding author. *Email address:* myeung@uwyo.edu (M.-C. Yeung)

†For this type of Krylov subspace methods, one can consult [9]. They are called hybrid BiCG methods in [27].

often converges much faster than BiCG. However, CGS usually behaves irregularly due to a lack of a smoothing mechanism. In BiCGStab [31], the ϕ_k is

$$\phi_k(\lambda) = \begin{cases} 1 & \text{if } k = 0, \\ (1-\omega_k\lambda)\phi_{k-1}(\lambda) & \text{if } k > 0. \end{cases} \tag{1.2}$$

Here ω_k is a free parameter selected to minimize the 2-norm of $\mathbf{r}_k^{BiCGStab}$ in the kth iteration. As a result, BiCGStab is generally more stable and robust than CGS. BiCGStab has been extended to BiCGStab2 [7] and BiCGStab(l) [23, 27] through the use of minimizing polynomials of higher degree. In BiCGStab2, the ϕ_k is defined by the recursion

$$\phi_k(\lambda) = \begin{cases} 1 & \text{if } k = 0, \\ (1-\omega_k\lambda)\phi_{k-1}(\lambda) & \text{if } k \text{ is odd}, \\ ((\alpha_k\lambda+\beta_k)(1-\omega_{k-1}\lambda)+1-\beta_k)\phi_{k-2}(\lambda) & \text{if } k \text{ is even}. \end{cases}$$

The parameters are again chosen to minimize BiCGStab2 residuals. Likewise, BiCGStab(l) defines its ϕ_k as

$$\phi_k(\lambda) = \begin{cases} 1 & \text{if } k = 0, \\ (1+\sum_{j=1}^{l}\alpha_j\lambda^j)\phi_{k-l}(\lambda) & \text{if } k \text{ is a multiple of } l, \end{cases}$$

where the parameters in the factor $1+\sum_{j=1}^{l}\alpha_j\lambda^j$ yields an l-dimensional minimization in every lth step. BiCGStab2 and BiCGStab(l) usually converge faster than BiCGStab because of smaller residuals in magnitude while avoiding near-breakdowns caused by a possibly too small ω_k. CGS, BiCGStab and BiCGStab2 have been summarized and generalized by GPBi-CG [40] where ϕ_k is

$$\phi_k(\lambda) = \begin{cases} 1 & \text{if } k = 0, \\ 1-\omega_1\lambda & \text{if } k = 1, \\ (1+\beta_k-\omega_k\lambda)\phi_{k-1}(\lambda)-\beta_k\phi_{k-2}(\lambda) & \text{if } k > 1. \end{cases}$$

GPBi-CG will become CGS, BiCGStab or BiCGStab2 when the α, β, ω are appropriately chosen. For detailed descriptions of these and other product-type methods, one is referred to [6,8,20,22,32] and the references therein. Moreover, a history of product-type methods can be found in [10]. The history starts three decades ago with IDR [36] method which can be considered as the predecessor of CGS and BiCGStab [24]. Recently, IDR has been generalized to IDR(s) with a shadow space of higher dimension, see [24, 30, 34]. IDR(s) has close relations with ML(s)BiCGStab.

Generalizations of BiCGStab to methods based on the generalizations of BiCG have been made. For example, BL-BiCGStab [3] is a BiCGStab variant built on the BL-BiCG [16] for the solution of systems with multiple right-hand sides. ML(n)BiCGStab [39] is another BiCGStab variant built on ML(n)BiCG, a BiCG-like method derived from a variant of the band Lanczos process described in [1] with n left-starting vectors and a single right-starting vector.

The derivation of the ML(n)BiCGStab algorithm in [39] was complicated. In this paper, we exploit the concept of index functions to re-derive the algorithm in a more systematic way, step by step. Index functions were introduced in [38] by Boley for the purpose of simplifying the development of the transpose-free multiple starting Lanczos process or the Sonneveld-van der Vorst-Lanczos process (SVLP)[‡], and they proved to be very helpful.

Motivated from the study of SVLP in [38], we recognized that the definition of the ML(n)BiCGStab residual $\mathbf{r}_k$ in [39] is not unique. There are n different ways to define $\mathbf{r}_k$. Let $\widehat{\mathbf{r}}_k$ be the residual of ML(n)BiCG and $\phi_k(\lambda)$ as in (1.2). Then, the ML(n)BiCGStab residual $\mathbf{r}_k$ in [39] is

$$\mathbf{r}_k = \phi_{j+1}(\mathbf{A})\widehat{\mathbf{r}}_k, \tag{1.3}$$

where $k = jn + i$, $1 \le i \le n$, $j = 0, 1, 2, \cdots$. Starting from $k = 1$, let us call every n consecutive k-iterations an iteration "cycle". For example, iterations $k = 1, 2, \cdots, n$ form the first cycle, iterations $k = n+1, n+2, \cdots, 2n$ the second cycle and so on. Then definition (1.3) increases the degree of ϕ by 1 at the beginning of a cycle. One actually can define $\mathbf{r}_k$ by increasing the degree of ϕ by 1 anywhere within an iteration cycle. Each definition will lead to a different ML(n)BiCGStab algorithm. As an illustration, we shall derive a second ML(n)BiCGStab algorithm associated with the definition

$$\mathbf{r}_{jn+i} = \begin{cases} \phi_j(\mathbf{A})\widehat{\mathbf{r}}_{jn+i} & \text{if} \quad 1 \le i \le n-1, \\ \phi_{j+1}(\mathbf{A})\widehat{\mathbf{r}}_{jn+i} & \text{if} \quad i = n. \end{cases} \tag{1.4}$$

Eq. (1.4) increases the degree of ϕ by 1 at the end of a cycle. The resulting algorithm requires about 25% less storage (not counting the storage of the coefficient matrix and the preconditioner) than the algorithm associated with definition (1.3). However, one drawback with this storage-saving algorithm is that, in some experiments, its computed residual $\mathbf{r}_k$ can easily diverge from the corresponding exact residual when n is moderately large.

ML(n)BiCG and ML(n)BiCGStab possess a set of left starting vectors (or, shadow vectors) $\mathbf{q}_1, \cdots, \mathbf{q}_n$ that can be chosen freely. This freedom appears to be an advantage of the methods. It helps stabilize the performance of the algorithms (see [39, p.1] for an explanation) and allows us to see a connection between the Lanczos-based BiCG/BiCGStab and the Arnoldi-based FOM.

One question of interest about ML(n)BiCG and ML(n)BiCGStab is: can they solve (1.1) when $\mathbf{A}$ is singular? Since both methods search for a solution of (1.1) in the affine Krylov subspace

$$\mathbf{x}_0 + \mathbb{K}(\mathbf{A}, \mathbf{r}_0) \equiv \mathbf{x}_0 + \left\{ \sum_{i=0}^{k} c_i \mathbf{A}^i \mathbf{r}_0 \,\middle|\, c_i \in \mathbb{C}, k \in \mathbb{N}_0 \right\}, \tag{1.5}$$

where $\mathbb{N}_0$ is the set of nonnegative integers, they must fail to converge if (1.5) contains no solution of the linear system. Notice that (1.5) contains a solution of (1.1) if and only if GMRES converges to a solution of (1.1) that lies in (1.5) — this can be derived from [2, Lemma 2.1]. It then follows from [2, Th. 2.6] and [5, Th. 4.1] that

[‡]We rename the process to remember the contributions of the two pioneers in this field of transpose-free Lanczos. In [38], it was shown that the Arnoldi process is an extreme case of SVLP.

Eq. (1.5) contains a solution of (1.1) for any initial guess $\mathbf{x}_0$ if and only if (1.1) is consistent and $ker(\mathbf{A}) \cap Im(\mathbf{A}) = \{\mathbf{0}\}$.

In the case when (1.1) is consistent but $ker(\mathbf{A}) \cap Im(\mathbf{A}) \neq \{\mathbf{0}\}$, the selection of $\mathbf{x}_0$ should be a careful step. For example, consider the example from [2] where $\mathbf{A} = [0,1;0,0], \mathbf{b} = [1,0]^T$. This system is consistent. If one selects $\widehat{\mathbf{x}}_0 = [1,0]^T$, then (1.5) contains no solution. The other thing that hampers the convergence of ML(n)BiCG and ML(n)BiCGStab is breakdown by zero division. We shall show that the two methods will almost surely converge without breakdown by zero division to a solution of (1.1) if the shadow vectors are chosen randomly and the initial guess $\mathbf{x}_0$ is selected such that (1.5) contains a solution of (1.1).

The outline of the paper is as follows. In §2, we introduce index functions. In §3, we present the ML(n)BiCG algorithm introduced in [39], from which ML(n)BiCGStab algorithms are derived. In §4, we rederive the ML(n)BiCGStab algorithm in [39] by index functions. In §5, we derive a storage-saving ML(n)BiCGStab algorithm from a different definition of the residual vector. In §6, we discuss relationships of ML(n)BiCGStab with some other methods. In §7, implementation issues are addressed. Conclusions are made in §8.

2. Index Functions

Let be given a positive integer n. For all integers k, we define

$$g_n(k) = \lfloor (k-1)/n \rfloor \quad \text{and} \quad r_n(k) = k - n g_n(k),$$

where $\lfloor \cdot \rfloor$ rounds its argument to the nearest integer towards minus infinity. We call g_n and r_n index functions; they are defined on $\mathbb{Z}$, the set of all integers, with ranges $\mathbb{Z}$ and $\{1, 2, \cdots, n\}$, respectively. If we write

$$k = jn + i, \tag{2.1}$$

with $1 \leq i \leq n$ and $j \in \mathbb{Z}$, then

$$g_n(jn+i) = j \quad \text{and} \quad r_n(jn+i) = i. \tag{2.2}$$

Table 1 illustrates the behavior of g_n and r_n with $n = 3$. It can be seen that $g_n(k)$ has a jump when k, moved from left to right, passes a multiple of n.

Table 1: Simple illustration of the index functions for $n = 3$.

k	0	1 2 3	4 5 6	7 8 9	10 11 12	$\cdots$
$g_n(k)$	-1	0 0 0	1 1 1	2 2 2	3 3 3	$\cdots$
$r_n(k)$	3	1 2 3	1 2 3	1 2 3	1 2 3	$\cdots$

The following properties can be easily verified by using (2.2).

Proposition 2.1. *Let $k \in \mathbb{N}$, the set of all positive integers, and $s \in \mathbb{N}_0 = \mathbb{N} \cup \{0\}$.*

(a) $g_n(k+n) = g_n(k) + 1$ *and* $r_n(k+n) = r_n(k)$.

(b) $g_n(s+1)+1 = g_n(k+1)$ *if* $\max(k-n,0) \le s \le g_n(k)n - 1$.

(c) $g_n(s+1) = g_n(g_n(k)n+1) = g_n(k)$ *if* $g_n(k)n \le s \le k-1$.

(d) $g_n(k+1) = g_n(k)+1$ *if* $r_n(k) = n$; $g_n(k+1) = g_n(k)$ *if* $r_n(k) < n$.

(e) $\max(k-n,0) > g_n(k)n - 1$ *if* $r_n(k) = n$ *or* $g_n(k) = 0$.

3. A ML(n)BiCG Algorithm

Analogously to the derivation of BiCGStab from BiCG, ML(n)BiCGStab in [39] was derived from a BiCG-like method named ML(n)BiCG, which was built upon a band Lanczos process with n left starting vectors and a single right starting vector. In this section, we present the algorithm of ML(n)BiCG from [39] and summarize some of its properties.

3.1. The Algorithm

Consider the solution of (1.1). Throughout the paper we do not assume that the coefficient matrix $\mathbf{A}$ is nonsingular. The iterative solution of singular systems has been extensively studied, see, for instance, [2, 5, 14, 18, 35] and the references therein.

Let be given n vectors $\mathbf{q}_1, \ldots, \mathbf{q}_n \in \mathbb{C}^N$, which we call *left starting vectors* or *shadow vectors*. Define

$$\mathbf{p}_k = \left(\mathbf{A}^H\right)^{g_n(k)} \mathbf{q}_{r_n(k)} \tag{3.1}$$

for $k = 1, 2, 3, \cdots$. The following algorithm for the solution of (1.1) is from [39].

Algorithm 3.1. ML(n)BiCG

1. *Choose an initial guess* $\widehat{\mathbf{x}}_0$ *and n vectors* $\mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_n$.
2. *Compute* $\widehat{\mathbf{r}}_0 = \mathbf{b} - \mathbf{A}\widehat{\mathbf{x}}_0$ *and set* $\mathbf{p}_1 = \mathbf{q}_1$, $\widehat{\mathbf{g}}_0 = \widehat{\mathbf{r}}_0$.
3. *For* $k = 1, 2, 3, \cdots$, *until convergence:*
4. $\quad \alpha_k = \mathbf{p}_k^H \widehat{\mathbf{r}}_{k-1} / \mathbf{p}_k^H \mathbf{A}\widehat{\mathbf{g}}_{k-1}$;
5. $\quad \widehat{\mathbf{x}}_k = \widehat{\mathbf{x}}_{k-1} + \alpha_k \widehat{\mathbf{g}}_{k-1}$;
6. $\quad \widehat{\mathbf{r}}_k = \widehat{\mathbf{r}}_{k-1} - \alpha_k \mathbf{A}\widehat{\mathbf{g}}_{k-1}$;
7. $\quad$ *For* $s = \max(k-n,0), \cdots, k-1$
8. $\quad\quad \beta_s^{(k)} = -\mathbf{p}_{s+1}^H \mathbf{A}\left(\widehat{\mathbf{r}}_k + \sum_{t=\max(k-n,0)}^{s-1} \beta_t^{(k)} \widehat{\mathbf{g}}_t\right) / \mathbf{p}_{s+1}^H \mathbf{A}\widehat{\mathbf{g}}_s$;
9. $\quad$ *End*
10. $\quad \widehat{\mathbf{g}}_k = \widehat{\mathbf{r}}_k + \sum_{s=\max(k-n,0)}^{k-1} \beta_s^{(k)} \widehat{\mathbf{g}}_s$;
11. $\quad$ *Compute* $\mathbf{p}_{k+1}$ *according to (3.1)*
12. *End*

Algorithm 3.1 consists of exact mathematical formulas for $\alpha_k, \beta_s^{(k)}, \widehat{\mathbf{x}}_k, \widehat{\mathbf{r}}_k$ and $\widehat{\mathbf{g}}_k$ obtained in §3 of [39]. Even though the algorithm has not been tested, it is believed to be numerically instable because of Line 11 in which the shadow vectors are repeatedly multiplied by $\mathbf{A}^H$, a type of operation which is highly sensitive to round-off errors. The algorithm is introduced only for the purpose of developing ML(n)BiCGStab algorithms.

Algorithm 3.1 is a variation of the classical BiCG algorithm. The left-hand side (shadow) Krylov subspace of BiCG is replaced by the block Krylov subspace with n starting vectors $\mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_n$:

$$\begin{aligned}\mathbb{B}_k &\equiv \text{the space spanned by the first } k \text{ columns of } [\mathbf{Q}, \mathbf{A}^H\mathbf{Q}, (\mathbf{A}^H)^2\mathbf{Q}, \cdots] \\ &= span\{\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_k\} \\ &= \textstyle\sum_{i=1}^{r_n(k)} \mathbb{K}_{g_n(k)+1}(\mathbf{A}^H, \mathbf{q}_i) + \sum_{i=r_n(k)+1}^{n} \mathbb{K}_{g_n(k)}(\mathbf{A}^H, \mathbf{q}_i),\end{aligned}$$

where $\mathbf{Q} \equiv [\mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_n]$ and

$$\mathbb{K}_t(\mathbf{M}, \mathbf{v}) \equiv span\{\mathbf{v}, \mathbf{M}\mathbf{v}, \cdots, \mathbf{M}^{t-1}\mathbf{v}\}$$

for $\mathbf{M} \in \mathbb{C}^{N\times N}, \mathbf{v} \in \mathbb{C}^N$ and $t \in \mathbb{N}$. Moreover, in ML(n)BiCG, the basis used for $\mathbb{B}_k$ is not chosen to be bi-orthogonal, but simply the set $\{\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_k\}$. Therefore, the ML($n$)BiCG algorithm can be viewed as a generalization of a one-sided Lanczos algorithm (see [9,19]). The likely ill-conditioning of this basis does not matter, as the algorithm is only a technical tool for deriving ML(n)BiCGStab and this basis disappears in ML(n)BiCGStab because $\mathbf{A}^H$ will be absorbed by the residuals and direction vectors of ML(n)BiCGStab. For constructing the right-hand side basis consisting of residuals $\widehat{\mathbf{r}}_k$, we used recurrences that generalize the coupled two-term recurrences of BiCG, that is, direction vectors $\widehat{\mathbf{g}}_k$ are also constructed.

3.2. Properties

Let ν be the degree of the minimal polynomial $p_{min}(\lambda; \mathbf{A}, \widehat{\mathbf{r}}_0)$ of $\widehat{\mathbf{r}}_0$ with respect to $\mathbf{A}$, namely, the unique monic polynomial $p(\lambda)$ of minimum degree such that $p(\mathbf{A})\widehat{\mathbf{r}}_0 = \mathbf{0}$, and let

$$\widehat{\mathbf{S}}_\nu = [\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_\nu]^H \mathbf{A}[\widehat{\mathbf{r}}_0, \mathbf{A}\widehat{\mathbf{r}}_0, \cdots, \mathbf{A}^{\nu-1}\widehat{\mathbf{r}}_0]$$

and

$$\widehat{\mathbf{W}}_\nu = [\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_\nu]^H [\widehat{\mathbf{r}}_0, \mathbf{A}\widehat{\mathbf{r}}_0, \cdots, \mathbf{A}^{\nu-1}\widehat{\mathbf{r}}_0].$$

Denote by $\widehat{\mathbf{S}}_l$ and $\widehat{\mathbf{W}}_l$ the $l \times l$ leading principal submatrices of $\widehat{\mathbf{S}}_\nu$ and $\widehat{\mathbf{W}}_\nu$ respectively. We now summarize some useful facts about Algorithm 3.1. They can be derived from the construction procedure of the algorithm.

Proposition 3.1. *In infinite precision arithmetic, if* $\prod_{l=1}^{\nu} \det(\widehat{\mathbf{S}}_l)\det(\widehat{\mathbf{W}}_l) \neq 0$, *then Algorithm* 3.1 *does not break down by zero division for* $k = 1, 2, \cdots, \nu$, *and* x_ν *is an exact solution of* (1.1). *Moreover, the computed quantities satisfy*

(a) $\widehat{\mathbf{x}}_k \in \widehat{\mathbf{x}}_0 + \mathbb{K}_k(\mathbf{A}, \widehat{\mathbf{r}}_0)$ *and* $\widehat{\mathbf{r}}_k = \mathbf{b} - \mathbf{A}\widehat{\mathbf{x}}_k \in \widehat{\mathbf{r}}_0 + \mathbf{A}\mathbb{K}_k(\mathbf{A}, \widehat{\mathbf{r}}_0)$ *for* $1 \leq k \leq \nu$.

(b) $span\{\widehat{\mathbf{r}}_0, \widehat{\mathbf{r}}_1, \cdots, \widehat{\mathbf{r}}_{k-1}\} = \mathbb{K}_k(\mathbf{A}, \widehat{\mathbf{r}}_0)$ *for* $1 \le k \le \nu$.

(c) $span\{\mathbf{A}\widehat{\mathbf{r}}_0, \mathbf{A}\widehat{\mathbf{r}}_1, \cdots, \mathbf{A}\widehat{\mathbf{r}}_{\nu-1}\} = \mathbb{K}_\nu(\mathbf{A}, \widehat{\mathbf{r}}_0)$.

(d) $\widehat{\mathbf{r}}_k \perp span\{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_k\}$ *and* $\widehat{\mathbf{r}}_k \not\perp \mathbf{p}_{k+1}$ *for* $0 \le k \le \nu - 1$.[§]

(e) $span\{\widehat{\mathbf{g}}_0, \widehat{\mathbf{g}}_1, \cdots, \widehat{\mathbf{g}}_{k-1}\} = \mathbb{K}_k(\mathbf{A}, \widehat{\mathbf{r}}_0)$ *for* $1 \le k \le \nu$.

(f) $span\{\mathbf{A}\widehat{\mathbf{g}}_0, \mathbf{A}\widehat{\mathbf{g}}_1, \cdots, \mathbf{A}\widehat{\mathbf{g}}_{\nu-1}\} = \mathbb{K}_\nu(\mathbf{A}, \widehat{\mathbf{r}}_0)$.

(g) $\mathbf{A}\widehat{\mathbf{g}}_k \perp span\{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_k\}$ *and* $\mathbf{A}\widehat{\mathbf{g}}_k \not\perp \mathbf{p}_{k+1}$ *for* $0 \le k \le \nu - 1$.

Because of Proposition 3.1(a) and (d), ML(n)BiCG is an oblique projection Krylov subspace method [20].

Remark 3.1.

(i) The matrices $\widehat{\mathbf{S}}_l$ and $\widehat{\mathbf{W}}_l$ have already appeared in [12, 13] where they were called moment matrices. Proposition 3.1 can be regarded as a generalization of Theorem 2 in [13] from $n = 1$ to $n > 1$.

(ii) Just like BiCG, ML(n)BiCG also has two types of breakdown caused, respectively, by the failure of the underlying Lanczos process and the nonexistence of the LU factorizations of the Hessenberg matrix of the recurrence coefficients. Both types of breakdown are reflected in Algorithm 3.1 by $\mathbf{p}_k^H \mathbf{A}\widehat{\mathbf{g}}_{k-1} = 0$. The condition $\prod_{l=1}^{\nu} \det(\widehat{\mathbf{W}}_l) \neq 0$ guarantees that the underlying Lanczos process works without breakdown by zero division, and the condition $\prod_{l=1}^{\nu} \det(\widehat{\mathbf{S}}_l) \neq 0$ ensures that the LU factorizations exist.

(iii) $\det(\widehat{\mathbf{S}}_\nu) \neq 0$ implies that $p_{min}(0; \mathbf{A}, \widehat{\mathbf{r}}_0) \neq 0$ which, in turn, implies that (1.1) has a solution lying in $\widehat{\mathbf{x}}_0 + \mathbb{K}_\nu(\mathbf{A}, \widehat{\mathbf{r}}_0)$.

The derivation of a ML(n)BiCGStab algorithm will require the following result which, in the case when $n = 1$, has been used in CGS and BiCGStab.

Corollary 3.1. *Let* $s \in \mathbb{N}$ *and*

$$\psi_{g_n(s)}(\lambda) = c_{g_n(s)} \lambda^{g_n(s)} + c_{g_n(s)-1} \lambda^{g_n(s)-1} + \cdots + c_0$$

be any polynomial of exact degree $g_n(s)$. *Then, under the assumptions of Proposition* 3.1,

$$\mathbf{p}_s^H \widehat{\mathbf{r}}_k = \frac{1}{c_{g_n(s)}} \mathbf{q}_{r_n(s)}^H \psi_{g_n(s)}(\mathbf{A}) \widehat{\mathbf{r}}_k \quad \textit{and} \quad \mathbf{p}_s^H \mathbf{A}\widehat{\mathbf{g}}_k = \frac{1}{c_{g_n(s)}} \mathbf{q}_{r_n(s)}^H \mathbf{A} \psi_{g_n(s)}(\mathbf{A}) \widehat{\mathbf{g}}_k$$

if $0 \le k \le \nu - 1$ *and* $s \le k + n$.

[§]We say that $\mathbf{u} \perp \mathbf{v}$ if $\mathbf{u}^H \mathbf{v} = 0$.

Proof. It is easy to verify that

$$\mathbf{p}_s - \frac{1}{\bar{c}_{g_n(s)}}\bar{\psi}_{g_n(s)}(\mathbf{A}^H)\mathbf{q}_{r_n(s)} \in \mathbb{B}_k$$

by Proposition 2.1(a) and (3.1), where the overbar denotes complex conjugation. The corollary then follows from Proposition 3.1(d) and (g). □

Corollary 3.1 essentially says that adding to $\mathbf{p}_s$ a vector from $\mathbb{B}_k$ does not change the inner products $\mathbf{p}_s^H\widehat{\mathbf{r}}_k$ and $\mathbf{p}_s^H\mathbf{A}\widehat{\mathbf{g}}_k$.

The condition $\prod_{l=1}^{\nu}\det(\widehat{\mathbf{W}}_l)\det(\widehat{\mathbf{S}}_l) \neq 0$ in Proposition 3.1 holds in all but very exceptional examples. In fact, it is a generic property in the following sense.

Lemma 3.1. [13, Prop. 4] *If p is a nonzero polynomial in the variables $x_1, x_2, \cdots, x_k \in \mathbb{C}$, then $\{(x_1, x_2, \cdots, x_k) \in \mathbb{C}^k | p(x_1, x_2, \cdots, x_k) = 0\}$ is a measure-zero set in $\mathbb{C}^k$.*

We apply Lemma 3.1 to $\det(\widehat{\mathbf{W}}_l)$ and $\det(\widehat{\mathbf{S}}_l)$, which are polynomials in the elements of $\bar{\mathbf{q}}_1, \cdots, \bar{\mathbf{q}}_n$, to obtain

Lemma 3.2. *Let $\widehat{\mathbf{r}}_0 \in \mathbb{C}^N$, $\widehat{\mathbf{r}}_0 \neq \mathbf{0}$[¶] and $\mathbf{A} \in \mathbb{C}^{N\times N}$. Then*

(a) $\{(\mathbf{q}_1, \cdots, \mathbf{q}_n) \in \mathbb{C}^{nN} | \det(\widehat{\mathbf{W}}_l) = 0\}$ *is measure-zero for* $1 \leq l \leq \nu$.

(b) $\{(\mathbf{q}_1, \cdots, \mathbf{q}_n) \in \mathbb{C}^{nN} | \det(\widehat{\mathbf{S}}_l) = 0\}$ *is measure-zero for* $1 \leq l \leq \nu - 1$.

(c) $\{(\mathbf{q}_1, \cdots, \mathbf{q}_n) \in \mathbb{C}^{nN} | \det(\widehat{\mathbf{S}}_\nu) = 0\}$ *is measure-zero if* $p_{min}(0; \mathbf{A}, \widehat{\mathbf{r}}_0) \neq 0$*;* $\det(\widehat{\mathbf{S}}_\nu) = 0$ *for all* $(\mathbf{q}_1, \cdots, \mathbf{q}_n) \in \mathbb{C}^{nN}$ *if* $p_{min}(0; \mathbf{A}, \widehat{\mathbf{r}}_0) = 0$.

Proof. We only prove Part (c) since the arguments for Parts (a) and (b) are similar. If $p_{min}(0; \mathbf{A}, \widehat{\mathbf{r}}_0) = 0$, then $\mathbf{A}^\nu\widehat{\mathbf{r}}_0$ is a linear combination of $\mathbf{A}\widehat{\mathbf{r}}_0, \cdots, \mathbf{A}^{\nu-1}\widehat{\mathbf{r}}_0$ or $\mathbf{A}^\nu\widehat{\mathbf{r}}_0 = \mathbf{0}$ in the case when $\nu = 1$. Hence $\det(\widehat{\mathbf{S}}_\nu) \equiv 0$ no matter what $\mathbf{q}_1, \cdots, \mathbf{q}_n$ are. Thus, we assume $p_{min}(0; \mathbf{A}, \widehat{\mathbf{r}}_0) \neq 0$ in the following.

Case $n = 1$. In this case, $\mathbf{p}_k = (\mathbf{A}^H)^{g_n(k)}\mathbf{q}_{r_n(k)} = (\mathbf{A}^H)^{k-1}\mathbf{q}_1$ and $\widehat{\mathbf{S}}_\nu$ is a Hankel matrix

$$\widehat{\mathbf{S}}_\nu = \begin{bmatrix} \widehat{s}_1 & \widehat{s}_2 & \cdots & \widehat{s}_\nu \\ \widehat{s}_2 & \widehat{s}_3 & \cdots & \widehat{s}_{\nu+1} \\ \cdots & \cdots & \cdots & \cdots \\ \widehat{s}_\nu & \widehat{s}_{\nu+1} & \cdots & \widehat{s}_{2\nu-1} \end{bmatrix}, \tag{3.2}$$

where $\widehat{s}_t = \mathbf{q}_1^H\mathbf{A}^t\widehat{\mathbf{r}}_0$ for $t = 1, 2, \cdots, 2\nu - 1$. Since $p_{min}(0; \mathbf{A}, \widehat{\mathbf{r}}_0) \neq 0$, $\{\mathbf{A}\widehat{\mathbf{r}}_0, \mathbf{A}^2\widehat{\mathbf{r}}_0, \cdots, \mathbf{A}^\nu\widehat{\mathbf{r}}_0\}$ is a linearly independent set. Perform a *QR* factorization on the $N \times \nu$ matrix

$$\left[\mathbf{A}\widehat{\mathbf{r}}_0, \mathbf{A}^2\widehat{\mathbf{r}}_0, \cdots, \mathbf{A}^\nu\widehat{\mathbf{r}}_0\right] = \mathbf{QR},$$

[¶]In Lemma 3.2 and Corollary 3.2, $\widehat{\mathbf{r}}_0$ can be any non-zero vector in $\mathbb{C}^N$. It is not necessarily a residual vector like $\widehat{\mathbf{r}}_0 = \mathbf{b} - \mathbf{A}\widehat{\mathbf{x}}_0$.

where $\mathbf{Q} \in \mathbb{C}^{N\times N}$ is unitary and $\mathbf{R} \in \mathbb{C}^{N\times \nu}$ is upper triangular with positive main diagonal elements $r_{11}, r_{22}, \cdots, r_{\nu\nu}$. Write

$$\begin{aligned}
&[\widehat{s}_1, \widehat{s}_2, \cdots, \widehat{s}_\nu, \vdots \widehat{s}_{\nu+1}, \cdots, \widehat{s}_{2\nu-1}] \\
&= \mathbf{q}_1^H [\mathbf{A}\widehat{\mathbf{r}}_0, \mathbf{A}^2\widehat{\mathbf{r}}_0, \cdots, \mathbf{A}^\nu\widehat{\mathbf{r}}_0, \vdots \mathbf{A}^{\nu+1}\widehat{\mathbf{r}}_0, \cdots, \mathbf{A}^{2\nu-1}\widehat{\mathbf{r}}_0] \\
&= \mathbf{q}_1^H [\mathbf{QR}, \vdots \mathbf{A}^{\nu+1}\widehat{\mathbf{r}}_0, \cdots, \mathbf{A}^{2\nu-1}\widehat{\mathbf{r}}_0] \\
&= \theta^H [\mathbf{R}, \vdots \mathbf{Q}^H [\mathbf{A}^{\nu+1}\widehat{\mathbf{r}}_0, \cdots, \mathbf{A}^{2\nu-1}\widehat{\mathbf{r}}_0]], \qquad (3.3)
\end{aligned}$$

where $\theta = [\theta_1, \cdots, \theta_N]^T \equiv \mathbf{Q}^H\mathbf{q}_1$. If we set $\theta_\nu = 1$ and all other elements of θ to zero, then (3.3) has the form

$$[\widehat{s}_1, \cdots, \widehat{s}_{\nu-1}, \widehat{s}_\nu, \vdots \widehat{s}_{\nu+1}, \cdots, \widehat{s}_{2\nu-1}] = [0, \cdots, 0, r_{\nu\nu}, \vdots *, \cdots, *].$$

For this special choice of θ, we have $|\det(\widehat{\mathbf{S}}_\nu)| = r_{\nu\nu}^\nu \neq 0$. Thus $\det(\widehat{\mathbf{S}}_l)$ is a nonzero polynomial and Part (c) follows from Lemma 3.1.

Case $n > 1$. The proof of this case can be reduced to that of case $n = 1$ by setting $\mathbf{q}_i = (\mathbf{A}^H)^{k_i}\mathbf{q}_1$ for some appropriate integers $k_i > 0$ where $2 \le i \le n$. As an illustration, assume that $n = 2$ and $\nu = 5$. Then, by definition,

$$\widehat{\mathbf{S}}_\nu = \widehat{\mathbf{S}}_5 = [\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_5]^H \mathbf{A}[\widehat{\mathbf{r}}_0, \mathbf{A}\widehat{\mathbf{r}}_0, \cdots, \mathbf{A}^4\widehat{\mathbf{r}}_0]$$

where $[\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_5] = [\mathbf{q}_1, \mathbf{q}_2, \mathbf{A}^H\mathbf{q}_1, \mathbf{A}^H\mathbf{q}_2, (\mathbf{A}^H)^2\mathbf{q}_1]$. If we set $\mathbf{q}_2 = (\mathbf{A}^H)^3\mathbf{q}_1$, $\widehat{\mathbf{S}}_5$ will be the Hankel matrix (3.2) with some rows permuted, and the proof of $\{(\mathbf{q}_1, \mathbf{q}_2) \in \mathbb{C}^{2N} | \det(\widehat{\mathbf{S}}_5) = 0\}$ being a measure-zero set will be reduced to the case $n = 1$. □

Corollary 3.2. *Let $\widehat{\mathbf{r}}_0 \in \mathbb{C}^N$, $\widehat{\mathbf{r}}_0 \neq \mathbf{0}$ and $\mathbf{A} \in \mathbb{C}^{N\times N}$. Then*

$$\left\{ (\mathbf{q}_1, \cdots, \mathbf{q}_n) \in \mathbb{C}^{nN} \Big| \prod_{l=1}^{\nu} \left(\det(\widehat{\mathbf{W}}_l) \det(\widehat{\mathbf{S}}_l) \right) = 0 \right\}$$

is a measure-zero set in $\mathbb{C}^{nN}$ if and only if $p_{min}(0; \mathbf{A}, \widehat{\mathbf{r}}_0) \neq 0$.

Proof. Note that

$$\left\{ \prod_{l=1}^{\nu} \det(\widehat{\mathbf{W}}_l) \det(\widehat{\mathbf{S}}_l) = 0 \right\} \subset \bigcup_{l=1}^{\nu} \left(\{ \det(\widehat{\mathbf{W}}_l) = 0 \} \cup \{ \det(\widehat{\mathbf{S}}_l) = 0 \} \right)$$

and a finite union of measure-zero sets is measure-zero. □

Remark 3.2.

(i) Parts (a) and (b) of Lemma 3.2 have been proved in [29, Th. 3] for non-defective $\mathbf{A}$. Eigenvalues were employed in the proof. In Lemma 3.2, we presented an alternative proof and removed the constraint of non-defectiveness.

(ii) $p_{min}(0;\mathbf{A},\widehat{\mathbf{r}}_0) \neq 0$ if and only if the affine subspace $\widehat{\mathbf{x}}_0 + \mathbb{K}(\mathbf{A},\widehat{\mathbf{r}}_0)$ contains a solution of (1.1).

The following theorem then follows from Proposition 3.1 and Corollary 3.2.

Theorem 3.1. *If $\mathbf{q}_1,\cdots,\mathbf{q}_n$ are vectors with independent and continuous random elements, Algorithm 3.1 will almost surely work without breakdown by zero division to find a solution from the affine subspace $\widehat{\mathbf{x}}_0 + \mathbb{K}(\mathbf{A},\widehat{\mathbf{r}}_0)$ provided that $\widehat{\mathbf{x}}_0$ is chosen such that the affine subspace contains a solution of* (1.1).

Remark 3.2.

(i) If we randomly pick the initial guess $\widehat{\mathbf{x}}_0$ and set $\mathbf{q}_1 = \mathbf{b} - \mathbf{A}\widehat{\mathbf{x}}_0$, then Algorithm 3.1 with $n = 1$, or equivalently in mathematics, the standard BiCG (see §6), will almost surely solve (1.1) without breakdown by zero division for all, but a certain small class of, nonsingular $\mathbf{A}$. For details, see [13]. In Theorem 3.1, the vectors $\mathbf{q}_1,\cdots,\mathbf{q}_n$ are independent of $\widehat{\mathbf{x}}_0$.

(ii) The $\widehat{\mathbf{x}}_0$ in Theorem 3.1 is a user-provided vector. It may not be a random vector in some applications. For example, in cases where a sequence of similar linear systems is solved, the solution from the previous system may be used as the $\widehat{\mathbf{x}}_0$ for the new system.

4. A ML(n)BiCGStab Algorithm

A ML(n)BiCGStab algorithm was derived from ML(n)BiCG in [39] (Algorithm 2 without preconditioning and Algorithm 3 with preconditioning in [39]), but the derivation there is complicated and less inspiring. In this section, we re-derive the algorithm in a more systematic fashion with the help of index functions.

4.1. Notation and Definitions

Let $\phi_k(\lambda)$ be defined by (1.2). If expressed in terms of the power basis

$$\phi_k(\lambda) = c_k^{(k)}\lambda^k + \cdots + c_1^{(k)}\lambda + c_0^{(k)}, \tag{4.1}$$

it is clear that $c_k^{(k)} = (-1)^k\omega_1\omega_2\cdots\omega_k$ and $c_0^{(k)} = 1$. Thus,

$$c_k^{(k)} = -\omega_k c_{k-1}^{(k-1)}. \tag{4.2}$$

In ML(n)BiCGStab, we construct the following vectors: for $k \in \mathbb{N}$,

$$\mathbf{r}_k = \phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{r}}_k, \qquad \mathbf{u}_k = \phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{r}}_k, \tag{4.3a}$$

$$\mathbf{g}_k = \phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{g}}_k, \qquad \mathbf{d}_k = -\omega_{g_n(k)+1}\mathbf{A}\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{g}}_k, \tag{4.3b}$$

$$\mathbf{w}_k = \mathbf{A}\mathbf{g}_k \tag{4.3c}$$

and for $k = 0$, set

$$\mathbf{r}_0 = \widehat{\mathbf{r}}_0 \quad \text{and} \quad \mathbf{g}_0 = \widehat{\mathbf{g}}_0. \tag{4.4}$$

The vector $\mathbf{r}_k$ will be the residual vector of the approximate solution $\mathbf{x}_k$ computed by ML(n)BiCGStab.

4.2. Algorithm Derivation

The derivation parallels the one of BiCGStab from BiCG. We first replace all the inner products $\mathbf{p}^H\widehat{\mathbf{r}}$ and $\mathbf{p}^H\mathbf{A}\widehat{\mathbf{g}}$ in ML(n)BiCG respectively by the inner products of the forms $\mathbf{q}^H\phi(\mathbf{A})\widehat{\mathbf{r}}$ and $\mathbf{q}^H\mathbf{A}\phi(\mathbf{A})\widehat{\mathbf{g}}$, where ϕ is the polynomial (1.2). Corollary 3.1 guarantees that the inner products remain unchanged with such replacements. Then we compile the recurrences for the new residuals $\mathbf{r}_k$ and the corresponding iterates. The overall derivation is best described and verified in stages, and depends on Proposition 2.1 and Corollary 3.1.

The derivation is complicated by the fact that the recurrences in the kth iteration of ML(n)BiCG involve n terms which stretch from $k-n$ to $k-1$. Note that $k-n \le g_n(k)n \le k-1$. The degrees of the $\phi_{g_n(s)}$ and $\phi_{g_n(s)+1}$ in (4.3) are increased at $g_n(k)n+1$ as s runs from $k-n$ to $k-1$ (see, for example, Table 1). Therefore, our first task is to split up in ML(n)BiCG the loops and the sums of length n into two parts, one from $k-n$ to $g_n(k)n-1$ and the other from $g_n(k)n+1$ to $k-1$. The following Derivation Stage (DS) #1 is computationally equivalent to Algorithm 3.1 (forgetting Lines 1, 2, 5 and 11).

Derivation Stage #1.

1. For $k = 1, 2, \cdots$, until convergence:
2. If $r_n(k) = 1$
3. $\alpha_k = \mathbf{p}_k^H\widehat{\mathbf{r}}_{k-1}/\mathbf{p}_k^H\mathbf{A}\widehat{\mathbf{g}}_{k-1}$;
4. $\widehat{\mathbf{r}}_k = \widehat{\mathbf{r}}_{k-1} - \alpha_k\mathbf{A}\widehat{\mathbf{g}}_{k-1}$;
5. Else
6. $\alpha_k = \mathbf{p}_k^H\widehat{\mathbf{r}}_{k-1}/\mathbf{p}_k^H\mathbf{A}\widehat{\mathbf{g}}_{k-1}$;
7. $\widehat{\mathbf{r}}_k = \widehat{\mathbf{r}}_{k-1} - \alpha_k\mathbf{A}\widehat{\mathbf{g}}_{k-1}$;
8. End
9. If $r_n(k) < n$
10. For $s = \max(k-n, 0), \cdots, g_n(k)n-1$
11. $\beta_s^{(k)} = -\mathbf{p}_{s+1}^H\mathbf{A}\left(\widehat{\mathbf{r}}_k + \sum_{t=\max(k-n,0)}^{s-1}\beta_t^{(k)}\widehat{\mathbf{g}}_t\right)/\mathbf{p}_{s+1}^H\mathbf{A}\widehat{\mathbf{g}}_s$;
12. End
13. $\beta_{g_n(k)n}^{(k)} = -\mathbf{p}_{g_n(k)n+1}^H\mathbf{A}\left(\widehat{\mathbf{r}}_k + \sum_{t=\max(k-n,0)}^{g_n(k)n-1}\beta_t^{(k)}\widehat{\mathbf{g}}_t\right)/\mathbf{p}_{g_n(k)n+1}^H\mathbf{A}\widehat{\mathbf{g}}_{g_n(k)n}$;
14. For $s = g_n(k)n+1, \cdots, k-1$
15. $\beta_s^{(k)} = -\mathbf{p}_{s+1}^H\mathbf{A}\left(\widehat{\mathbf{r}}_k + \sum_{t=\max(k-n,0)}^{g_n(k)n}\beta_t^{(k)}\widehat{\mathbf{g}}_t + \sum_{t=g_n(k)n+1}^{s-1}\beta_t^{(k)}\widehat{\mathbf{g}}_t\right)/\mathbf{p}_{s+1}^H\mathbf{A}\widehat{\mathbf{g}}_s$;
16. End
17. $\widehat{\mathbf{g}}_k = \widehat{\mathbf{r}}_k + \sum_{s=\max(k-n,0)}^{g_n(k)n}\beta_s^{(k)}\widehat{\mathbf{g}}_s + \sum_{s=g_n(k)n+1}^{k-1}\beta_s^{(k)}\widehat{\mathbf{g}}_s$;
18. Else

19. $\beta^{(k)}_{g_n(k)n} = -\mathbf{p}^H_{g_n(k)n+1}\mathbf{A}\widehat{\mathbf{r}}_k / \mathbf{p}^H_{g_n(k)n+1}\mathbf{A}\widehat{\mathbf{g}}_{g_n(k)n}$;
20. For $s = g_n(k)n+1, \cdots, k-1$
21. $\beta^{(k)}_s = -\mathbf{p}^H_{s+1}\mathbf{A}\left(\widehat{\mathbf{r}}_k + \beta^{(k)}_{g_n(k)n}\widehat{\mathbf{g}}_{g_n(k)n} + \sum_{t=g_n(k)n+1}^{s-1}\beta^{(k)}_t\widehat{\mathbf{g}}_t\right)/\mathbf{p}^H_{s+1}\mathbf{A}\widehat{\mathbf{g}}_s$;
22. End
23. $\widehat{\mathbf{g}}_k = \widehat{\mathbf{r}}_k + \beta^{(k)}_{g_n(k)n}\widehat{\mathbf{g}}_{g_n(k)n} + \sum_{s=g_n(k)n+1}^{k-1}\beta^{(k)}_s\widehat{\mathbf{g}}_s$;
24. End
25. End

We have adopted the conventions: *empty loops are skipped and empty sums are zero*. These conventions will also be applied in the sequel.

In the next stage of the derivation, we replace inner products $\mathbf{p}^H\widehat{\mathbf{r}}$ and $\mathbf{p}^H\mathbf{A}\widehat{\mathbf{g}}$ by inner products of the forms $\mathbf{q}^H\phi(\mathbf{A})\widehat{\mathbf{r}}$ and $\mathbf{q}^H\mathbf{A}\phi(\mathbf{A})\widehat{\mathbf{g}}$ respectively. That is, the factor $(\mathbf{A}^H)^{g_n(k)}$ that is hidden in the left basis vector $\mathbf{p}_k$ is moved to the right-hand side space and replaced by the factor $\phi_{g_n(k)}(\mathbf{A})$. Formally, by Corollary 3.1 together with (3.1), (4.2) and Proposition 2.1(a), DS#1 can be further transformed into the version below. Explanations are given after listing.

Derivation Stage #2.

1. For $k = 1, 2, \cdots$, until convergence:
2. If $r_n(k) = 1$
3. $\alpha_k = \mathbf{q}^H_{r_n(k)}\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{r}}_{k-1} / \mathbf{q}^H_{r_n(k)}\mathbf{A}\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{g}}_{k-1}$;
4. $\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{r}}_k = \phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{r}}_{k-1} - \alpha_k\mathbf{A}\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{g}}_{k-1}$;
5. $\phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{r}}_k = (\mathbf{I} - \omega_{g_n(k)+1}\mathbf{A})\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{r}}_k$;
6. Else
7. $\alpha_k = \mathbf{q}^H_{r_n(k)}\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{r}}_{k-1} / \mathbf{q}^H_{r_n(k)}\mathbf{A}\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{g}}_{k-1}$;
8. $\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{r}}_k = \phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{r}}_{k-1} - \alpha_k\mathbf{A}\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{g}}_{k-1}$;
9. $\phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{r}}_k = \phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{r}}_{k-1} - \alpha_k\mathbf{A}\phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{g}}_{k-1}$;
10. End
11. If $r_n(k) < n$
12. For $s = \max(k-n, 0), \cdots, g_n(k)n-1$
13. $\beta^{(k)}_s = \mathbf{q}^H_{r_n(s+1)}\left(\phi_{g_n(s+1)+1}(\mathbf{A})\widehat{\mathbf{r}}_k - \sum_{t=\max(k-n,0)}^{s-1}\beta^{(k)}_t\omega_{g_n(s+1)+1}\right.$
$\left.\mathbf{A}\phi_{g_n(s+1)}(\mathbf{A})\widehat{\mathbf{g}}_t\right)/\omega_{g_n(s+1)+1}\mathbf{q}^H_{r_n(s+1)}\mathbf{A}\phi_{g_n(s+1)}(\mathbf{A})\widehat{\mathbf{g}}_s$;
14. End
15. $\beta^{(k)}_{g_n(k)n} = \mathbf{q}^H_1\left(\phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{r}}_k - \sum_{t=\max(k-n,0)}^{g_n(k)n-1}\beta^{(k)}_t\omega_{g_n(k)+1}\right.$
$\left.\mathbf{A}\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{g}}_t\right)/\omega_{g_n(k)+1}\mathbf{q}^H_1\mathbf{A}\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{g}}_{g_n(k)n}$;
16. For $s = g_n(k)n+1, \cdots, k-1$
17. $\beta^{(k)}_s = \mathbf{q}^H_{r_n(s+1)}\left(\phi_{g_n(s+1)+1}(\mathbf{A})\widehat{\mathbf{r}}_k - \sum_{t=\max(k-n,0)}^{g_n(k)n}\beta^{(k)}_t\omega_{g_n(s+1)+1}\right.$
$\left.\mathbf{A}\phi_{g_n(s+1)}(\mathbf{A})\widehat{\mathbf{g}}_t - \sum_{t=g_n(k)n+1}^{s-1}\beta^{(k)}_t\omega_{g_n(s+1)+1}\right.$

$\mathbf{A}\phi_{g_n(s+1)}(\mathbf{A})\widehat{\mathbf{g}}_t\big)/\omega_{g_n(s+1)+1}\mathbf{q}_{r_n(s+1)}^H\mathbf{A}\phi_{g_n(s+1)}(\mathbf{A})\widehat{\mathbf{g}}_s$;

18. End

19. $\omega_{g_n(k)+1}\mathbf{A}\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{g}}_k = \omega_{g_n(k)+1}\mathbf{A}\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{r}}_k$
$+\sum_{s=\max(k-n,0)}^{g_n(k)n}\beta_s^{(k)}\omega_{g_n(k)+1}\mathbf{A}\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{g}}_s + \sum_{s=g_n(k)n+1}^{k-1}$
$\beta_s^{(k)}\omega_{g_n(k)+1}\mathbf{A}\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{g}}_s$;

20. $\phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{g}}_k = \phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{r}}_k + \sum_{s=\max(k-n,0)}^{g_n(k)n}\beta_s^{(k)}\phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{g}}_s$
$+\sum_{s=g_n(k)n+1}^{k-1}\beta_s^{(k)}\phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{g}}_s$;

21. Else

22. $\beta_{g_n(k)n}^{(k)} = \mathbf{q}_1^H\phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{r}}_k/\omega_{g_n(k)+1}\mathbf{q}_1^H\mathbf{A}\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{g}}_{g_n(k)n}$;

23. For $s = g_n(k)n+1,\cdots,k-1$

24. $\beta_s^{(k)} = \mathbf{q}_{r_n(s+1)}^H\Big(\phi_{g_n(s+1)+1}(\mathbf{A})\widehat{\mathbf{r}}_k - \beta_{g_n(k)n}^{(k)}\omega_{g_n(s+1)+1}\mathbf{A}\phi_{g_n(s+1)}$
$(\mathbf{A})\widehat{\mathbf{g}}_{g_n(k)n} - \sum_{t=g_n(k)n+1}^{s-1}\beta_t^{(k)}\omega_{g_n(s+1)+1}\mathbf{A}\phi_{g_n(s+1)}$
$(\mathbf{A})\widehat{\mathbf{g}}_t\Big)/\omega_{g_n(s+1)+1}\mathbf{q}_{r_n(s+1)}^H\mathbf{A}\phi_{g_n(s+1)}(\mathbf{A})\widehat{\mathbf{g}}_s$;

25. End

26. $\phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{g}}_k = \phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{r}}_k + \beta_{g_n(k)n}^{(k)}\phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{g}}_{g_n(k)n}$
$+\sum_{s=g_n(k)n+1}^{k-1}\beta_s^{(k)}\phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{g}}_s$;

27. End

28. End

Lines 4, 8, 9, 19, 20 and 26, DS#2, were obtained from Lines 4, 7, 17 and 23, DS#1, through a multiplication by $\phi_{g_n(k)}(\mathbf{A}), \phi_{g_n(k)+1}(\mathbf{A})$ and $\omega_{g_n(k)+1}\mathbf{A}\phi_{g_n(k)}(\mathbf{A})$ respectively. Line 5, DS#2, is a direct result of the definition (1.2) of ϕ. These lines are prepared for the updates of the vectors defined in (4.3).

To help understand how DS#1 is turned into DS#2, let us demonstrate (i) the transformation of Line 3, DS#1, into Line 3, DS#2 and (ii) the transformation of the term $\mathbf{p}_{g_n(k)n+1}^H\mathbf{A}\widehat{\mathbf{r}}_k$ in Line 13, DS#1, into the term $\mathbf{q}_1^H\phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{r}}_k$ in Line 15, DS#2, as follows.

(i) By Corollary 3.1,

$$\alpha_k = \frac{\mathbf{p}_k^H\widehat{\mathbf{r}}_{k-1}}{\mathbf{p}_k^H\mathbf{A}\widehat{\mathbf{g}}_{k-1}} = \frac{\frac{1}{c_{g_n(k)}^{(g_n(k))}}\mathbf{q}_{r_n(k)}^H\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{r}}_{k-1}}{\frac{1}{c_{g_n(k)}^{(g_n(k))}}\mathbf{q}_{r_n(k)}^H\mathbf{A}\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{g}}_{k-1}} = \frac{\mathbf{q}_{r_n(k)}^H\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{r}}_{k-1}}{\mathbf{q}_{r_n(k)}^H\mathbf{A}\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{g}}_{k-1}},$$

where $c_{g_n(k)}^{(g_n(k))}$ is the leading coefficient of $\phi_{g_n(k)}(\lambda)$ (see (4.1)).

(ii) By (3.1) and Proposition 2.1(a), we have

$$\begin{aligned}\mathbf{A}^H\mathbf{p}_{g_n(k)n+1} &= (\mathbf{A}^H)^{g_n(g_n(k)n+1)+1}\mathbf{q}_{r_n(g_n(k)n+1)}\\ &= (\mathbf{A}^H)^{g_n((g_n(k)+1)n+1)}\mathbf{q}_{r_n((g_n(k)+1)n+1)}\\ &= \mathbf{p}_{(g_n(k)+1)n+1}.\end{aligned}$$

Hence

$$\mathbf{p}_{g_n(k)n+1}^H \mathbf{A}\widehat{\mathbf{r}}_k = \mathbf{p}_{(g_n(k)+1)n+1}^H \widehat{\mathbf{r}}_k.$$

Since $(g_n(k)+1)n+1 \le k+n$, an application of Corollary 3.1 to $\mathbf{p}_{(g_n(k)+1)n+1}^H \widehat{\mathbf{r}}_k$ thus yields

$$\begin{aligned}\mathbf{p}_{g_n(k)n+1}^H \mathbf{A}\widehat{\mathbf{r}}_k &= \frac{1}{c_{g_n((g_n(k)+1)n+1)}^{(g_n((g_n(k)+1)n+1))}} \mathbf{q}_{r_n((g_n(k)+1)n+1)}^H \phi_{g_n((g_n(k)+1)n+1)}(\mathbf{A})\widehat{\mathbf{r}}_k \\ &= \frac{1}{c_{g_n(k)+1}^{(g_n(k)+1)}} \mathbf{q}_1^H \phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{r}}_k.\end{aligned}$$

The second equation above follows from (2.2). The coefficient $1/c_{g_n(k)+1}^{(g_n(k)+1)}$ is missed from Line 15, DS#2, because it was canceled out by the coefficient from the denominator.

Our goal is to establish updating relations for the quantities introduced in (4.3). To this end, we further transform DS#2 into the following version. This time, we work on the index function g_n with the aid of Proposition 2.1 so that the definitions in (4.3) can be applied. Again, further explanations are given after the listing.

Derivation Stage #3.

1. For $k = 1, 2, \cdots$, until convergence:
2. If $r_n(k) = 1$
3. $\alpha_k = \mathbf{q}_{r_n(k)}^H \phi_{g_n(k-1)+1}(\mathbf{A})\widehat{\mathbf{r}}_{k-1} / \mathbf{q}_{r_n(k)}^H \mathbf{A}\phi_{g_n(k-1)+1}(\mathbf{A})\widehat{\mathbf{g}}_{k-1}$;
4. $\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{r}}_k = \phi_{g_n(k-1)+1}(\mathbf{A})\widehat{\mathbf{r}}_{k-1} - \alpha_k \mathbf{A}\phi_{g_n(k-1)+1}(\mathbf{A})\widehat{\mathbf{g}}_{k-1}$;
5. $\phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{r}}_k = (\mathbf{I} - \omega_{g_n(k)+1}\mathbf{A})\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{r}}_k$;
6. Else
7. $\alpha_k = \mathbf{q}_{r_n(k)}^H \phi_{g_n(k-1)}(\mathbf{A})\widehat{\mathbf{r}}_{k-1} / \mathbf{q}_{r_n(k)}^H \mathbf{A}\phi_{g_n(k-1)}(\mathbf{A})\widehat{\mathbf{g}}_{k-1}$;
8. $\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{r}}_k = \phi_{g_n(k-1)}(\mathbf{A})\widehat{\mathbf{r}}_{k-1} - \alpha_k \mathbf{A}\phi_{g_n(k-1)}(\mathbf{A})\widehat{\mathbf{g}}_{k-1}$;
9. $\phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{r}}_k = \phi_{g_n(k-1)+1}(\mathbf{A})\widehat{\mathbf{r}}_{k-1} - \alpha_k \mathbf{A}\phi_{g_n(k-1)+1}(\mathbf{A})\widehat{\mathbf{g}}_{k-1}$;
10. End
11. If $r_n(k) < n$
12. For $s = \max(k-n, 0), \cdots, g_n(k)n - 1$
13. $\beta_s^{(k)} = \mathbf{q}_{r_n(s+1)}^H \Big(\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{r}}_k - \sum_{t=\max(k-n,0)}^{s-1} \beta_t^{(k)} \omega_{g_n(t)+1} \mathbf{A}\phi_{g_n(t)}(\mathbf{A})\widehat{\mathbf{g}}_t \Big) / \omega_{g_n(s)+1} \mathbf{q}_{r_n(s+1)}^H \mathbf{A}\phi_{g_n(s)}(\mathbf{A})\widehat{\mathbf{g}}_s$;
14. End
15. $\beta_{g_n(k)n}^{(k)} = \mathbf{q}_1^H \Big(\phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{r}}_k - \sum_{t=\max(k-n,0)}^{g_n(k)n-1} \beta_t^{(k)} \omega_{g_n(k)+1} \mathbf{A}\phi_{g_n(t)+1}(\mathbf{A})\widehat{\mathbf{g}}_t \Big) / \omega_{g_n(k)+1} \mathbf{q}_1^H \mathbf{A}\phi_{g_n(g_n(k)n)+1}(\mathbf{A})\widehat{\mathbf{g}}_{g_n(k)n}$;
16. For $s = g_n(k)n + 1, \cdots, k-1$
17. $\beta_s^{(k)} = \mathbf{q}_{r_n(s+1)}^H \Big(\phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{r}}_k - \sum_{t=\max(k-n,0)}^{g_n(k)n} \beta_t^{(k)} \omega_{g_n(k)+1}$

$$\mathbf{A}\phi_{g_n(t)+1}(\mathbf{A})\widehat{\mathbf{g}}_t - \sum_{t=g_n(k)n+1}^{s-1}\beta_t^{(k)}\omega_{g_n(t)+1}$$
$$\mathbf{A}\phi_{g_n(t)}(\mathbf{A})\widehat{\mathbf{g}}_t\Big)/\omega_{g_n(s)+1}\mathbf{q}_{r_n(s+1)}^H\mathbf{A}\phi_{g_n(s)}(\mathbf{A})\widehat{\mathbf{g}}_s;$$

18. End

19. $\omega_{g_n(k)+1}\mathbf{A}\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{g}}_k = \omega_{g_n(k)+1}\mathbf{A}\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{r}}_k$
$+\sum_{s=\max(k-n,0)}^{g_n(k)n}\beta_s^{(k)}\omega_{g_n(k)+1}\mathbf{A}\phi_{g_n(s)+1}(\mathbf{A})\widehat{\mathbf{g}}_s$
$+\sum_{s=g_n(k)n+1}^{k-1}\beta_s^{(k)}\omega_{g_n(s)+1}\mathbf{A}\phi_{g_n(s)}(\mathbf{A})\widehat{\mathbf{g}}_s;$

20. $\phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{g}}_k = \phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{r}}_k + \sum_{s=\max(k-n,0)}^{g_n(k)n}\beta_s^{(k)}(\mathbf{I}-\omega_{g_n(k)+1}\mathbf{A})\phi_{g_n(s)+1}$
$(\mathbf{A})\widehat{\mathbf{g}}_s + \sum_{s=g_n(k)n+1}^{k-1}\beta_s^{(k)}\phi_{g_n(s)+1}(\mathbf{A})\widehat{\mathbf{g}}_s;$

21. Else

22. $\beta_{g_n(k)n}^{(k)} = \mathbf{q}_1^H\phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{r}}_k/\omega_{g_n(k)+1}\mathbf{q}_1^H\mathbf{A}\phi_{g_n(g_n(k)n)+1}(\mathbf{A})\widehat{\mathbf{g}}_{g_n(k)n};$

23. For $s = g_n(k)n+1,\cdots,k-1$

24. $\beta_s^{(k)} = \mathbf{q}_{r_n(s+1)}^H\Big(\phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{r}}_k - \beta_{g_n(k)n}^{(k)}\omega_{g_n(k)+1}\mathbf{A}\phi_{g_n(g_n(k)n)+1}(\mathbf{A})\widehat{\mathbf{g}}_{g_n(k)n}$
$-\sum_{t=g_n(k)n+1}^{s-1}\beta_t^{(k)}\omega_{g_n(t)+1}\mathbf{A}\phi_{g_n(t)}(\mathbf{A})\widehat{\mathbf{g}}_t\Big)/\omega_{g_n(s)+1}$
$\mathbf{q}_{r_n(s+1)}^H\mathbf{A}\phi_{g_n(s)}(\mathbf{A})\widehat{\mathbf{g}}_s;$

25. End

26. $\phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{g}}_k = \phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{r}}_k + \beta_{g_n(k)n}^{(k)}(\mathbf{I}-\omega_{g_n(k)+1}\mathbf{A})\phi_{g_n(g_n(k)n)+1}(\mathbf{A})\widehat{\mathbf{g}}_{g_n(k)n}$
$+\sum_{s=g_n(k)n+1}^{k-1}\beta_s^{(k)}\phi_{g_n(s)+1}(\mathbf{A})\widehat{\mathbf{g}}_s;$

27. End

28. End

As an example, let us show how the $g_n(s+1)$ inside the sum $\sum_{t=\max(k-n,0)}^{s-1}\cdots$ in Line 13, DS#2, was written as the $g_n(t)$ in Line 13, DS#3.

If $g_n(k)=0$, Line 13 of DS#2 is not implemented because of the conventions immediately following DS#1. So, we assume that $g_n(k)>0$. Since

$$\max(k-n,0)\le s,\ t\le g_n(k)n-1, \tag{4.5}$$

we have

$$g_n(s+1) = g_n(k+1)-1 = g_n(t+1) \tag{4.6}$$

by Proposition 2.1(b). Now that $g_n(k)>0$, $\max(k-n,0)=k-n$. Hence

$$k-n\le t\le g_n(k)n-1 \tag{4.7}$$

by (4.5). Let $k=jn+i$ as in (2.1). Then (4.7) is

$$(j-1)n+i\le t\le (j-1)n+n-1$$

which implies that $r_n(t)<n$. Now, Proposition 2.1(d) yields $g_n(t+1)=g_n(t)$ and therefore we have $g_n(s+1)=g_n(t)$ by (4.6).

We are now ready to apply the vectors defined in (4.3) and (4.4). Substituting them into DS#3 leads to the following stage.

Derivation Stage #4.

1. For $k = 1, 2, \cdots$, until convergence:
2. If $r_n(k) = 1$
3. $\alpha_k = \mathbf{q}_{r_n(k)}^H \mathbf{r}_{k-1} / \mathbf{q}_{r_n(k)}^H \mathbf{A}\mathbf{g}_{k-1}$;
4. $\mathbf{u}_k = \mathbf{r}_{k-1} - \alpha_k \mathbf{A}\mathbf{g}_{k-1}$;
5. $\mathbf{r}_k = -\omega_{g_n(k)+1} \mathbf{A}\mathbf{u}_k + \mathbf{u}_k$;
6. Else
7. $\alpha_k = -\omega_{g_n(k-1)+1} \mathbf{q}_{r_n(k)}^H \mathbf{u}_{k-1} / \mathbf{q}_{r_n(k)}^H \mathbf{d}_{k-1}$;
8. $\mathbf{u}_k = \mathbf{u}_{k-1} + (\alpha_k / \omega_{g_n(k-1)+1}) \mathbf{d}_{k-1}$;
9. $\mathbf{r}_k = \mathbf{r}_{k-1} - \alpha_k \mathbf{A}\mathbf{g}_{k-1}$;
10. End
11. If $r_n(k) < n$
12. For $s = \max(k-n, 0), \cdots, g_n(k)n - 1$
13. $\beta_s^{(k)} = -\mathbf{q}_{r_n(s+1)}^H \left(\mathbf{u}_k + \sum_{t=\max(k-n,0)}^{s-1} \beta_t^{(k)} \mathbf{d}_t \right) / \mathbf{q}_{r_n(s+1)}^H \mathbf{d}_s$;
14. End
15. $\beta_{g_n(k)n}^{(k)} = \mathbf{q}_1^H \left(\mathbf{r}_k - \omega_{g_n(k)+1} \sum_{t=\max(k-n,0)}^{g_n(k)n-1} \beta_t^{(k)} \mathbf{A}\mathbf{g}_t \right) / \omega_{g_n(k)+1} \mathbf{q}_1^H \mathbf{A}\mathbf{g}_{g_n(k)n}$;
16. For $s = g_n(k)n + 1, \cdots, k-1$
17. $\beta_s^{(k)} = -\mathbf{q}_{r_n(s+1)}^H \left(\mathbf{r}_k - \omega_{g_n(k)+1} \sum_{t=\max(k-n,0)}^{g_n(k)n} \beta_t^{(k)} \mathbf{A}\mathbf{g}_t + \sum_{t=g_n(k)n+1}^{s-1} \beta_t^{(k)} \mathbf{d}_t \right) / \mathbf{q}_{r_n(s+1)}^H \mathbf{d}_s$;
18. End
19. $\mathbf{d}_k = \mathbf{r}_k - \mathbf{u}_k - \omega_{g_n(k)+1} \sum_{s=\max(k-n,0)}^{g_n(k)n} \beta_s^{(k)} \mathbf{A}\mathbf{g}_s + \sum_{s=g_n(k)n+1}^{k-1} \beta_s^{(k)} \mathbf{d}_s$;
20. $\mathbf{g}_k = \mathbf{r}_k + \sum_{s=\max(k-n,0)}^{g_n(k)n} \beta_s^{(k)} (\mathbf{I} - \omega_{g_n(k)+1} \mathbf{A}) \mathbf{g}_s + \sum_{s=g_n(k)n+1}^{k-1} \beta_s^{(k)} \mathbf{g}_s$;
21. Else
22. $\beta_{g_n(k)n}^{(k)} = \mathbf{q}_1^H \mathbf{r}_k / \omega_{g_n(k)+1} \mathbf{q}_1^H \mathbf{A}\mathbf{g}_{g_n(k)n}$;
23. For $s = g_n(k)n + 1, \cdots, k-1$
24. $\beta_s^{(k)} = -\mathbf{q}_{r_n(s+1)}^H \left(\mathbf{r}_k - \omega_{g_n(k)+1} \beta_{g_n(k)n}^{(k)} \mathbf{A}\mathbf{g}_{g_n(k)n} + \sum_{t=g_n(k)n+1}^{s-1} \beta_t^{(k)} \mathbf{d}_t \right) / \mathbf{q}_{r_n(s+1)}^H \mathbf{d}_s$;
25. End
26. $\mathbf{g}_k = \mathbf{r}_k + \beta_{g_n(k)n}^{(k)} (\mathbf{I} - \omega_{g_n(k)+1} \mathbf{A}) \mathbf{g}_{g_n(k)n} + \sum_{s=g_n(k)n+1}^{k-1} \beta_s^{(k)} \mathbf{g}_s$;
27. End
28. End

We consider $\mathbf{r}_k$ to be the residual of the kth approximate solution $\mathbf{x}_k$. Updating relations for $\mathbf{x}_k$ can be obtained from Lines 4, 5 and 9 respectively:

$$\mathbf{x}_k = \begin{cases} \mathbf{x}_{k-1} + \omega_{g_n(k)+1} \mathbf{u}_k + \alpha_k \mathbf{g}_{k-1}, & \text{if } r_n(k) = 1, \\ \mathbf{x}_{k-1} + \alpha_k \mathbf{g}_{k-1}, & \text{if } r_n(k) > 1. \end{cases} \tag{4.8}$$

After adding (4.8) to DS#4 and simplifying the operations appropriately, we arrive at the following ML(n)BiCGStab algorithm. Just like BiCGStab, the free parameter $\omega_{g_n(k)+1}$ in Line 5, DS#4, is chosen to minimize the 2-norm of $\mathbf{r}_k$.

Algorithm 4.1. ML(n)BiCGStab without preconditioning associated with (4.3)

1. *Choose an initial guess* $\mathbf{x}_0$ *and n vectors* $\mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_n$.
2. *Compute* $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$ *and set* $\mathbf{g}_0 = \mathbf{r}_0$. *Compute* $\mathbf{w}_0 = \mathbf{A}\mathbf{g}_0,\ c_0 = \mathbf{q}_1^H \mathbf{w}_0$.
3. *For* $k = 1, 2, \cdots$, *until convergence:*
4. *If* $r_n(k) = 1$
5. $\alpha_k = \mathbf{q}_{r_n(k)}^H \mathbf{r}_{k-1} / c_{k-1}$;
6. $\mathbf{u}_k = \mathbf{r}_{k-1} - \alpha_k \mathbf{w}_{k-1}$;
7. $\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k \mathbf{g}_{k-1}$;
8. $\omega_{g_n(k)+1} = (\mathbf{A}\mathbf{u}_k)^H \mathbf{u}_k / \|\mathbf{A}\mathbf{u}_k\|_2^2$;
9. $\mathbf{x}_k = \mathbf{x}_k + \omega_{g_n(k)+1} \mathbf{u}_k$;
10. $\mathbf{r}_k = -\omega_{g_n(k)+1} \mathbf{A}\mathbf{u}_k + \mathbf{u}_k$;
11. *Else*
12. $\widetilde{\alpha}_k = -\mathbf{q}_{r_n(k)}^H \mathbf{u}_{k-1} / c_{k-1}$; % $\widetilde{\alpha}_k = \alpha_k / \omega_{g_n(k-1)+1}$
13. *If* $r_n(k) < n$
14. $\mathbf{u}_k = \mathbf{u}_{k-1} + \widetilde{\alpha}_k \mathbf{d}_{k-1}$;
15. *End*
16. $\mathbf{x}_k = \mathbf{x}_{k-1} + \omega_{g_n(k-1)+1} \widetilde{\alpha}_k \mathbf{g}_{k-1}$;
17. $\mathbf{r}_k = \mathbf{r}_{k-1} - \omega_{g_n(k-1)+1} \widetilde{\alpha}_k \mathbf{w}_{k-1}$;
18. *End*
19. *If* $r_n(k) < n$
20. $\mathbf{z}_d = \mathbf{u}_k,\ \mathbf{g}_k = \mathbf{0},\ \mathbf{z}_w = \mathbf{0}$;
21. *For* $s = k - n, \cdots, g_n(k)n - 1$ *and* $g_n(k) \geq 1$
22. $\beta_s^{(k)} = -\mathbf{q}_{r_n(s+1)}^H \mathbf{z}_d / c_s$;
23. $\mathbf{z}_d = \mathbf{z}_d + \beta_s^{(k)} \mathbf{d}_s$;
24. $\mathbf{g}_k = \mathbf{g}_k + \beta_s^{(k)} \mathbf{g}_s$;
25. $\mathbf{z}_w = \mathbf{z}_w + \beta_s^{(k)} \mathbf{w}_s$;
26. *End*
27. $\mathbf{z}_w = \mathbf{r}_k - \omega_{g_n(k)+1} \mathbf{z}_w$;
28. $\beta_{g_n(k)n}^{(k)} = \mathbf{q}_1^H \mathbf{z}_w / (\omega_{g_n(k)+1} c_{g_n(k)n})$;
29. $\mathbf{z}_w = \mathbf{z}_w - \omega_{g_n(k)+1} \beta_{g_n(k)n}^{(k)} \mathbf{w}_{g_n(k)n}$;
30. $\mathbf{g}_k = \mathbf{g}_k + \mathbf{z}_w + \beta_{g_n(k)n}^{(k)} \mathbf{g}_{g_n(k)n}$;
31. *For* $s = g_n(k)n + 1, \cdots, k - 1$
32. $\beta_s^{(k)} = -\mathbf{q}_{r_n(s+1)}^H \mathbf{z}_w / c_s$;
33. $\mathbf{g}_k = \mathbf{g}_k + \beta_s^{(k)} \mathbf{g}_s$;
34. $\mathbf{z}_w = \mathbf{z}_w + \beta_s^{(k)} \mathbf{d}_s$;
35. *End*
36. $\mathbf{d}_k = \mathbf{z}_w - \mathbf{u}_k;\ \ c_k = \mathbf{q}_{r_n(k+1)}^H \mathbf{d}_k$;
37. $\mathbf{w}_k = \mathbf{A}\mathbf{g}_k$;
38. *Else*
39. $\beta_{g_n(k)n}^{(k)} = \mathbf{q}_1^H \mathbf{r}_k / (\omega_{g_n(k)+1} c_{g_n(k)n})$;
40. $\mathbf{z}_w = \mathbf{r}_k - \omega_{g_n(k)+1} \beta_{g_n(k)n}^{(k)} \mathbf{w}_{g_n(k)n}$;
41. $\mathbf{g}_k = \mathbf{z}_w + \beta_{g_n(k)n}^{(k)} \mathbf{g}_{g_n(k)n}$;
42. *For* $s = g_n(k)n + 1, \cdots, k - 1$

43. $\beta_s^{(k)} = -\mathbf{q}_{r_n(s+1)}^H \mathbf{z}_w / c_s$;
44. $\mathbf{g}_k = \mathbf{g}_k + \beta_s^{(k)} \mathbf{g}_s$;
45. $\mathbf{z}_w = \mathbf{z}_w + \beta_s^{(k)} \mathbf{d}_s$;
46. *End*
47. $\mathbf{w}_k = \mathbf{A}\mathbf{g}_k$; $c_k = \mathbf{q}_{r_n(k+1)}^H \mathbf{w}_k$;
48. *End*
49. *End*

Remark 4.1.

(i) Algorithm 4.1 does not compute the quantities $\mathbf{u}_k$ and $\mathbf{d}_k$ when $r_n(k) = n$ (see Lines 13-15 and Lines 39-47).

(ii) If the $\mathbf{u}_k$ in Line 6 happens to be zero, then the $\omega_{g_n(k)+1}$ in Line 8 and therefore the $\mathbf{x}_k$ and $\mathbf{r}_k$ in Lines 9 and 10 will not be computable. In this case, however, the $\mathbf{x}_k$ in Line 7 will be an exact solution to system (1.1) and Algorithm 4.1 stops there.

We now compare Algorithm 4.1 with the ML(n)BiCGStab algorithm in [39]. First, the definitions of $\mathbf{r}_k$, $\mathbf{u}_k$ and $\mathbf{g}_k$ are the same in both algorithms, but $\mathbf{d}_k$ is defined differently. In [39], $\mathbf{d}_k = \phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{g}}_k$. In exact arithmetic, however, both algorithms compute the same $\mathbf{r}_k$ and $\mathbf{x}_k$. Second, the derivation of Algorithm 4.1 has been made simpler by using index functions. As a result, some redundant operations in Algorithm 2 of [39] can be seen and removed and some arithmetics are simplified. For example, the vectors $\mathbf{d}_k, \mathbf{u}_k$ are computed in every k-iteration in Algorithm 2 of [39]. They are now computed only when $r_n(k) < n$. Also, the expression of $\beta_{g_n(k)n}^{(k)}$ in Line 39 of Algorithm 4.1 is simpler. Some other minor changes were also made so that the algorithm becomes more efficient.

Computational cost and storage requirement of Algorithm 4.1, obtained based on its preconditioned version, Algorithm 9.1 in §9, are summarized in Table 2. Since the vectors $\{\mathbf{q}_1, \ldots, \mathbf{q}_n\}$, $\{\mathbf{d}_{k-n}, \ldots, \mathbf{d}_{g_n(k)n-1}, \mathbf{d}_{g_n(k)n+1}, \ldots, \mathbf{d}_{k-1}\}$, $\{\mathbf{g}_{k-n}, \ldots, \mathbf{g}_{k-1}\}$ and $\{\mathbf{w}_{k-n}, \ldots, \mathbf{w}_{g_n(k)n}, \mathbf{w}_{k-1}\}$ are required in iteration k, they must be stored. When n is large, this storage is dominant. So, the storage requirement of the algorithm is about $4nN$.

Table 2: Average cost per k-iteration of Algorithm 9.1 and its storage requirement. This table does not count the costs in Lines 1-2 of the algorithm.

Preconditioning $\mathbf{M}^{-1}\mathbf{v}$	$1+1/n$	$\mathbf{u} \pm \mathbf{v}$, $\alpha\mathbf{v}$	$\max(4-5/n, 0)$
Matvec $\mathbf{A}\mathbf{v}$	$1+1/n$	Saxpy $\mathbf{u}+\alpha\mathbf{v}$	$\max(2.5n+0.5+1/n, 6)$
dot product $\mathbf{u}^H\mathbf{v}$	$n+1+2/n$	Storage	$\mathbf{A}+\mathbf{M}+(4n+4)N+O(n)$

4.3. Properties

We summarize the properties of Algorithm 4.1 in the following proposition. Since $\mathbf{r}_0 = \widehat{\mathbf{r}}_0$ by (4.4), ν (see §3.2) is also the degree of $p_{min}(\lambda; \mathbf{A}, \mathbf{r}_0)$.

Proposition 4.1. *Under the assumptions of Proposition* 3.1*, if* $\omega_{g_n(k)+1} \neq 0$ *and* $1/\omega_{g_n(k)+1} \notin \sigma(\mathbf{A})$ *for* $1 \leq k \leq \nu - 1$*, where* $\sigma(\mathbf{A})$ *is the spectrum of* $\mathbf{A}$*, then Algorithm* 4.1 *does not break down by zero division for* $k = 1, 2, \cdots, \nu$*, and* $\mathbf{x}_\nu$ *is an exact solution of* (1.1)*. Moreover, the computed quantities satisfy*

(a) $\mathbf{x}_k \in \mathbf{x}_0 + \mathbb{K}_{g_n(k)+k+1}(\mathbf{A}, \mathbf{r}_0)$ *and* $\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k \in \mathbf{r}_0 + \mathbf{A}\mathbb{K}_{g_n(k)+k+1}(\mathbf{A}, \mathbf{r}_0)$ *for* $1 \leq k \leq \nu - 1$.

(b) $\mathbf{r}_k \neq \mathbf{0}$ *for* $1 \leq k \leq \nu - 1$ *and* $\mathbf{r}_\nu = \mathbf{0}$.

(c) $\mathbf{r}_k \not\perp \mathbf{q}_1$ *for* $1 \leq k \leq \nu - 1$ *with* $r_n(k) = n$.

(d) $\mathbf{u}_k \perp span\{\mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_{r_n(k)}\}$ *and* $\mathbf{u}_k \not\perp \mathbf{q}_{r_n(k)+1}$ *for* $1 \leq k \leq \nu - 1$ *with* $r_n(k) < n$.

(e) $\mathbf{d}_k \perp span\{\mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_{r_n(k)}\}$ *and* $\mathbf{d}_k \not\perp \mathbf{q}_{r_n(k)+1}$ *for* $1 \leq k \leq \nu - 1$ *with* $r_n(k) < n$.

Proof. The divisors in Algorithm 4.1 are $c_k, \|\mathbf{A}\mathbf{u}_k\|_2^2$ and $\omega_{g_n(k)+1}$ respectively, where the ω's have been assumed to be nonzero. By Proposition 3.1(c), we have $\mathbf{A}\widehat{\mathbf{r}}_k \neq \mathbf{0}$ for $1 \leq k \leq \nu - 1$. Since $1/\omega \notin \sigma(\mathbf{A})$ by assumption, $\phi_{g_n(k)}(\mathbf{A})$ is nonsingular. Hence $\mathbf{A}\mathbf{u}_k = \phi_{g_n(k)}(\mathbf{A})\mathbf{A}\widehat{\mathbf{r}}_k \neq \mathbf{0}$ (see (4.3) for the first equation). Therefore, $\|\mathbf{A}\mathbf{u}_k\|_2 \neq 0$ for $1 \leq k \leq \nu - 1$.

c_k is defined respectively in Lines 36 and 47 in the algorithm. When $r_n(k) < n$, we have $c_k = \mathbf{q}_{r_n(k+1)}^H \mathbf{d}_k$. In this case,

$$\begin{aligned} c_k &= -\omega_{g_n(k)+1}\mathbf{q}_{r_n(k+1)}^H \mathbf{A}\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{g}}_k = -\omega_{g_n(k)+1}\mathbf{q}_{r_n(k+1)}^H \mathbf{A}\phi_{g_n(k+1)}(\mathbf{A})\widehat{\mathbf{g}}_k \\ &= -\omega_{g_n(k)+1} c_{g_n(k+1)}^{(g_n(k+1))} \mathbf{p}_{k+1}^H \mathbf{A}\widehat{\mathbf{g}}_k = -\omega_{g_n(k)+1} c_{g_n(k)}^{(g_n(k))} \mathbf{p}_{k+1}^H \mathbf{A}\widehat{\mathbf{g}}_k = c_{g_n(k)+1}^{(g_n(k)+1)} \mathbf{p}_{k+1}^H \mathbf{A}\widehat{\mathbf{g}}_k, \end{aligned}$$

by (4.3), Proposition 2.1(d), Corollary 3.1, (4.1) and (4.2). Since the ω's are nonzero by assumption and $\mathbf{p}_{k+1}^H \mathbf{A}\widehat{\mathbf{g}}_k \neq 0$ by Proposition 3.1(g), we have $c_{g_n(k)+1}^{(g_n(k)+1)} \neq 0$ by (4.2) and hence $c_k \neq 0$. When $r_n(k) = n$, on the other hand,

$$c_k = \mathbf{q}_{r_n(k+1)}^H \mathbf{w}_k = \mathbf{q}_{r_n(k+1)}^H \mathbf{A}\mathbf{g}_k.$$

In this case,

$$\begin{aligned} c_k &= \mathbf{q}_{r_n(k+1)}^H \mathbf{A}\phi_{g_n(k)+1}(\mathbf{A})\,\widehat{\mathbf{g}}_k = \mathbf{q}_{r_n(k+1)}^H \mathbf{A}\phi_{g_n(k+1)}(\mathbf{A})\,\widehat{\mathbf{g}}_k \\ &= c_{g_n(k+1)}^{(g_n(k+1))} \mathbf{p}_{k+1}^H \mathbf{A}\widehat{\mathbf{g}}_k = c_{g_n(k)+1}^{(g_n(k)+1)} \mathbf{p}_{k+1}^H \mathbf{A}\widehat{\mathbf{g}}_k \neq 0. \end{aligned}$$

Therefore, in either case, we always have $c_k \neq 0$ for $1 \leq k \leq \nu - 1$. Moreover, $c_0 = \mathbf{q}_1^H \mathbf{w}_0 = \mathbf{q}_1^H \mathbf{A}\mathbf{g}_0$ according to Line 2 of the algorithm. Since $\mathbf{p}_1 = \mathbf{q}_1$ by (3.1) and $\mathbf{g}_0 = \widehat{\mathbf{g}}_0$ by (4.4), $c_0 \neq 0$ by Proposition 3.1(g).

Now that $\|\mathbf{A}\mathbf{u}_k\|_2 \neq 0$, $\omega_{g_n(k)+1} \neq 0$ for $1 \leq k \leq \nu - 1$ and $c_k \neq 0$ for $0 \leq k \leq \nu - 1$, Algorithm 4.1 does not break down by zero division in the first $\nu - 1$ iterations. When $k = \nu$, $\mathbf{u}_k = \mathbf{u}_\nu = \phi_{g_n(\nu)}(\mathbf{A})\widehat{\mathbf{r}}_\nu = \mathbf{0}$ and $\mathbf{r}_k = \mathbf{r}_\nu = \phi_{g_n(\nu)+1}(\mathbf{A})\widehat{\mathbf{r}}_\nu = \mathbf{0}$ due to $\widehat{\mathbf{r}}_\nu = \mathbf{0}$ by Proposition 3.1. If it happens that $r_n(\nu) = 1$, then the $\mathbf{x}_k(= \mathbf{x}_\nu)$ in Line 7 is an exact solution to system (1.1) because its residual $\mathbf{u}_\nu$ is zero. So, the algorithm stops there. Otherwise, the $\mathbf{x}_k(= \mathbf{x}_\nu)$ in Line 16 will be exact with residual $\mathbf{r}_\nu = \mathbf{0}$ and where the algorithm stops.

Part (a) follows from the definition of $\mathbf{r}_k$ in (4.3) and Proposition 3.1(a).

Since $\widehat{\mathbf{r}}_k \neq \mathbf{0}$ for $1 \leq k \leq \nu - 1$ by Proposition 3.1(b) and $\phi_{g_n(k)+1}(\mathbf{A})$ is nonsingular due to $1/\omega \notin \sigma(\mathbf{A})$, we have $\mathbf{r}_k = \phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{r}}_k \neq \mathbf{0}$. Therefore, Part (b) holds.

For Part (c), write $k = jn + n$ with $0 \leq j$. By (4.3), (4.1) and Corollary 3.1, we have

$$\begin{aligned}\mathbf{q}_1^H \mathbf{r}_k &= \mathbf{q}_1^H \phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{r}}_k = \mathbf{q}_{r_n((j+1)n+1)}^H \phi_{g_n((j+1)n+1)}(\mathbf{A})\widehat{\mathbf{r}}_k \\ &= \mathbf{q}_{r_n(k+1)}^H \phi_{g_n(k+1)}(\mathbf{A})\widehat{\mathbf{r}}_k = c_{g_n(k+1)}^{(g_n(k+1))} \mathbf{p}_{k+1}^H \widehat{\mathbf{r}}_k = c_{g_n(k)+1}^{(g_n(k)+1)} \mathbf{p}_{k+1}^H \widehat{\mathbf{r}}_k.\end{aligned}$$

Now Part (c) follows from Proposition 3.1(d) and $c_{g_n(k)+1}^{(g_n(k)+1)} \neq 0$.

For the proof of Part (d), we first note that Algorithm 4.1 does not compute $\mathbf{u}_k$ when $r_n(k) = n$ (see Lines 13 - 15). Write $k = jn + i$ as in (2.1) and let $1 \leq t \leq i < n$. Then $r_n(k) = i$, $g_n(k) = j = g_n(jn + t)$ and $r_n(jn + t) = t$. Now, by (4.3) and Corollary 3.1, we have

$$\mathbf{q}_t^H \mathbf{u}_k = \mathbf{q}_t^H \phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{r}}_k = \mathbf{q}_{r_n(jn+t)}^H \phi_{g_n(jn+t)}(\mathbf{A})\widehat{\mathbf{r}}_k = c_{g_n(jn+t)}^{(g_n(jn+t))} \mathbf{p}_{jn+t}^H \widehat{\mathbf{r}}_k.$$

Since $\mathbf{p}_{jn+t}^H \widehat{\mathbf{r}}_k = 0$ by Proposition 3.1(d), $\mathbf{q}_t^H \mathbf{u}_k = 0$ for $1 \leq t \leq i$. Similarly,

$$\mathbf{q}_{i+1}^H \mathbf{u}_k = c_{g_n(k)}^{(g_n(k))} \mathbf{p}_{jn+i+1}^H \widehat{\mathbf{r}}_k = c_{g_n(k)}^{(g_n(k))} \mathbf{p}_{k+1}^H \widehat{\mathbf{r}}_k$$

(the validity of the first equation requires $i < n$). Because of Proposition 3.1(d) and $c_{g_n(k)}^{(g_n(k))} \neq 0$, $\mathbf{q}_{i+1}^H \mathbf{u}_k \neq 0$.

Similar to the quantity $\mathbf{u}_k$, Algorithm 4.1 does not compute $\mathbf{d}_k$ when $r_n(k) = n$ (see Lines 39 - 47). By (4.3), $\mathbf{d}_k = -\omega_{g_n(k)+1}\mathbf{A}\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{g}}_k$ and the proof of Part (e) is parallel to that of Part (d). □

The conditions of $\omega_{g_n(k)+1} \neq 0$ and $1/\omega_{g_n(k)+1} \notin \sigma(\mathbf{A})$ can be easily made satisfied. For example, one can add some small random noise to $\omega_{g_n(k)+1}$ after it is computed. The following theorem then holds from Proposition 4.1, Corollary 3.2 and Remark 3.2(ii).

Theorem 4.1. *If $\mathbf{q}_1, \cdots, \mathbf{q}_n$ are vectors with independent and continuous random elements and if some small and continuous random number is added to $\omega_{g_n(k)+1}$ after it is computed, then Algorithm* 4.1 *will work almost surely without breakdown by zero division to find a solution of* (1.1) *from the affine subspace $\mathbf{x}_0 + \mathbb{K}(\mathbf{A}, \mathbf{r}_0)$ provided that $\mathbf{x}_0$ is chosen such that the affine subspace contains a solution to* (1.1).

Proposition 4.1 indicates that exact solution can only be found at iteration $k = \nu$. It is possible, however, that $\|\mathbf{r}_k\|_2$ can become very small for some $k < \nu$. In practice, we terminate the algorithm when $\|\mathbf{r}_k\|_2$ falls within a given tolerance.

Theorem 4.1 guarantees that an exact breakdown is almost impossible. However, ML(n)BiCGStab can encounter a near breakdown in its implementation. Besides the two types of breakdown of ML(n)BiCG, ML(n)BiCGStab has one more type of breakdown caused by $\omega_{g_n(k)+1}$. In more details, the divisors in Algorithm 4.1 are $c_k, \|\mathbf{Au}_k\|_2^2$ and $\omega_{g_n(k)+1}$. If $\|\mathbf{Au}_k\|_2 \approx 0$, then $\omega_{g_n(k)+1} \approx \infty$ and a breakdown due to the overflow of $\omega_{g_n(k)+1}$ occurs. It can be shown (see the proof of Proposition 4.1) that

$$c_k = c_{g_n(k)+1}^{(g_n(k)+1)} \mathbf{p}_{k+1}^H \mathbf{A}\widehat{\mathbf{g}}_k,$$

where $c_{g_n(k)+1}^{(g_n(k)+1)}$ is the leading coefficient of $\phi_{g_n(k)+1}(\lambda)$ (see (4.1)). So, c_k is a quantity that relates to $\omega_{g_n(k)+1}$ and the ML(n)BiCG divisor $\mathbf{p}_{k+1}^H \mathbf{A}\widehat{\mathbf{g}}_k$. Thus, either $\omega_{g_n(k)+1} \approx 0$ or $\mathbf{p}_{k+1}^H \mathbf{A}\widehat{\mathbf{g}}_k \approx 0$ can cause $c_k \approx 0$.

A breakdown-free ML(n)BiCGStab algorithm was derived in [17].

5. A Second ML(n)BiCGStab Algorithm

If we write $k = jn + i$ as in (2.1), then the $\mathbf{r}_k$ defined by (4.3) become

$$\mathbf{r}_{jn+i} = \phi_{j+1}(\mathbf{A})\widehat{\mathbf{r}}_{jn+i} \tag{5.1}$$

where $i = 1, 2, \cdots, n$ and $j = 0, 1, 2, \cdots$. (5.1) increases the degree of the polynomial ϕ by 1 at the beginning of every cycle (see §1 for the definition of a cycle). For example, consider $n = 3$. Then (5.1) implies that

$$\begin{array}{lll} \mathbf{r}_1 = \phi_1(\mathbf{A})\widehat{\mathbf{r}}_1, & \mathbf{r}_4 = \phi_2(\mathbf{A})\widehat{\mathbf{r}}_4, & \mathbf{r}_7 = \phi_3(\mathbf{A})\widehat{\mathbf{r}}_7, \\ \mathbf{r}_2 = \phi_1(\mathbf{A})\widehat{\mathbf{r}}_2, & \mathbf{r}_5 = \phi_2(\mathbf{A})\widehat{\mathbf{r}}_5, & \mathbf{r}_8 = \phi_3(\mathbf{A})\widehat{\mathbf{r}}_8, \\ \mathbf{r}_3 = \phi_1(\mathbf{A})\widehat{\mathbf{r}}_3, & \mathbf{r}_6 = \phi_2(\mathbf{A})\widehat{\mathbf{r}}_6, & \mathbf{r}_9 = \phi_3(\mathbf{A})\widehat{\mathbf{r}}_9. \end{array}$$

Iteration $k = 4$ is the first iteration of the second cycle and the degree of ϕ is increased from 1 to 2 there.

One can define $\mathbf{r}_k$ by increasing the degree of ϕ by one anywhere within a cycle. Increasing the degree of ϕ at a different moment in a cycle gives a different value for ω and hence a different polynomial ϕ. So the resulting algorithms are different. As an illustration, let us increase the degree at the end of a cycle and derive the algorithm associated with it.

5.1. Notation and Definitions

Let $\phi_k(\lambda)$ be defined as in (1.2). For $k \in \mathbb{N}$, define

$$\begin{array}{ll} \mathbf{r}_k = \phi_{g_n(k+1)}(\mathbf{A})\widehat{\mathbf{r}}_k, & \mathbf{g}_k = \phi_{g_n(k+1)}(\mathbf{A})\widehat{\mathbf{g}}_k, \\ \mathbf{u}_k = \phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{r}}_k, & \mathbf{w}_k = \mathbf{A}\mathbf{g}_k \end{array} \tag{5.2}$$

and set

$$\mathbf{r}_0 = \widehat{\mathbf{r}}_0 \quad \text{and} \quad \mathbf{g}_0 = \widehat{\mathbf{g}}_0. \tag{5.3}$$

We remark that $\mathbf{r}_k = \mathbf{u}_k$ when $r_n(k) < n$ since $g_n(k+1) = g_n(k)$ in this case.

Definition (5.2) increases the degree of ϕ at the end of a cycle. To see this, let $n = 3$. Then (5.2) yields

$$\begin{array}{lll} \mathbf{r}_1 = \phi_0(\mathbf{A})\widehat{\mathbf{r}}_1, & \mathbf{r}_4 = \phi_1(\mathbf{A})\widehat{\mathbf{r}}_4, & \mathbf{r}_7 = \phi_2(\mathbf{A})\widehat{\mathbf{r}}_7, \\ \mathbf{r}_2 = \phi_0(\mathbf{A})\widehat{\mathbf{r}}_2, & \mathbf{r}_5 = \phi_1(\mathbf{A})\widehat{\mathbf{r}}_5, & \mathbf{r}_8 = \phi_2(\mathbf{A})\widehat{\mathbf{r}}_8, \\ \mathbf{r}_3 = \phi_1(\mathbf{A})\widehat{\mathbf{r}}_3, & \mathbf{r}_6 = \phi_2(\mathbf{A})\widehat{\mathbf{r}}_6, & \mathbf{r}_9 = \phi_3(\mathbf{A})\widehat{\mathbf{r}}_9. \end{array}$$

5.2. Algorithm Derivation

To derive the algorithm associated with (5.2), we first transform Algorithm 3.1 (forgetting Lines 1, 2, 5 and 11) into the following version which is computationally equivalent to Algorithm 3.1, but is more convenient for us to apply Proposition 2.1.

Derivation Stage #5.

1. For $k = 1, 2, \cdots$, until convergence:
2. $\quad \alpha_k = \mathbf{p}_k^H \widehat{\mathbf{r}}_{k-1} / \mathbf{p}_k^H \mathbf{A}\widehat{\mathbf{g}}_{k-1}$;
3. $\quad$ If $r_n(k) < n$
4. $\quad\quad \widehat{\mathbf{r}}_k = \widehat{\mathbf{r}}_{k-1} - \alpha_k \mathbf{A}\widehat{\mathbf{g}}_{k-1}$;
5. $\quad\quad$ For $s = \max(k-n, 0), \cdots, g_n(k)n - 1$
6. $\quad\quad\quad \beta_s^{(k)} = -\mathbf{p}_{s+1}^H \mathbf{A}\left(\widehat{\mathbf{r}}_k + \sum_{t=\max(k-n,0)}^{s-1} \beta_t^{(k)} \widehat{\mathbf{g}}_t\right) / \mathbf{p}_{s+1}^H \mathbf{A}\widehat{\mathbf{g}}_s$;
7. $\quad\quad$ End
8. $\quad\quad$ For $s = g_n(k)n, \cdots, k-1$
9. $\quad\quad\quad \beta_s^{(k)} = -\mathbf{p}_{s+1}^H \mathbf{A}\left(\widehat{\mathbf{r}}_k + \sum_{t=\max(k-n,0)}^{g_n(k)n-1} \beta_t^{(k)} \widehat{\mathbf{g}}_t + \sum_{t=g_n(k)n}^{s-1} \beta_t^{(k)} \widehat{\mathbf{g}}_t\right) / \mathbf{p}_{s+1}^H \mathbf{A}\widehat{\mathbf{g}}_s$;
10. $\quad\quad$ End
11. $\quad\quad \widehat{\mathbf{g}}_k = \widehat{\mathbf{r}}_k + \sum_{s=\max(k-n,0)}^{g_n(k)n-1} \beta_s^{(k)} \widehat{\mathbf{g}}_s + \sum_{s=g_n(k)n}^{k-1} \beta_s^{(k)} \widehat{\mathbf{g}}_s$;
12. $\quad$ Else
13. $\quad\quad \widehat{\mathbf{r}}_k = \widehat{\mathbf{r}}_{k-1} - \alpha_k \mathbf{A}\widehat{\mathbf{g}}_{k-1}$;
14. $\quad\quad$ For $s = g_n(k)n, \cdots, k-1$
15. $\quad\quad\quad \beta_s^{(k)} = -\mathbf{p}_{s+1}^H \mathbf{A}\left(\widehat{\mathbf{r}}_k + \sum_{t=g_n(k)n}^{s-1} \beta_t^{(k)} \widehat{\mathbf{g}}_t\right) / \mathbf{p}_{s+1}^H \mathbf{A}\widehat{\mathbf{g}}_s$;
16. $\quad\quad$ End
17. $\quad\quad \widehat{\mathbf{g}}_k = \widehat{\mathbf{r}}_k + \sum_{s=g_n(k)n}^{k-1} \beta_s^{(k)} \widehat{\mathbf{g}}_s$;
18. $\quad$ End
19. End

Then we transform DS#5 as follows by Corollary 3.1.

Derivation Stage #6.

1. For $k = 1, 2, \cdots$, until convergence:
2. $\quad \alpha_k = \mathbf{q}_{r_n(k)}^H \phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{r}}_{k-1} / \mathbf{q}_{r_n(k)}^H \mathbf{A}\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{g}}_{k-1}$;

3. If $r_n(k) < n$
4. $\quad \phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{r}}_k = \phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{r}}_{k-1} - \alpha_k \mathbf{A}\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{g}}_{k-1}$;
5. $\quad$ For $s = \max(k-n, 0), \cdots, g_n(k)n - 1$
6. $\quad\quad \beta_s^{(k)} = \mathbf{q}_{r_n(s+1)}^H \left(\phi_{g_n(s+1)+1}(\mathbf{A})\widehat{\mathbf{r}}_k - \omega_{g_n(s+1)+1} \sum_{t=\max(k-n,0)}^{s-1} \beta_t^{(k)} \right.$
$\quad\quad\quad \left. \mathbf{A}\phi_{g_n(s+1)}(\mathbf{A})\widehat{\mathbf{g}}_t \right) / \omega_{g_n(s+1)+1} \mathbf{q}_{r_n(s+1)}^H \mathbf{A}\phi_{g_n(s+1)}(\mathbf{A})\widehat{\mathbf{g}}_s$;
7. $\quad$ End
8. $\quad$ For $s = g_n(k)n, \cdots, k-1$
9. $\quad\quad \beta_s^{(k)} = -\mathbf{q}_{r_n(s+1)}^H \mathbf{A} \left(\phi_{g_n(s+1)}(\mathbf{A})\widehat{\mathbf{r}}_k + \sum_{t=\max(k-n,0)}^{g_n(k)n-1} \beta_t^{(k)} \phi_{g_n(s+1)}(\mathbf{A})\widehat{\mathbf{g}}_t \right.$
$\quad\quad\quad \left. + \sum_{t=g_n(k)n}^{s-1} \beta_t^{(k)} \phi_{g_n(s+1)}(\mathbf{A})\widehat{\mathbf{g}}_t \right) / \mathbf{q}_{r_n(s+1)}^H \mathbf{A}\phi_{g_n(s+1)}(\mathbf{A})\widehat{\mathbf{g}}_s$;
10. $\quad$ End
11. $\quad \phi_{g_n(k+1)}(\mathbf{A})\widehat{\mathbf{g}}_k = \phi_{g_n(k+1)}(\mathbf{A})\widehat{\mathbf{r}}_k + (\mathbf{I} - \omega_{g_n(k+1)}\mathbf{A}) \sum_{s=\max(k-n,0)}^{g_n(k)n-1} \beta_s^{(k)} \phi_{g_n(k+1)-1}(\mathbf{A})\widehat{\mathbf{g}}_s$
$\quad\quad\quad + \sum_{s=g_n(k)n}^{k-1} \beta_s^{(k)} \phi_{g_n(k+1)}(\mathbf{A})\widehat{\mathbf{g}}_s$;
12. Else
13. $\quad \phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{r}}_k = \phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{r}}_{k-1} - \alpha_k \mathbf{A}\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{g}}_{k-1}$;
14. $\quad \phi_{g_n(k+1)}(\mathbf{A})\widehat{\mathbf{r}}_k = (\mathbf{I} - \omega_{g_n(k+1)}\mathbf{A})\phi_{g_n(k+1)-1}(\mathbf{A})\widehat{\mathbf{r}}_k$;
15. $\quad$ For $s = g_n(k)n, \cdots, k-1$
16. $\quad\quad \beta_s^{(k)} = \mathbf{q}_{r_n(s+1)}^H \left(\phi_{g_n(s+1)+1}(\mathbf{A})\widehat{\mathbf{r}}_k - \omega_{g_n(s+1)+1} \sum_{t=g_n(k)n}^{s-1} \beta_t^{(k)} \right.$
$\quad\quad\quad \left. \mathbf{A}\phi_{g_n(s+1)}(\mathbf{A})\widehat{\mathbf{g}}_t \right) / \omega_{g_n(s+1)+1} \mathbf{q}_{r_n(s+1)}^H \mathbf{A}\phi_{g_n(s+1)}(\mathbf{A})\widehat{\mathbf{g}}_s$;
17. $\quad$ End
18. $\quad \phi_{g_n(k+1)}(\mathbf{A})\widehat{\mathbf{g}}_k = \phi_{g_n(k+1)}(\mathbf{A})\widehat{\mathbf{r}}_k + (\mathbf{I} - \omega_{g_n(k+1)}\mathbf{A}) \sum_{s=g_n(k)n}^{k-1} \beta_s^{(k)} \phi_{g_n(k+1)-1}(\mathbf{A})\widehat{\mathbf{g}}_s$;
19. End
20. End

Lines 4, 11, 13 and 18, DS#6, were obtained from Lines 4, 11, 13 and 17, DS#5, by multiplying them with $\phi_{g_n(k)}(\mathbf{A})$ and $\phi_{g_n(k+1)}(\mathbf{A})$ respectively. Line 14, DS#6, is a direct result of the definition (1.2) of ϕ.

Now we use Proposition 2.1 to write DS#6 as

Derivation Stage #7.

1. For $k = 1, 2, \cdots$, until convergence:
2. $\quad \alpha_k = \mathbf{q}_{r_n(k)}^H \phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{r}}_{k-1} / \mathbf{q}_{r_n(k)}^H \mathbf{A}\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{g}}_{k-1}$;
3. $\quad$ If $r_n(k) < n$
4. $\quad\quad \phi_{g_n(k+1)}(\mathbf{A})\widehat{\mathbf{r}}_k = \phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{r}}_{k-1} - \alpha_k \mathbf{A}\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{g}}_{k-1}$;
5. $\quad\quad$ For $s = \max(k-n, 0), \cdots, g_n(k)n - 1$
6. $\quad\quad\quad \beta_s^{(k)} = \mathbf{q}_{r_n(s+1)}^H \left(\phi_{g_n(k+1)}(\mathbf{A})\widehat{\mathbf{r}}_k - \omega_{g_n(k+1)} \sum_{t=\max(k-n,0)}^{s-1} \right.$
$\quad\quad\quad\quad \left. \beta_t^{(k)} \mathbf{A}\phi_{g_n(t+1)}(\mathbf{A})\widehat{\mathbf{g}}_t \right) / \omega_{g_n(k+1)} \mathbf{q}_{r_n(s+1)}^H \mathbf{A}\phi_{g_n(s+1)}(\mathbf{A})\widehat{\mathbf{g}}_s$;
7. $\quad\quad$ End

8. For $s = g_n(k)n, \cdots, k-1$
9. $\beta_s^{(k)} = -\mathbf{q}_{r_n(s+1)}^H \mathbf{A}\Big(\phi_{g_n(k+1)}(\mathbf{A})\widehat{\mathbf{r}}_k + \sum_{t=\max(k-n,0)}^{g_n(k)n-1} \beta_t^{(k)} \phi_{g_n(t+1)+1}(\mathbf{A})\widehat{\mathbf{g}}_t + \sum_{t=g_n(k)n}^{s-1} \beta_t^{(k)} \phi_{g_n(t+1)}(\mathbf{A})\widehat{\mathbf{g}}_t\Big) / \mathbf{q}_{r_n(s+1)}^H \mathbf{A}\phi_{g_n(s+1)}(\mathbf{A})\widehat{\mathbf{g}}_s;$
10. End
11. $\phi_{g_n(k+1)}(\mathbf{A})\widehat{\mathbf{g}}_k = \phi_{g_n(k+1)}(\mathbf{A})\widehat{\mathbf{r}}_k + (\mathbf{I} - \omega_{g_n(k+1)}\mathbf{A}) \sum_{s=\max(k-n,0)}^{g_n(k)n-1} \beta_s^{(k)} \phi_{g_n(s+1)}(\mathbf{A})\widehat{\mathbf{g}}_s + \sum_{s=g_n(k)n}^{k-1} \beta_s^{(k)} \phi_{g_n(s+1)}(\mathbf{A})\widehat{\mathbf{g}}_s;$
12. Else
13. $\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{r}}_k = \phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{r}}_{k-1} - \alpha_k \mathbf{A}\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{g}}_{k-1};$
14. $\phi_{g_n(k+1)}(\mathbf{A})\widehat{\mathbf{r}}_k = (\mathbf{I} - \omega_{g_n(k+1)}\mathbf{A})\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{r}}_k;$
15. For $s = g_n(k)n, \cdots, k-1$
16. $\beta_s^{(k)} = \mathbf{q}_{r_n(s+1)}^H \Big(\phi_{g_n(k+1)}(\mathbf{A})\widehat{\mathbf{r}}_k - \omega_{g_n(k+1)} \sum_{t=g_n(k)n}^{s-1} \beta_t^{(k)} \mathbf{A}\phi_{g_n(t+1)}(\mathbf{A})\widehat{\mathbf{g}}_t\Big) / \omega_{g_n(k+1)} \mathbf{q}_{r_n(s+1)}^H \mathbf{A}\phi_{g_n(s+1)}(\mathbf{A})\widehat{\mathbf{g}}_s;$
17. End
18. $\phi_{g_n(k+1)}(\mathbf{A})\widehat{\mathbf{g}}_k = \phi_{g_n(k+1)}(\mathbf{A})\widehat{\mathbf{r}}_k + (\mathbf{I} - \omega_{g_n(k+1)}\mathbf{A}) \sum_{s=g_n(k)n}^{k-1} \beta_s^{(k)} \phi_{g_n(s+1)}(\mathbf{A})\widehat{\mathbf{g}}_s;$
19. End
20. End

We remark that the term $\phi_{g_n(t+1)+1}(\mathbf{A})\widehat{\mathbf{g}}_t$ in the first sum in Line 9 can be further written as

$$\begin{aligned} \phi_{g_n(t+1)+1}(\mathbf{A})\widehat{\mathbf{g}}_t &= (\mathbf{I} - \omega_{g_n(t+1)+1}\mathbf{A})\phi_{g_n(t+1)}(\mathbf{A})\widehat{\mathbf{g}}_t \\ &= (\mathbf{I} - \omega_{g_n(k+1)}\mathbf{A})\phi_{g_n(t+1)}(\mathbf{A})\widehat{\mathbf{g}}_t. \end{aligned} \tag{5.4}$$

Substituting (5.4) and (5.2) into DS#7 then yields a set of updating relations of the vectors defined by (5.2).

Derivation Stage #8.

1. For $k = 1, 2, \cdots,$ until convergence:
2. $\alpha_k = \mathbf{q}_{r_n(k)}^H \mathbf{r}_{k-1} / \mathbf{q}_{r_n(k)}^H \mathbf{w}_{k-1};$
3. If $r_n(k) < n$
4. $\mathbf{r}_k = \mathbf{r}_{k-1} - \alpha_k \mathbf{w}_{k-1};$
5. For $s = \max(k-n, 0), \cdots, g_n(k)n - 1$
6. $\beta_s^{(k)} = \mathbf{q}_{r_n(s+1)}^H \Big(\mathbf{r}_k - \omega_{g_n(k+1)} \sum_{t=\max(k-n,0)}^{s-1} \beta_t^{(k)} \mathbf{w}_t\Big) / \omega_{g_n(k+1)} \mathbf{q}_{r_n(s+1)}^H \mathbf{w}_s;$
7. End
8. For $s = g_n(k)n, \cdots, k-1$
9. $\beta_s^{(k)} = -\mathbf{q}_{r_n(s+1)}^H \Big(\mathbf{A}\mathbf{r}_k + \sum_{t=\max(k-n,0)}^{g_n(k)n-1} \beta_t^{(k)} (\mathbf{I} - \omega_{g_n(k+1)}\mathbf{A})\mathbf{w}_t + \sum_{t=g_n(k)n}^{s-1} \beta_t^{(k)} \mathbf{w}_t\Big) / \mathbf{q}_{r_n(s+1)}^H \mathbf{w}_s;$
10. End
11. $\mathbf{g}_k = \mathbf{r}_k - \omega_{g_n(k+1)} \sum_{s=\max(k-n,0)}^{g_n(k)n-1} \beta_s^{(k)} \mathbf{w}_s + \sum_{s=\max(k-n,0)}^{g_n(k)n-1} \beta_s^{(k)} \mathbf{g}_s + \sum_{s=g_n(k)n}^{k-1} \beta_s^{(k)} \mathbf{g}_s;$

12. Else
13. $\mathbf{u}_k = \mathbf{r}_{k-1} - \alpha_k \mathbf{w}_{k-1}$;
14. $\mathbf{r}_k = (\mathbf{I} - \omega_{g_n(k+1)}\mathbf{A})\mathbf{u}_k$;
15. For $s = g_n(k)n, \cdots, k-1$
16. $\beta_s^{(k)} = \mathbf{q}_{r_n(s+1)}^H \left(\mathbf{r}_k - \omega_{g_n(k+1)} \sum_{t=g_n(k)n}^{s-1} \beta_t^{(k)} \mathbf{w}_t\right) / \omega_{g_n(k+1)} \mathbf{q}_{r_n(s+1)}^H \mathbf{w}_s$;
17. End
18. $\mathbf{g}_k = \mathbf{r}_k - \omega_{g_n(k+1)} \sum_{s=g_n(k)n}^{k-1} \beta_s^{(k)} \mathbf{w}_s + \sum_{s=g_n(k)n}^{k-1} \beta_s^{(k)} \mathbf{g}_s$;
19. End
20. End

DS#8 does not contain any update about $\mathbf{w}_k$. For the updates, we multiply the equations in Lines 11 and 18 by $\mathbf{A}$ to get

$$\begin{aligned}\mathbf{w}_k =& \mathbf{A}(\mathbf{r}_k - \omega_{g_n(k+1)} \sum_{s=\max(k-n,0)}^{g_n(k)n-1} \beta_s^{(k)} \mathbf{w}_s) + \sum_{s=\max(k-n,0)}^{g_n(k)n-1} \beta_s^{(k)} \mathbf{w}_s \\ &+ \sum_{s=g_n(k)n}^{k-1} \beta_s^{(k)} \mathbf{w}_s \end{aligned} \tag{5.5}$$

if $r_n(k) < n$, and

$$\mathbf{w}_k = \mathbf{A}(\mathbf{r}_k - \omega_{g_n(k+1)} \sum_{s=g_n(k)n}^{k-1} \beta_s^{(k)} \mathbf{w}_s) + \sum_{s=g_n(k)n}^{k-1} \beta_s^{(k)} \mathbf{w}_s \tag{5.6}$$

if $r_n(k) = n$.

Again, we consider $\mathbf{r}_k$ to be a residual. To be consistent with Lines 4, 13 and 14, we update the approximate solution $\mathbf{x}_k$ as

$$\mathbf{x}_k = \begin{cases} \mathbf{x}_{k-1} + \alpha_k \mathbf{g}_{k-1}, & \text{if } r_n(k) < n, \\ \omega_{g_n(k+1)} \mathbf{u}_k + \mathbf{x}_{k-1} + \alpha_k \mathbf{g}_{k-1}, & \text{if } r_n(k) = n. \end{cases} \tag{5.7}$$

Now adding (5.5)-(5.7) to DS#8 and simplifying the operations appropriately, we then arrive at the following algorithm. The free parameter $\omega_{g_n(k+1)}$ is chosen to minimize the 2-norm of $\mathbf{r}_k$.

Algorithm 5.1. ML(n)BiCGStab without preconditioning associated with (5.2)

1. *Choose an initial guess* $\mathbf{x}_0$ *and n vectors* $\mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_n$.
2. *Compute* $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$ *and* $\mathbf{g}_0 = \mathbf{r}_0$, $\mathbf{w}_0 = \mathbf{A}\mathbf{g}_0$, $c_0 = \mathbf{q}_1^H \mathbf{w}_0$.
3. *For* $k = 1, 2, \cdots$, *until convergence:*
4. $\alpha_k = \mathbf{q}_{r_n(k)}^H \mathbf{r}_{k-1} / c_{k-1}$;
5. *If* $r_n(k) < n$
6. $\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k \mathbf{g}_{k-1}$;

7. $\mathbf{r}_k = \mathbf{r}_{k-1} - \alpha_k \mathbf{w}_{k-1}$;
8. $\mathbf{z}_w = \mathbf{r}_k,\ \mathbf{g}_k = \mathbf{0}$;
9. *For* $s = \max(k-n, 0), \cdots, g_n(k)n - 1$
10. $\tilde{\beta}_s^{(k)} = -\mathbf{q}_{r_n(s+1)}^H \mathbf{z}_w / c_s$; % $\tilde{\beta}_s^{(k)} = -\beta_s^{(k)} \omega_{g_n(k+1)}$
11. $\mathbf{z}_w = \mathbf{z}_w + \tilde{\beta}_s^{(k)} \mathbf{w}_s$;
12. $\mathbf{g}_k = \mathbf{g}_k + \tilde{\beta}_s^{(k)} \mathbf{g}_s$;
13. *End*
14. $\mathbf{g}_k = \mathbf{z}_w - \dfrac{1}{\omega_{g_n(k+1)}} \mathbf{g}_k;\ \mathbf{w}_k = \mathbf{A}\mathbf{g}_k$;
15. *For* $s = g_n(k)n, \cdots, k-1$
16. $\beta_s^{(k)} = -\mathbf{q}_{r_n(s+1)}^H \mathbf{w}_k / c_s$;
17. $\mathbf{w}_k = \mathbf{w}_k + \beta_s^{(k)} \mathbf{w}_s$;
18. $\mathbf{g}_k = \mathbf{g}_k + \beta_s^{(k)} \mathbf{g}_s$;
19. *End*
20. *Else*
21. $\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k \mathbf{g}_{k-1}$;
22. $\mathbf{u}_k = \mathbf{r}_{k-1} - \alpha_k \mathbf{w}_{k-1}$;
23. $\omega_{g_n(k+1)} = (\mathbf{A}\mathbf{u}_k)^H \mathbf{u}_k / \|\mathbf{A}\mathbf{u}_k\|_2^2$;
24. $\mathbf{x}_k = \mathbf{x}_k + \omega_{g_n(k+1)} \mathbf{u}_k$;
25. $\mathbf{r}_k = -\omega_{g_n(k+1)} \mathbf{A}\mathbf{u}_k + \mathbf{u}_k$;
26. $\mathbf{z}_w = \mathbf{r}_k,\ \mathbf{g}_k = \mathbf{0}$;
27. *For* $s = g_n(k)n, \cdots, k-1$
28. $\tilde{\beta}_s^{(k)} = -\mathbf{q}_{r_n(s+1)}^H \mathbf{z}_w / c_s$; % $\tilde{\beta}_s^{(k)} = -\beta_s^{(k)} \omega_{g_n(k+1)}$
29. $\mathbf{z}_w = \mathbf{z}_w + \tilde{\beta}_s^{(k)} \mathbf{w}_s$;
30. $\mathbf{g}_k = \mathbf{g}_k + \tilde{\beta}_s^{(k)} \mathbf{g}_s$;
31. *End*
32. $\mathbf{g}_k = \mathbf{z}_w - \dfrac{1}{\omega_{g_n(k+1)}} \mathbf{g}_k;\ \mathbf{w}_k = \mathbf{A}\mathbf{g}_k$;
33. *End*
34. $c_k = \mathbf{q}_{r_n(k+1)}^H \mathbf{w}_k$;
35. *End*

We remark that (i) the algorithm does not compute $\mathbf{u}_k$ when $r_n(k) < n$. In fact, $\mathbf{u}_k = \mathbf{r}_k$ when $r_n(k) < n$ (see the remark right after (5.3)); (ii) if the $\mathbf{u}_k$ in Line 22 happens to be zero, then the $\mathbf{x}_k$ in Line 21 will be the exact solution to system (1.1) and the algorithm stops there.

The cost and storage requirement, obtained from its preconditioned version, Algorithm 9.2 in §9, are listed in Table 3. Compared to Algorithm 4.1, Algorithm 5.1 saves about 20% in saxpy. Since only three sets of vectors $\{\mathbf{q}_1, \cdots, \mathbf{q}_n\}$, $\{\mathbf{g}_{k-n}, \cdots, \mathbf{g}_{k-1}\}$ and $\{\mathbf{w}_{k-n}, \cdots, \mathbf{w}_{k-1}\}$ are needed in iteration k, the storage is about $3nN$ besides storing $\mathbf{A}$ and $\mathbf{M}$.

Table 3: Average cost per k-iteration of Algorithm 9.2 and its storage requirement. This table does not count the costs in Lines 1-2 of the algorithm.

Preconditioning $\mathbf{M}^{-1}\mathbf{v}$	$1+1/n$	$\mathbf{u}\pm\mathbf{v}$, $\alpha\mathbf{v}$	1
Matvec $\mathbf{Av}$	$1+1/n$	Saxpy $\mathbf{u}+\alpha\mathbf{v}$	$2n+2+2/n$
dot product $\mathbf{u}^H\mathbf{v}$	$n+1+2/n$	Storage	$\mathbf{A}+\mathbf{M}+(3n+5)N+O(n)$

5.3. Properties

We summarize the properties about Algorithm 5.1 below. Their proofs are similar to those in Proposition 4.1. Since $\mathbf{r}_0=\widehat{\mathbf{r}}_0$ by (5.3), ν is also the degree of $p_{min}(\lambda;\mathbf{A},\mathbf{r}_0)$.

Proposition 5.1. *Under the assumptions of Proposition* 3.1*, if* $\omega_{g_n(k+1)}\neq 0$ *and* $1/\omega_{g_n(k+1)}\notin\sigma(\mathbf{A})$ *for* $1\le k\le\nu-1$*, then Algorithm* 5.1 *does not break down by zero division for* $k=1,2,\cdots,\nu$*, and the approximate solution* $\mathbf{x}_\nu$ *at step* $k=\nu$ *is exact to the system* (1.1)*. Moreover, the computed quantities satisfy*

(a) $\mathbf{x}_k\in\mathbf{x}_0+span\{\mathbf{r}_0,\mathbf{Ar}_0,\ldots,\mathbf{A}^{g_n(k+1)+k-1}\mathbf{r}_0\}$ *and* $\mathbf{r}_k=\mathbf{b}-\mathbf{Ax}_k\in\mathbf{r}_0+span\{\mathbf{Ar}_0,\mathbf{A}^2\mathbf{r}_0,\ldots,\mathbf{A}^{g_n(k+1)+k}\mathbf{r}_0\}$ *for* $1\le k\le\nu-1$.

(b) $\mathbf{r}_k\neq\mathbf{0}$ *for* $1\le k\le\nu-1$*;* $\mathbf{r}_\nu=\mathbf{0}$.

(c) $\mathbf{r}_k\perp span\{\mathbf{q}_1,\mathbf{q}_2,\cdots,\mathbf{q}_{r_n(k)}\}$ *and* $\mathbf{r}_k\not\perp\mathbf{q}_{r_n(k)+1}$ *for* $1\le k\le\nu-1$ *with* $r_n(k)<n$*;* $\mathbf{r}_k\not\perp\mathbf{q}_1$ *for* $1\le k\le\nu-1$ *with* $r_n(k)=n$.

(d) $\mathbf{u}_k\perp span\{\mathbf{q}_1,\mathbf{q}_2,\cdots,\mathbf{q}_n\}$ *for* $1\le k\le\nu$ *with* $r_n(k)=n$.

(e) $\mathbf{Ag}_k\perp span\{\mathbf{q}_1,\mathbf{q}_2,\cdots,\mathbf{q}_{r_n(k)}\}$ *and* $\mathbf{Ag}_k\not\perp\mathbf{q}_{r_n(k)+1}$ *for* $1\le k\le\nu-1$ *with* $r_n(k)<n$*;* $\mathbf{Ag}_k\not\perp\mathbf{q}_1$ *for* $1\le k\le\nu-1$ *with* $r_n(k)=n$.

6. Relations to Some Existing Methods

In this section, we discuss the relations of ML(n)BiCGStab with the FOM, BiCGStab and IDR(s) methods under the assumptions of Proposition 3.1.

6.1. Algorithm 4.1

1. *Relation with FOM [21].* Consider the case where $n\ge\nu$. In this case, $g_n(k)=0$ and $r_n(k)=k$ for $k=1,2,\cdots,\nu$.[‖] Hence $\mathbf{p}_k=\mathbf{q}_k$ by (3.1). If we choose $\mathbf{q}_k=\widehat{\mathbf{r}}_{k-1}$ in Algorithm 3.1 (it is possible since $\widehat{\mathbf{r}}_{k-1}$ is computed before $\mathbf{q}_k$ is used), then the $\widehat{\mathbf{x}}_k$ and $\widehat{\mathbf{r}}_k$ computed by the algorithm satisfy

$$\begin{cases}\widehat{\mathbf{x}}_k\in\widehat{\mathbf{x}}_0+span\{\widehat{\mathbf{r}}_0,\mathbf{A}\widehat{\mathbf{r}}_0,\ldots,\mathbf{A}^{k-1}\widehat{\mathbf{r}}_0\},\\ \widehat{\mathbf{r}}_k\perp span\{\widehat{\mathbf{r}}_0,\widehat{\mathbf{r}}_1,\ldots,\widehat{\mathbf{r}}_{k-1}\}\end{cases}\tag{6.1}$$

[‖]We are only concerned about the first ν iterations since ML(n)BiCG and ML(n)BiCGStab converge at the ν-th iteration.

for $1 \le k \le \nu$ by Proposition 3.1(a), (d). (6.1) is what the FOM approximate solution $\mathbf{x}_k^{FOM}$ needs to satisfy. Therefore, when $n \ge \nu$ and with the choice $\mathbf{q}_k = \widehat{\mathbf{r}}_{k-1}$, Algorithm 3.1 is mathematically equivalent to FOM.

Now, from (4.3), the $\mathbf{r}_k$ computed by Algorithm 4.1 satisfies

$$\mathbf{r}_k = \phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{r}}_k = \phi_1(\mathbf{A})\widehat{\mathbf{r}}_k = (\mathbf{I} - \omega_1\mathbf{A})\widehat{\mathbf{r}}_k.$$

Note that $\mathbf{u}_k = \phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{r}}_k = \phi_0(\mathbf{A})\widehat{\mathbf{r}}_k = \widehat{\mathbf{r}}_k$. Thus, for $1 \le k \le \nu$, $\mathbf{r}_k$ is the factor $\mathbf{I} - \omega_1\mathbf{A}$ times the FOM residual $\mathbf{u}_k$ if we set $\mathbf{q}_1 = \mathbf{r}_0$ and $\mathbf{q}_{k+1} = \mathbf{u}_k$ in Algorithm 4.1.[**]

2. *Relation with BiCGStab* [31]. When $n = 1$, we have $g_n(k) = k - 1$ and $r_n(k) = 1$ for $k \in \mathbb{N}$. Hence $\mathbf{p}_k = \left(\mathbf{A}^H\right)^{k-1}\mathbf{q}_1$ by (3.1). By Proposition 3.1(a) and (d), the $\widehat{\mathbf{x}}_k$ and $\widehat{\mathbf{r}}_k$ computed by Algorithm 3.1 satisfy

$$\begin{cases} \widehat{\mathbf{x}}_k \in \widehat{\mathbf{x}}_0 + span\{\widehat{\mathbf{r}}_0, \mathbf{A}\widehat{\mathbf{r}}_0, \ldots, \mathbf{A}^{k-1}\widehat{\mathbf{r}}_0\} \\ \widehat{\mathbf{r}}_k \perp span\{\mathbf{q}_1, \mathbf{A}^H\mathbf{q}_1, \ldots, \left(\mathbf{A}^H\right)^{k-1}\mathbf{q}_1\} \end{cases} \tag{6.2}$$

for $1 \le k \le \nu$. (6.2) is what the BiCG approximate solution $\mathbf{x}_k^{BiCG}$ needs to satisfy. Therefore, when $n = 1$, Algorithm 3.1 is mathematically equivalent to BiCG.

Now, from (4.3), the $\mathbf{r}_k$ computed by Algorithm 4.1 satisfies

$$\mathbf{r}_k = \phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{r}}_k = \phi_k(\mathbf{A})\widehat{\mathbf{r}}_k$$

which is the definition of the BiCGStab residuals. Thus Algorithm 4.1 is mathematically equivalent to BiCGStab when $n = 1$.

3. *Relation with IDR(s)* [30]. Write $k = jn + i$ as in (2.1) with $1 \le i \le n, 0 \le j$. Let $\mathbb{G}_0 = \mathbb{K}(\mathbf{A}, \mathbf{r}_0)$ be the complete Krylov subspace and let $\mathbb{S} = span\{\mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_n\}^\perp$. Define the Sonneveld spaces

$$\mathbb{G}_{j+1} = (\mathbf{I} - \omega_{j+1}\mathbf{A})(\mathbb{G}_j \cap \mathbb{S}) = (\mathbf{I} - \omega_{g_n(k)+1}\mathbf{A})(\mathbb{G}_j \cap \mathbb{S})$$

for $j = 0, 1, 2, \cdots$. By (4.3), we have

$$\mathbf{r}_{jn+i} = \phi_{j+1}(\mathbf{A})\widehat{\mathbf{r}}_{jn+i} = (\mathbf{I} - \omega_{j+1}\mathbf{A})\phi_j(\mathbf{A})\widehat{\mathbf{r}}_{jn+i} = (\mathbf{I} - \omega_{j+1}\mathbf{A})\mathbf{u}_{jn+i}.$$

From Proposition 4.1(d), $\mathbf{u}_{jn+i} \not\perp \mathbf{q}_{i+1}$ if $i < n$. Hence $\mathbf{u}_{jn+i} \notin \mathbb{G}_j \cap \mathbb{S}$ and therefore $\mathbf{r}_{jn+i} \notin \mathbb{G}_{j+1}$ when $i < n$. From this point of view, Algorithm 4.1 is not a IDR(n) algorithm.

[**]In [39], a remark immediately following Theorem 4.1 states that, when $n \ge \nu$ and with the choice that $\mathbf{q}_1 = \phi_1(\mathbf{A}^H)\phi_1(\mathbf{A})\mathbf{r}_0$ and $\mathbf{q}_k = \phi_1(\mathbf{A}^H)\mathbf{r}_{k-1}$ for $k \ge 2$, the $\mathbf{x}_k$ and $\mathbf{r}_k$ computed by Algorithm 2 (which is mathematically equivalent to Algorithm 4.1 of this paper) will satisfies (6.1) and therefore Algorithm 2 is a FOM. The argument there about the remark is not correct. The author remembers that the referees of [39] were skeptical about that argument.

However, if we regard $\mathbf{r}_{jn+i}$ with $1 \le i < n$ as auxiliary vectors and instead, consider the followings as residuals

$$\mathbf{u}_{jn+1}, \quad \cdots, \quad \mathbf{u}_{jn+n-1}, \quad \mathbf{r}_{jn+n} \tag{6.3}$$

where $j = 0, 1, 2, \cdots$. Then Algorithm 4.1 is a IDR(n) algorithm — the parameter ω, however, is different from the ω of the IDR algorithm in [30] since they are selected to minimize the norm of different residuals. In fact, by (3.1) and Proposition 3.1(d), we have

$$\begin{cases} \phi_t(\mathbf{A})\widehat{\mathbf{r}}_{jn+i} \in \mathbb{S} & \text{if } 1 \le i < n \text{ and } 0 \le t < j, \\ \phi_t(\mathbf{A})\widehat{\mathbf{r}}_{jn+n} \in \mathbb{S} & \text{if } 0 \le t \le j. \end{cases}$$

Thus, by induction on t,

$$\begin{cases} \phi_t(\mathbf{A})\widehat{\mathbf{r}}_{jn+i} \in \mathbb{G}_t \cap \mathbb{S} & \text{if } 1 \le i < n \text{ and } 0 \le t < j, \\ \phi_t(\mathbf{A})\widehat{\mathbf{r}}_{jn+n} \in \mathbb{G}_t \cap \mathbb{S} & \text{if } 0 \le t \le j. \end{cases}$$

Therefore, by (4.3),

$$\begin{cases} \mathbf{u}_{jn+i} = \widehat{\mathbf{r}}_i \in \mathbb{G}_0 & \text{if } 1 \le i < n, j = 0, \\ \mathbf{u}_{jn+i} = (\mathbf{I} - \omega_j \mathbf{A})\phi_{j-1}(\mathbf{A})\widehat{\mathbf{r}}_{jn+i} \in \mathbb{G}_j & \text{if } 1 \le i < n, 1 \le j, \\ \mathbf{r}_{jn+n} = (\mathbf{I} - \omega_{j+1}\mathbf{A})\phi_j(\mathbf{A})\widehat{\mathbf{r}}_{jn+n} \in \mathbb{G}_{j+1}. \end{cases} \tag{6.4}$$

So, the residuals in (6.3) lie in the Sonneveld spaces $\mathbb{G}_j$.

That (6.4) holds becomes obvious if one applies the following result from [24] and Proposition 3.1(d),

$$\mathbb{G}_j = \{\phi_j(\mathbf{A})\mathbf{v} \,|\, \mathbf{v} \perp span\{\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_{jn}\}\}. \tag{6.5}$$

An early discussion of the relation between ML(n)BiCGStab and IDR(s) was made in [30].

6.2. Algorithm 5.1

1. *Relation with FOM*. When $n \ge \nu$, $g_n(k) = 0$ and $r_n(k) = k$ for $1 \le k \le \nu$ and Algorithm 3.1, with the choice $\mathbf{q}_k = \widehat{\mathbf{r}}_{k-1}$, is a FOM algorithm as seen in §6.1. Now, from (5.2), the $\mathbf{r}_k$ computed by Algorithm 5.1 satisfies

$$\mathbf{r}_k = \phi_{g_n(k+1)}(\mathbf{A})\widehat{\mathbf{r}}_k = \phi_0(\mathbf{A})\widehat{\mathbf{r}}_k = \widehat{\mathbf{r}}_k.$$

Thus Algorithm 5.1 is a FOM algorithm when we set $\mathbf{q}_k = \mathbf{r}_{k-1}$.

2. *Relation with BiCGStab*. When $n = 1$, we have $g_n(k) = k - 1$ and $r_n(k) = 1$ for $k \in \mathbb{N}$ and Algorithm 3.1 is a BiCG algorithm. From (5.2), the $\mathbf{r}_k$ computed by Algorithm 5.1 satisfies

$$\mathbf{r}_k = \phi_{g_n(k+1)}(\mathbf{A})\widehat{\mathbf{r}}_k = \phi_k(\mathbf{A})\widehat{\mathbf{r}}_k$$

which is the definition of the BiCGStab residuals. Thus Algorithm 5.1 is mathematically equivalent to BiCGStab.

3. *Relation with IDR(s)*. Write $k = jn + i$ as in (2.1) with $1 \le i \le n, 0 \le j$. By (5.2), we have

$$\mathbf{r}_{jn+i} = \phi_{g_n(jn+i+1)}(\mathbf{A})\widehat{\mathbf{r}}_{jn+i} = \begin{cases} \phi_j(\mathbf{A})\widehat{\mathbf{r}}_{jn+i} & \text{if } 1 \le i < n, \\ \phi_{j+1}(\mathbf{A})\widehat{\mathbf{r}}_{jn+i} & \text{if } i = n. \end{cases}$$

Thus, (6.5) and Proposition 3.1(d) yield

$$\mathbf{r}_{jn+i} \in \begin{cases} \mathbb{G}_j & \text{if } 1 \le i < n, \\ \mathbb{G}_{j+1} & \text{if } i = n. \end{cases}$$

So, the residuals computed by Algorithm 5.1 lie in the $\mathbb{G}$ spaces and therefore it is a IDR(n) algorithm.

7. Implementation Issues

A preconditioned ML(n)BiCGStab algorithm can be obtained by applying either Algorithm 4.1 or Algorithm 5.1 to the system

$$\mathbf{A}\mathbf{M}^{-1}\mathbf{y} = \mathbf{b}$$

where $\mathbf{M}$ is nonsingular, then recovering $\mathbf{x}$ through $\mathbf{x} = \mathbf{M}^{-1}\mathbf{y}$. The resulting algorithms, Algorithm 9.1 and Algorithm 9.2, together with their Matlab codes are presented in §9. To avoid calling the index functions $r_n(k)$ and $g_n(k)$ every k-iteration, we have split the k-loop into a i-loop and a j-loop where i, j, k are related by (2.1) with $1 \le i \le n, 0 \le j$. Moreover, we have optimized the operations as much as possible in the resulting preconditioned algorithms.

Since we have compared ML(n)BiCGStab with some existing methods in [39], we will only concentrate on the performance of ML(n)BiCGStab itself. The following test data were downloaded from Matrix Market at http://math.nist.gov/MatrixMarket/.

1. *e20r0100*, DRIVCAV Fluid Dynamics. *e20r0100* contains a 4241×4241 real unsymmetric matrix $\mathbf{A}$ with $131,556$ nonzero entries and a real right-hand side $\mathbf{b}$.

2. *qc2534*, H2PLUS Quantum Chemistry, NEP Collection. *qc2534* contains a 2534×2534 complex symmetric indefinite matrix with $463,360$ nonzero entries, but does not provide the right-hand side $\mathbf{b}$. Following [24], we set $\mathbf{b} = \mathbf{A1}$ with $\mathbf{1} = [1, , 1, \cdots, 1]^T$.

3. *utm5940*, TOKAMAK Nuclear Physics (Plasmas). *utm5940* contains a 5940×5940 real unsymmetric matrix $\mathbf{A}$ with $83,842$ nonzero entries and a real right-hand side $\mathbf{b}$.

All computing was done in Matlab Version 7.1 on a Windows XP machine with a Pentium 4 processor. $ILU(0)$ preconditioner (p.294, [20]) was used in all the experiments. For *e20r0100*, the U-factor of the $ILU(0)$ decomposition of $\mathbf{A}$ has some zeros along its

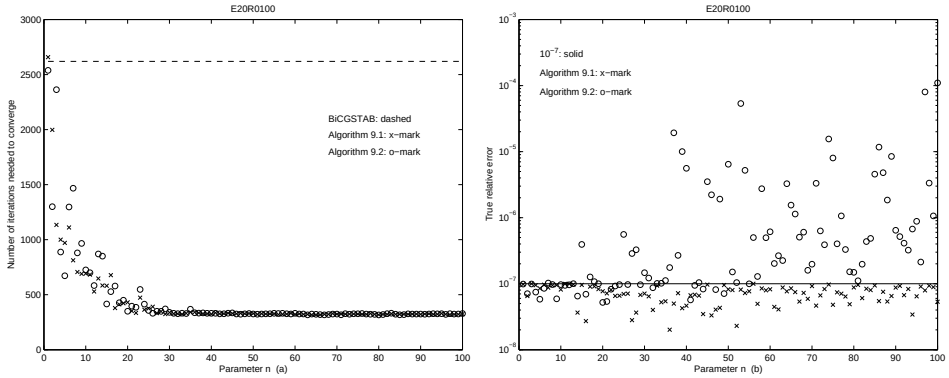

Figure 1: e20r0100: (a) Graphs of $I_{conv}(n)$ against n. BiCGStab took 2620 iterations/5240 MVs to converge. Full GMRES converged with 308 MVs. (b) Graphs of $E(n)$ against n.

main diagonal. In that experiment, we replaced those zeros by 1 so that the U-factor was invertible.

In all the experiments, initial guess was $\mathbf{x}_0 = \mathbf{0}$ with the stopping criterion $\|\mathbf{r}_k\|_2/\|\mathbf{b}\|_2 < 10^{-7}$ where $\mathbf{r}_k$ was the computed residual. Except where specified, shadow vectors $\mathbf{Q} = [\mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_n]$ were chosen to be $\mathbf{Q} = [\mathbf{r}_0, randn(N, n-1)]$ for *e20r0100* and *utm5940* and $\mathbf{Q} = [\mathbf{r}_0, randn(N, n-1) + sqrt(-1) * randn(N, n-1)]$ for *qc2534*.

Moreover, for the convenience of our presentation, we introduce the following functions:

(a) $T_{conv}(n)$ is the time that a ML(n)BiCGStab algorithm takes to converge.

(b) $I_{conv}(n)$ is the number of k-iterations (not iteration cycles) that a ML(n)BiCGStab algorithm takes to converge.

(c) $E(n) \equiv \|\mathbf{b}-\mathbf{A}\mathbf{x}\|_2/\|\mathbf{b}\|_2$ is the true relative error of $\mathbf{x}$ where $\mathbf{x}$ is the computed solution output by a ML(n)BiCGStab algorithm when it converges.

7.1. Stability

We plot the graphs of $I_{conv}(n)$ in Figures 1(a), 2(a) and 3(a). For *e20r0100* and *qc2534*, $I_{conv}(n)$ decreases as n increases. However, the $I_{conv}(n)$ for *utm5940* behaves very irregularly due to some of the ω's are too small.

The graphs of $E(n)$ are plotted in Figures 1(b), 2(b) and 3(b). It can be seen that the computed relative errors $\|\mathbf{r}_k\|_2/\|\mathbf{b}\|_2$ by Algorithm 9.2 can significantly diverge from its exact counterpart $\|\mathbf{b}-\mathbf{A}\mathbf{x}_k\|_2/\|\mathbf{b}\|_2$. By contrast, the computed $\|\mathbf{r}_k\|_2/\|\mathbf{b}\|_2$ by Algorithm 9.1 well approximate their corresponding true ones. Thus, from this point of view, we consider that Algorithm 9.1 is numerically more stable than Algorithm 9.2. One explanation about this difference in stability is the following. Recall that $\mathbf{w}_k = \mathbf{A}\mathbf{g}_k$ by definition (see (4.3) and (5.2)). In Algorithm 4.1, $\mathbf{w}_k$ is updated by $\mathbf{w}_k = \mathbf{A}\mathbf{g}_k$ (see Lines 37 and 47) for all k. In Algorithm 5.1, however, $\mathbf{w}_k$ is updated by $\mathbf{w}_k = \mathbf{A}\mathbf{g}_k$ only when $r_n(k) = n$ (see Line 32).

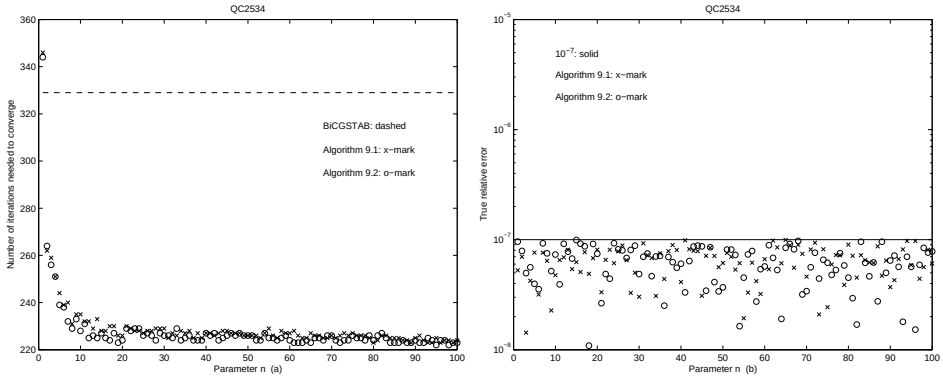

Figure 2: qc2534: (a) Graphs of $I_{conv}(n)$ against n. BiCGStab took 329 iterations/658 MVs to converge. Full GMRES converged with 439 MVs. (b) Graphs of $E(n)$ against n.

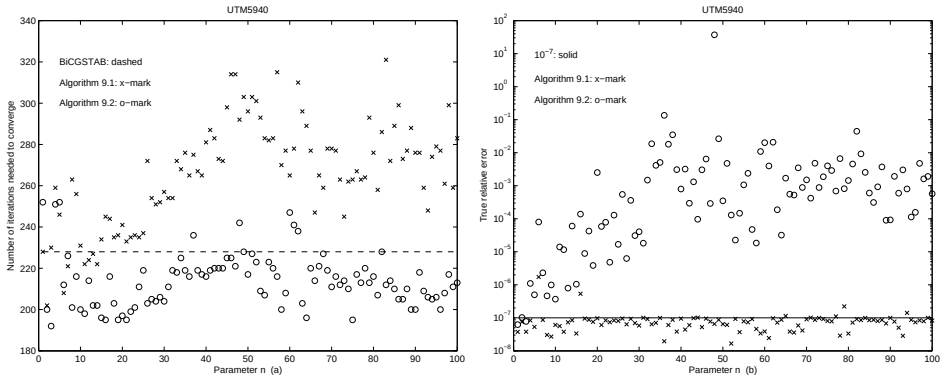

Figure 3: utm5940: (a) Graphs of $I_{conv}(n)$ against n. BiCGStab took 228 iterations/455 MVs to converge. Full GMRES converged with 176 MVs. (b) Graphs of $E(n)$ against n.

As a result, the $\mathbf{r}_k$'s computed by Algorithm 4.1 (see Lines 6, 10 and 17) are closer to the true residuals $\mathbf{b}-\mathbf{A}\mathbf{x}_k$ than those computed by Algorithm 5.1. This observation has led to a ML(n)BiCGStab algorithm which updates $\mathbf{w}_k$ by $\mathbf{w}_k = \mathbf{A}\mathbf{g}_k$ for all k, has less computational cost than Algorithm 5.1, but involves $\mathbf{A}^H$ in its implementation. See [37] for details.

We remark that the issues of divergence of computed residuals and corresponding remedy techniques were discussed in details in [26, 33].

7.2. Choice of n

In this and the following subsections, we will focus on Algorithm 9.1.

From the experiments in [39] and this paper, we have observed that ML(n)BiCGStab behaves more and more robust as n is increased. So, for an ill-conditioned system, we would tend to suggest a large n for ML(n)BiCGStab. On the other hand, ML(n)BiCGStab minimizes $\|\mathbf{r}_k\|_2$ once every n k-iterations. Notice that the convergence of a well-conditioned system is usually accelerated by the minimization steps. So, when a problem is well-conditioned, we would suggest a small n.

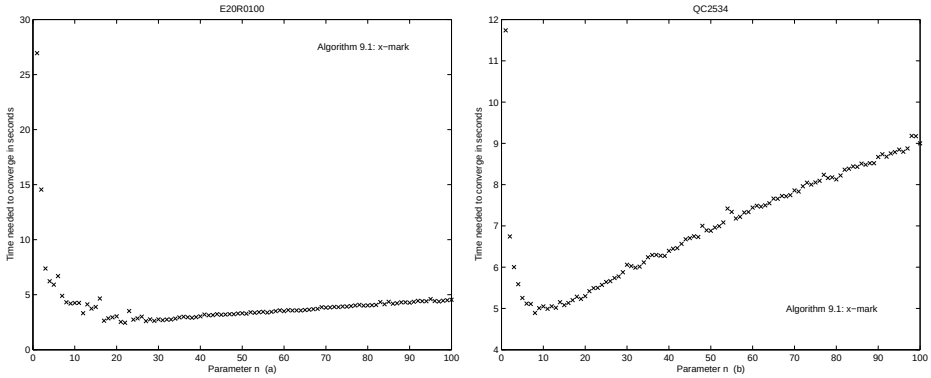

Figure 4: Graphs of $T_{conv}(n)$ of Algorithm 9.1 against n. (a) e20r0100: $T_{conv}(n)$ reaches its minimum at $n = 22$. (b) qc2534: $T_{conv}(n)$ reaches its minimum at $n = 8$.

In [30,34], it is suggested to fix s at 4 or 8 for the general use of IDR(s). This idea also applies to ML(n)BiCGStab.

In the case where a sequence of linear systems is solved, one can fix the parameter n for the overall solution process or alternatively one can choose n dynamically based on the information obtained from the solution of previous systems. We once tested the Matlab code of Algorithm 9.1 in §9.1, translated into Fortran and with $n = 9$, $\kappa = 0$ (see §7.3 for κ) and $ILU(0)$ preconditioners, on the standard oil reservoir simulation test data called SPE9 at University of Calgary. We found that ML(n)BiCGStab reduced the total computational time by over 70% when compared to BiCGStab. A later test on SPE9 with Code #4 in §9.3, at Jinan University, showed that a 60% reduction in time could be reached. Code #4 is a design of an automatic selection of n during the solution process of a sequence of linear systems. It tries to minimize the time per k-iteration of ML(n)BiCGStab.

We also plot the graphs of $T_{conv}(n)$ in Figures 4 and 5(a) to provide more information on how n affects the performance of ML(n)BiCGStab.

7.3. Choice of ω

The standard choice for ω_{j+1} in Algorithm 9.1 (see Line 8) is

$$\omega_{j+1} = (\mathbf{A}\widetilde{\mathbf{u}}_{jn+1})^H \mathbf{u}_{jn+1} / \|\mathbf{A}\widetilde{\mathbf{u}}_{jn+1}\|_2^2. \tag{7.1}$$

This choice of ω_{j+1} minimizes the 2-norm of $\mathbf{r}_{jn+1} = -\omega_{j+1}\mathbf{A}\widetilde{\mathbf{u}}_{jn+1} + \mathbf{u}_{jn+1}$ (see Line 10), but sometimes can cause instability due to that it can be very small during an execution. The following remedy to guard ω_{j+1} away from zero has been proposed in [25]:

$$\begin{array}{l} \omega_{j+1} = (\mathbf{A}\widetilde{\mathbf{u}}_{jn+1})^H \mathbf{u}_{jn+1} / \|\mathbf{A}\widetilde{\mathbf{u}}_{jn+1}\|_2^2; \\ \rho = (\mathbf{A}\widetilde{\mathbf{u}}_{jn+1})^H \mathbf{u}_{jn+1} / (\|\mathbf{A}\widetilde{\mathbf{u}}_{jn+1}\|_2 \, \|\mathbf{u}_{jn+1}\|_2); \\ \text{if } |\rho| < \kappa, \ \omega_{j+1} = \kappa\omega_{j+1}/|\rho|; \ \text{end} \end{array} \tag{7.2}$$

where κ is a user-defined parameter. In Figures 5(b) and 6(a), we compare the performances of Algorithm 9.1 with (7.1) and (7.2) respectively (we only plot the results of

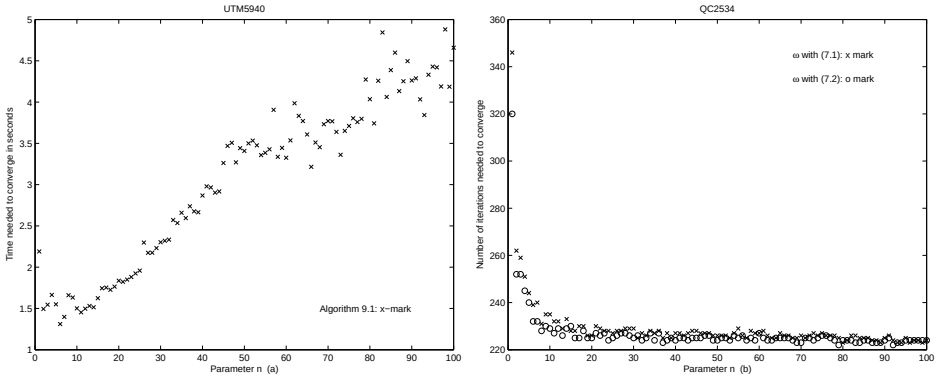

Figure 5: (a) utm5940: Graph of $T_{conv}(n)$ of Algorithm 9.1 against n. $T_{conv}(n)$ reaches its minimum at $n = 6$. (b) qc2534: Graphs of $I_{conv}(n)$ of Algorithm 9.1 against n with choices (7.1) and (7.2) for ω respectively. In this experiment, we picked $\kappa = 0.7$.

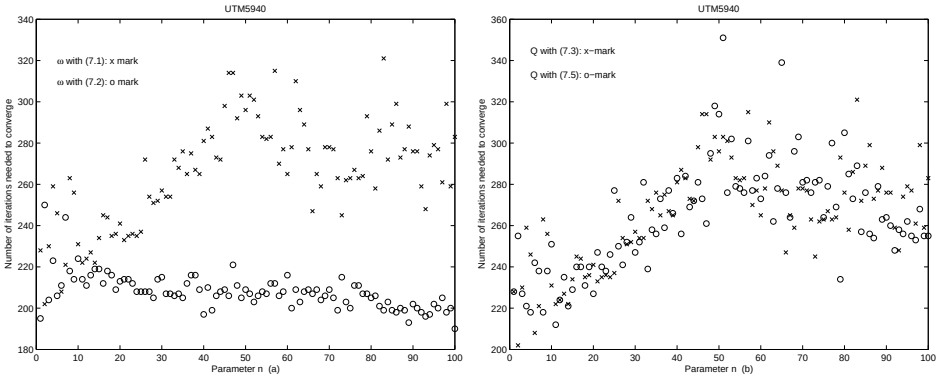

Figure 6: utm5940: (a) Graphs of $I_{conv}(n)$ of Algorithm 9.1 against n with choices (7.1) and (7.2) for ω respectively. In this experiment, we picked $\kappa = 0.7$. (b) Graphs of $I_{conv}(n)$ of Algorithm 9.1 against n with choices (7.3) and (7.5) for $\mathbf{Q}$ respectively.

qc2534 and *utm5940*. The result of *e20r0100* with $\kappa = 0.1$ is analogous to Figure 5(b)). Also, see the numerical experiments in [30] for more information about (7.2).

7.4. Choice of q's

We usually pick

$$\mathbf{Q} = [\mathbf{r}_0, randn(N, n-1)] \tag{7.3}$$

for a real problem and

$$\mathbf{Q} = [\mathbf{r}_0, randn(N, n-1) + sqrt(-1) * randn(N, n-1)] \tag{7.4}$$

for a complex problem. In our experiments, however, we observed a comparable performance when we chose

$$\mathbf{Q} = [\mathbf{r}_0, sign(randn(N, n-1))] \tag{7.5}$$

or

$$\mathbf{Q} = [\mathbf{r}_0, sign(randn(N, n-1)) + sqrt(-1) * sign(randn(N, n-1))]. \tag{7.6}$$

See Figure 6(b) (we only plot the result of *utm5940* for saving space).

The advantages of (7.5) and (7.6) over (7.3) and (7.4) are that (i) the storage of $\mathbf{Q}$ is substantially reduced. In fact, we just need to store the random signs (except its first column); (ii) an inner product with $\mathbf{q}_i$, $2 \le i \le n$, is now reduced to a sum without involving scalar multiplications.

For other choices for $\mathbf{Q}$, one is referred to [30].

8. Conclusions

With the help of index functions, we re-derived the ML(n)BiCGStab algorithm in [39] in a more systematic way. This time, we have been able to find out and remove some redundant operations so that the algorithm becomes more efficient. We also recognized that there were n ways to define the ML(n)BiCGStab residual $\mathbf{r}_k$. Each of the definitions leads to a different algorithm. We presented two definitions together with their associated algorithms, namely, (i) definition (4.3), increasing the degree of ϕ at the beginning of an iteration cycle, and the associated Algorithm 4.1; (ii) definition (5.2), increasing the degree of ϕ at the end of an iteration cycle, and the associated Algorithm 5.1. By comparison, Algorithm 5.1 is cheaper in storage and computational cost, faster to converge, but less stable. For other definitions of $\mathbf{r}_k$ that increase the degree of ϕ somewhere within a cycle, we expect that the associated algorithms would lie between Algorithms 4.1 and 5.1 in computational cost, storage and performance.

We showed that the Lanczos-based BiCG/BiCGStab and the Arnoldi-based FOM are the extreme cases of ML(n)BiCG /ML(n)BiCGStab.

In this paper, we did not assume that $\mathbf{A}$ is a nonsingular matrix. When a singular system (1.1) is solved, selecting an appropriate initial guess $\widehat{\mathbf{x}}_0$ is a crucial step. If $\widehat{\mathbf{x}}_0$ is selected such that the affine subspace $\widehat{\mathbf{x}}_0 + \mathbb{K}(\mathbf{A}, \widehat{\mathbf{r}}_0)$ contains a solution of (1.1), ML(n)BiCG will almost surely converge (see Theorem 3.1). Otherwise, we shall have $p_{min}(0, \mathbf{A}, \widehat{\mathbf{r}}_0) = 0$ (see Remark 3.2(ii)) which yields $\det(\widehat{\mathbf{S}}_\nu) = 0$ (see Lemma 3.2(c)). In this case, in the last iteration $k = \nu$, the LU-factorization in the construction of ML(n)BiCG does not exist (see Remark 3.1(ii)). As a result, it is likely that $\|\widehat{\mathbf{r}}_\nu\|_2$ blows up to ∞.[††] A similar remark also applies to ML(n)BiCGStab.

In the solution of a sequence of linear systems where BiCGStab is convergent throughout the sequence, the parameter n in ML(n)BiCGStab can be chosen dynamically.

9. Appendix

In this section, we present the preconditioned ML(n)BiCGStab algorithms together with their Matlab codes.

[††]Lemma 3.2, together with Remark 3.1(ii), indicates that ML(n)BiCG will run almost surely without encountering zero division from iteration $k = 1$ to iteration $k = \nu - 1$ in any situation.

9.1. ML(n)BiCGStab with Definition (4.3)

Algorithm 9.1 is a preconditioned version of Algorithm 4.1.

Algorithm 9.1. ML(n)BiCGStab with preconditioning associated with definition (4.3).

1. Choose an initial guess $\mathbf{x}_0$ and n vectors $\mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_n$.
2. Compute $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$ and set $\mathbf{g}_0 = \mathbf{r}_0$.
   Compute $\widetilde{\mathbf{g}}_0 = \mathbf{M}^{-1}\mathbf{g}_0$, $\mathbf{w}_0 = \mathbf{A}\widetilde{\mathbf{g}}_0$, $c_0 = \mathbf{q}_1^H\mathbf{w}_0$ and $e_0 = \mathbf{q}_1^H\mathbf{r}_0$.
3. For $j = 0, 1, 2, \cdots$
4. $\quad \alpha_{jn+1} = e_{(j-1)n+n}/c_{(j-1)n+n}$;
5. $\quad \mathbf{u}_{jn+1} = \mathbf{r}_{(j-1)n+n} - \alpha_{jn+1}\mathbf{w}_{(j-1)n+n}$;
6. $\quad \mathbf{x}_{jn+1} = \mathbf{x}_{(j-1)n+n} + \alpha_{jn+1}\widetilde{\mathbf{g}}_{(j-1)n+n}$;
7. $\quad \widetilde{\mathbf{u}}_{jn+1} = \mathbf{M}^{-1}\mathbf{u}_{jn+1}$;
8. $\quad \omega_{j+1} = (\mathbf{A}\widetilde{\mathbf{u}}_{jn+1})^H\mathbf{u}_{jn+1}/\|\mathbf{A}\widetilde{\mathbf{u}}_{jn+1}\|_2^2$;
9. $\quad \mathbf{x}_{jn+1} = \mathbf{x}_{jn+1} + \omega_{j+1}\widetilde{\mathbf{u}}_{jn+1}$;
10. $\quad \mathbf{r}_{jn+1} = -\omega_{j+1}\mathbf{A}\widetilde{\mathbf{u}}_{jn+1} + \mathbf{u}_{jn+1}$;
11. $\quad$ For $i = 1, 2, \cdots, n-1$
12. $\quad\quad f_{jn+i} = \mathbf{q}_{i+1}^H\mathbf{u}_{jn+i}$;
13. $\quad\quad$ If $j \geq 1$
14. $\quad\quad\quad \beta_{(j-1)n+i}^{(jn+i)} = -f_{jn+i}/c_{(j-1)n+i}$;
15. $\quad\quad\quad$ If $i \leq n-2$
16. $\quad\quad\quad\quad \mathbf{z}_d = \mathbf{u}_{jn+i} + \beta_{(j-1)n+i}^{(jn+i)}\mathbf{d}_{(j-1)n+i}$;
17. $\quad\quad\quad\quad \mathbf{g}_{jn+i} = \beta_{(j-1)n+i}^{(jn+i)}\mathbf{g}_{(j-1)n+i}$;
18. $\quad\quad\quad\quad \mathbf{z}_w = \beta_{(j-1)n+i}^{(jn+i)}\mathbf{w}_{(j-1)n+i}$;
19. $\quad\quad\quad\quad \beta_{(j-1)n+i+1}^{(jn+i)} = -\mathbf{q}_{i+2}^H\mathbf{z}_d/c_{(j-1)n+i+1}$;
20. $\quad\quad\quad\quad$ For $s = i+1, \cdots, n-2$
21. $\quad\quad\quad\quad\quad \mathbf{z}_d = \mathbf{z}_d + \beta_{(j-1)n+s}^{(jn+i)}\mathbf{d}_{(j-1)n+s}$;
22. $\quad\quad\quad\quad\quad \mathbf{g}_{jn+i} = \mathbf{g}_{jn+i} + \beta_{(j-1)n+s}^{(jn+i)}\mathbf{g}_{(j-1)n+s}$;
23. $\quad\quad\quad\quad\quad \mathbf{z}_w = \mathbf{z}_w + \beta_{(j-1)n+s}^{(jn+i)}\mathbf{w}_{(j-1)n+s}$;
24. $\quad\quad\quad\quad\quad \beta_{(j-1)n+s+1}^{(jn+i)} = -\mathbf{q}_{s+2}^H\mathbf{z}_d/c_{(j-1)n+s+1}$;
25. $\quad\quad\quad\quad$ End
26. $\quad\quad\quad\quad \mathbf{g}_{jn+i} = \mathbf{g}_{jn+i} + \beta_{(j-1)n+n-1}^{(jn+i)}\mathbf{g}_{(j-1)n+n-1}$;
27. $\quad\quad\quad\quad \mathbf{z}_w = \mathbf{z}_w + \beta_{(j-1)n+n-1}^{(jn+i)}\mathbf{w}_{(j-1)n+n-1}$;
28. $\quad\quad\quad\quad \mathbf{z}_w = \mathbf{r}_{jn+i} - \omega_{j+1}\mathbf{z}_w$;
29. $\quad\quad\quad$ Else
30. $\quad\quad\quad\quad \mathbf{g}_{jn+i} = \beta_{(j-1)n+n-1}^{(jn+i)}\mathbf{g}_{(j-1)n+n-1}$;
31. $\quad\quad\quad\quad \mathbf{z}_w = \mathbf{r}_{jn+i} - \omega_{j+1}\beta_{(j-1)n+n-1}^{(jn+i)}\mathbf{w}_{(j-1)n+n-1}$;
32. $\quad\quad\quad$ End
33. $\quad\quad\quad \beta_{(j-1)n+n}^{(jn+i)} = \mathbf{q}_1^H\mathbf{z}_w/(\omega_{j+1}c_{(j-1)n+n})$;
34. $\quad\quad\quad \mathbf{z}_w = \mathbf{z}_w - \omega_{j+1}\beta_{(j-1)n+n}^{(jn+i)}\mathbf{w}_{(j-1)n+n}$;

35. $\quad\quad \mathbf{g}_{jn+i} = \mathbf{g}_{jn+i} + \mathbf{z}_w + \beta^{(jn+i)}_{(j-1)n+n}\mathbf{g}_{(j-1)n+n}$;
36. $\quad$ Else
37. $\quad\quad \beta^{(jn+i)}_{(j-1)n+n} = \mathbf{q}_1^H\mathbf{r}_{jn+i}/(\omega_{j+1}c_{(j-1)n+n})$;
38. $\quad\quad \mathbf{z}_w = \mathbf{r}_{jn+i} - \omega_{j+1}\beta^{(jn+i)}_{(j-1)n+n}\mathbf{w}_{(j-1)n+n}$;
39. $\quad\quad \mathbf{g}_{jn+i} = \mathbf{z}_w + \beta^{(jn+i)}_{(j-1)n+n}\mathbf{g}_{(j-1)n+n}$;
40. $\quad$ End
41. $\quad$ For $s = 1, \cdots, i-1$
42. $\quad\quad \beta^{(jn+i)}_{jn+s} = -\mathbf{q}_{s+1}^H\mathbf{z}_w/c_{jn+s}$;
43. $\quad\quad \mathbf{g}_{jn+i} = \mathbf{g}_{jn+i} + \beta^{(jn+i)}_{jn+s}\mathbf{g}_{jn+s}$;
44. $\quad\quad \mathbf{z}_w = \mathbf{z}_w + \beta^{(jn+i)}_{jn+s}\mathbf{d}_{jn+s}$;
45. $\quad$ End
46. $\quad$ If $i < n-1$
47. $\quad\quad \mathbf{d}_{jn+i} = \mathbf{z}_w - \mathbf{u}_{jn+i}$;
48. $\quad\quad c_{jn+i} = \mathbf{q}_{i+1}^H\mathbf{d}_{jn+i}$;
49. $\quad\quad \widetilde{\alpha}_{jn+i+1} = -f_{jn+i}/c_{jn+i}$; $\quad$ % $\widetilde{\alpha}_{jn+i+1} = \alpha_{jn+i+1}/\omega_{j+1}$
50. $\quad\quad \mathbf{u}_{jn+i+1} = \mathbf{u}_{jn+i} + \widetilde{\alpha}_{jn+i+1}\mathbf{d}_{jn+i}$;
51. $\quad$ Else
52. $\quad\quad c_{jn+i} = \mathbf{q}_{i+1}^H(\mathbf{z}_w - \mathbf{u}_{jn+i})$;
53. $\quad\quad \widetilde{\alpha}_{jn+i+1} = -f_{jn+i}/c_{jn+i}$; $\quad$ % $\widetilde{\alpha}_{jn+i+1} = \alpha_{jn+i+1}/\omega_{j+1}$
54. $\quad$ End
55. $\quad$ $\widetilde{\mathbf{g}}_{jn+i} = \mathbf{M}^{-1}\mathbf{g}_{jn+i}$; $\mathbf{w}_{jn+i} = \mathbf{A}\widetilde{\mathbf{g}}_{jn+i}$;
56. $\quad$ $\mathbf{x}_{jn+i+1} = \mathbf{x}_{jn+i} + \omega_{j+1}\widetilde{\alpha}_{jn+i+1}\widetilde{\mathbf{g}}_{jn+i}$;
57. $\quad$ $\mathbf{r}_{jn+i+1} = \mathbf{r}_{jn+i} - \omega_{j+1}\widetilde{\alpha}_{jn+i+1}\mathbf{w}_{jn+i}$;
58. End
59. $e_{jn+n} = \mathbf{q}_1^H\mathbf{r}_{jn+n}$; $\beta^{(jn+n)}_{(j-1)n+n} = e_{jn+n}/(\omega_{j+1}c_{(j-1)n+n})$;
60. $\mathbf{z}_w = \mathbf{r}_{jn+n} - \omega_{j+1}\beta^{(jn+n)}_{(j-1)n+n}\mathbf{w}_{(j-1)n+n}$;
61. $\mathbf{g}_{jn+n} = \mathbf{z}_w + \beta^{(jn+n)}_{(j-1)n+n}\mathbf{g}_{(j-1)n+n}$;
62. If $n \geq 2$
63. $\quad \beta^{(jn+n)}_{jn+1} = -\mathbf{q}_2^H\mathbf{z}_w/c_{jn+1}$;
64. $\quad$ For $s = 1, \cdots, n-2$
65. $\quad\quad \mathbf{g}_{jn+n} = \mathbf{g}_{jn+n} + \beta^{(jn+n)}_{jn+s}\mathbf{g}_{jn+s}$;
66. $\quad\quad \mathbf{z}_w = \mathbf{z}_w + \beta^{(jn+n)}_{jn+s}\mathbf{d}_{jn+s}$;
67. $\quad\quad \beta^{(jn+n)}_{jn+s+1} = -\mathbf{q}_{s+2}^H\mathbf{z}_w/c_{jn+s+1}$;
68. $\quad$ End
69. $\quad$ $\mathbf{g}_{jn+n} = \mathbf{g}_{jn+n} + \beta^{(jn+n)}_{jn+n-1}\mathbf{g}_{jn+n-1}$;
70. End
71. $\widetilde{\mathbf{g}}_{jn+n} = \mathbf{M}^{-1}\mathbf{g}_{jn+n}$; $\mathbf{w}_{jn+n} = \mathbf{A}\widetilde{\mathbf{g}}_{jn+n}$;
72. $c_{jn+n} = \mathbf{q}_1^H\mathbf{w}_{jn+n}$;
73. End

Code #1: **Matlab code of Algorithm 9.1**

```
1. function [x,err,iter,flag] = mlbicgstab(A,x,b,Q,M,max_it,tol,kappa)
2.
3. % input:  A:  N-by-N matrix. M:  N-by-N preconditioner matrix.
4. %         Q: N-by-n auxiliary matrix [q_1,...,q_n]. x: initial guess.
5. %         b:  right hand side vector. max_it:  maximum number of iterations.
6. %         tol:  error tolerance.
7. %         kappa:  (real number in [0,1]) minimization step controller:
8. %                 kappa = 0, standard minimization
9. %                 kappa > 0, Sleijpen-van der Vorst minimization
10.%output: x: solution computed. err: error norm. iter: number of iterations performed.
11.%         flag:     = 0, solution found to tolerance
12.%                   = 1, no convergence given max_it iterations
13.%                   = -1, breakdown.
14.% storage:D: N x (n-2) matrix defined only when n > 2.
15.%         G,Q,W: N x n matrices. A,M: N x N matrices.
16.%         x,r,g_t,u,z,b: N x 1 matrices. c: 1 x n matrix.
17.
18.    N = size(A,2); n = size(Q,2);
19.    G = zeros(N,n); W = zeros(N,n);    % initialize work spaces
20.    if n > 2,  D = zeros(N,n-2);  end
21.    c = zeros(1,n);       % end initialization
22.
23.    iter = 0; flag = 1; bnrm2 = norm(b);
24.    if bnrm2 == 0.0, bnrm2 = 1.0; end
25.
26.    r = b - A*x; err = norm(r)/bnrm2;
27.    if err < tol, flag = 0; return,  end
28.
29.    G(:,n) = r; g_t = M\r; W(:,n) = A*g_t; c(n) = Q(:,1)'*W(:,n);
30.    if c(n) == 0,  flag = -1; return, end
31.    e = Q(:,1)'*r;
32.
33.    for j = 0 : max_it
34.        alpha = e/c(n);  x = x + alpha*g_t;
35.        u = r - alpha*W(:,n);  err = norm(u)/bnrm2;
36.        if err < tol, flag = 0; iter = iter + 1; return, end
37.
38.        g_t = M\u; z = A*g_t; omega = z'*z;
39.        if omega == 0, flag = -1; return, end
40.        rho = z'*u; omega = rho/omega;
41.        if kappa > 0
42.            rho = rho/(norm(z)*norm(u)); abs_rho = abs(rho);
43.            if (abs_rho < kappa) & (abs_rho ~= 0), omega = omega*kappa/abs_rho;
                    end
44.        end
```

```
45.      x = x + omega * g_t;  r = -omega * z + u;
46.      err = norm(r)/bnrm2;  iter = iter + 1;
47.      if err < tol,  flag = 0;  return,  end
48.      if iter >= max_it,  return,  end
49.          50.          rc = omega * c(n);
51.      if rc == 0,  flag = -1;  return,  end
52.      for i = 1 : n - 1
53.          f = Q(:, i + 1)' * u;
54.          if j >= 1
55.              beta = -f / c(i);
56.              if i <= n - 2
57.                  D(:, i) = u + beta * D(:, i);  G(:, i) = beta * G(:, i);
58.                  W(:, i) = beta * W(:, i);  beta = -Q(:, i + 2)' * D(:, i)/c(i + 1);
59.                  for s = i + 1 : n - 2
60.                      D(:, i) = D(:, i) + beta * D(:, s);
61.                      G(:, i) = G(:, i) + beta * G(:, s);
62.                      W(:, i) = W(:, i) + beta * W(:, s);
63.                      beta = -Q(:, s + 2)' * D(:, i)/c(s + 1);
64.                  end
65.                  G(:, i) = G(:, i) + beta * G(:, n - 1);
66.                  W(:, i) = W(:, i) + beta * W(:, n - 1);
67.                  W(:, i) = r - omega * W(:, i);
68.              else
69.                  G(:, n - 1) = beta * G(:, n - 1);
70.                  W(:, n - 1) = r - (omega * beta) * W(:, n - 1);
71.              end
72.              beta = Q(:, 1)' * W(:, i)/rc;
73.              W(:, i) = W(:, i) - (omega * beta) * W(:, n);
74.              G(:, i) = G(:, i) + W(:, i) + beta * G(:, n);
75.          else
76.              beta = Q(:, 1)' * r/rc;
77.              W(:, i) = r - (omega * beta) * W(:, n);
78.              G(:, i) = W(:, i) + beta * G(:, n);
79.          end
80.          for s = 1 : i - 1
81.              beta = -Q(:, s + 1)' * W(:, i)/c(s);
82.              G(:, i) = G(:, i) + beta * G(:, s);  W(:, i) = W(:, i) + beta * D(:, s);
83.          end
84.          if i < n - 1
85.              D(:, i) = W(:, i) - u;  c(i) = Q(:, i + 1)' * D(:, i);
86.              if c(i) == 0,  flag = -1;  return,  end
87.              alpha = -f / c(i);  u = u + alpha * D(:, i);
88.          else
89.              c(i) = Q(:, i + 1)' * (W(:, i) - u);
90.              if c(i) == 0,  flag = -1;  return,  end
91.              alpha = -f / c(i);
```

```
92.             end
93.             g_t = M\G(:,i); W(:,i) = A*g_t;
94.             alpha = omega*alpha; x = x + alpha*g_t; r = r - alpha*W(:,i);
95.             err = norm(r)/bnrm2; iter = iter + 1;
96.             if err < tol, flag = 0; return, end
97.             if iter >= max_it, return, end
98.         end
99.         e = Q(:,1)'*r; beta = e/rc; W(:,n) = r - (omega*beta)*W(:,n);
100.        G(:,n) = W(:,n) + beta*G(:,n);
101.        if n >= 2
102.            beta = -Q(:,2)'*W(:,n)/c(1);
103.            for s = 1 : n-2
104.                G(:,n) = G(:,n) + beta*G(:,s); W(:,n) = W(:,n) + beta*D(:,s);
105.                beta = -Q(:,s+2)'*W(:,n)/c(s+1);
106.            end
107.            G(:,n) = G(:,n) + beta*G(:,n-1);
108.        end
109.        g_t = M\G(:,n); W(:,n) = A*g_t; c(n) = Q(:,1)'*W(:,n);
110.        if c(n) == 0, flag = -1; return, end
111.    end
```

9.2. ML(n)BiCGStab with Definition (5.2)

Algorithm 9.2 is a preconditioned version of Algorithm 5.1.

Algorithm 9.2 ML(n)BiCGStab with preconditioning associated with definition (5.2).

1. Choose an initial guess $\mathbf{x}_0$ and n vectors $\mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_n$.
2. Compute $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$, $\widetilde{\mathbf{g}}_0 = \mathbf{M}^{-1}\mathbf{r}_0$, $\mathbf{w}_0 = \mathbf{A}\widetilde{\mathbf{g}}_0$, $c_0 = \mathbf{q}_1^H\mathbf{w}_0$ and $e_0 = \mathbf{q}_1^H\mathbf{r}_0$.
3. For $j = 0, 1, 2, \cdots$
4. For $i = 1, 2, \cdots, n-1$
5. $\alpha_{jn+i} = e_{jn+i-1}/c_{jn+i-1}$;
6. $\mathbf{x}_{jn+i} = \mathbf{x}_{jn+i-1} + \alpha_{jn+i}\widetilde{\mathbf{g}}_{jn+i-1}$; % $\widetilde{\mathbf{g}} = \mathbf{M}^{-1}\mathbf{g}$
7. $\mathbf{r}_{jn+i} = \mathbf{r}_{jn+i-1} - \alpha_{jn+i}\mathbf{w}_{jn+i-1}$;
8. $e_{jn+i} = \mathbf{q}_{i+1}^H\mathbf{r}_{jn+i}$;
9. If $j \geq 1$
10. $\tilde{\beta}_{(j-1)n+i}^{(jn+i)} = -e_{jn+i}/c_{(j-1)n+i}$; % $\tilde{\beta}_{(j-1)n+i}^{(jn+i)} = -\omega_j\beta_{(j-1)n+i}^{(jn+i)}$
11. $\mathbf{z}_w = \mathbf{r}_{jn+i} + \tilde{\beta}_{(j-1)n+i}^{(jn+i)}\mathbf{w}_{(j-1)n+i}$;
12. $\widetilde{\mathbf{g}}_{jn+i} = \tilde{\beta}_{(j-1)n+i}^{(jn+i)}\widetilde{\mathbf{g}}_{(j-1)n+i}$;
13. For $s = i+1, \cdots, n-1$
14. $\tilde{\beta}_{(j-1)n+s}^{(jn+i)} = -\mathbf{q}_{s+1}^H\mathbf{z}_w/c_{(j-1)n+s}$; % $\tilde{\beta}_{(j-1)n+s}^{(jn+i)} = -\omega_j\beta_{(j-1)n+s}^{(jn+i)}$

15. $\quad\quad\quad \mathbf{z}_w = \mathbf{z}_w + \tilde{\beta}^{(jn+i)}_{(j-1)n+s}\mathbf{w}_{(j-1)n+s}$;
16. $\quad\quad\quad \widetilde{\mathbf{g}}_{jn+i} = \widetilde{\mathbf{g}}_{jn+i} + \tilde{\beta}^{(jn+i)}_{(j-1)n+s}\widetilde{\mathbf{g}}_{(j-1)n+s}$;
17. $\quad\quad$ End
18. $\quad\quad \widetilde{\mathbf{g}}_{jn+i} = \mathbf{M}^{-1}\mathbf{z}_w - \dfrac{1}{\omega_j}\widetilde{\mathbf{g}}_{jn+i}$;
19. $\quad$ Else
20. $\quad\quad \widetilde{\mathbf{g}}_{jn+i} = \mathbf{M}^{-1}\mathbf{r}_{jn+i}$;
21. $\quad$ End
22. $\quad \mathbf{w}_{jn+i} = \mathbf{A}\widetilde{\mathbf{g}}_{jn+i}$;
23. $\quad$ For $s = 0, \cdots, i-1$
24. $\quad\quad \beta^{(jn+i)}_{jn+s} = -\mathbf{q}^H_{s+1}\mathbf{w}_{jn+i}/c_{jn+s}$;
25. $\quad\quad \mathbf{w}_{jn+i} = \mathbf{w}_{jn+i} + \beta^{(jn+i)}_{jn+s}\mathbf{w}_{jn+s}$;
26. $\quad\quad \widetilde{\mathbf{g}}_{jn+i} = \widetilde{\mathbf{g}}_{jn+i} + \beta^{(jn+i)}_{jn+s}\widetilde{\mathbf{g}}_{jn+s}$;
27. $\quad$ End
28. $\quad c_{jn+i} = \mathbf{q}^H_{i+1}\mathbf{w}_{jn+i}$;
29. End
30. $\alpha_{jn+n} = e_{jn+n-1}/c_{jn+n-1}$;
31. $\mathbf{x}_{jn+n} = \mathbf{x}_{jn+n-1} + \alpha_{jn+n}\widetilde{\mathbf{g}}_{jn+n-1}$;
32. $\mathbf{u}_{jn+n} = \mathbf{r}_{jn+n-1} - \alpha_{jn+n}\mathbf{w}_{jn+n-1}$;
33. $\widetilde{\mathbf{u}}_{jn+n} = \mathbf{M}^{-1}\mathbf{u}_{jn+n}$;
34. $\omega_{j+1} = (\mathbf{A}\widetilde{\mathbf{u}}_{jn+n})^H\mathbf{u}_{jn+n}/\|\mathbf{A}\widetilde{\mathbf{u}}_{jn+n}\|_2^2$;
35. $\mathbf{x}_{jn+n} = \mathbf{x}_{jn+n} + \omega_{j+1}\widetilde{\mathbf{u}}_{jn+n}$;
36. $\mathbf{r}_{jn+n} = -\omega_{j+1}\mathbf{A}\widetilde{\mathbf{u}}_{jn+n} + \mathbf{u}_{jn+n}$;
37. $e_{jn+n} = \mathbf{q}^H_1\mathbf{r}_{jn+n}$;
38. $\tilde{\beta}^{(jn+n)}_{(j-1)n+n} = -e_{jn+n}/c_{(j-1)n+n}$; $\quad$ % $\tilde{\beta}^{(jn+n)}_{(j-1)n+n} = -\omega_{j+1}\beta^{(jn+n)}_{(j-1)n+n}$
39. $\mathbf{z}_w = \mathbf{r}_{jn+n} + \tilde{\beta}^{(jn+n)}_{(j-1)n+n}\mathbf{w}_{(j-1)n+n}$;
40. $\widetilde{\mathbf{g}}_{jn+n} = \tilde{\beta}^{(jn+n)}_{(j-1)n+n}\widetilde{\mathbf{g}}_{(j-1)n+n}$;
41. For $s = 1, \cdots, n-1$
42. $\quad \tilde{\beta}^{(jn+n)}_{jn+s} = -\mathbf{q}^H_{s+1}\mathbf{z}_w/c_{jn+s}$; $\quad$ % $\tilde{\beta}^{(jn+n)}_{jn+s} = -\omega_{j+1}\beta^{(jn+n)}_{jn+s}$
43. $\quad \mathbf{z}_w = \mathbf{z}_w + \tilde{\beta}^{(jn+n)}_{jn+s}\mathbf{w}_{jn+s}$;
44. $\quad \widetilde{\mathbf{g}}_{jn+n} = \widetilde{\mathbf{g}}_{jn+n} + \tilde{\beta}^{(jn+n)}_{jn+s}\widetilde{\mathbf{g}}_{jn+s}$;
45. End
46. $\widetilde{\mathbf{g}}_{jn+n} = \mathbf{M}^{-1}\mathbf{z}_w - \dfrac{1}{\omega_{j+1}}\widetilde{\mathbf{g}}_{jn+n}$; $\mathbf{w}_{jn+n} = \mathbf{A}\widetilde{\mathbf{g}}_{jn+n}$;
47. $c_{jn+n} = \mathbf{q}^H_1\mathbf{w}_{jn+n}$;
48. End

Code #2: **Matlab code of Algorithm 9.2**

```
1. function [x,err,iter,flag] = mlbicgstab(A,x,b,Q,M,max_it,tol,kappa)
2.
```

```
3.  % The input/output arguments are described as in Code #1.
4.  % storage:    c: 1 × n matrix. x, r, b, u_t, z: N-by-1 matrices.
5.  %             A, M: N-by-N matrices. Q, G, W: N-by-n matrices.
6.
7.      N = size(A,2); n = size(Q,2);
8.      G = zeros(N,n); W = zeros(N,n);    % initialize work spaces
9.      c = zeros(1,n);          % end initialization
10.
11.     iter = 0; flag = 1; bnrm2 = norm(b);
12.     if bnrm2 == 0.0, bnrm2 = 1.0; end
13.     r = b - A*x; err = norm(r)/bnrm2;
14.     if err < tol, flag = 0; return, end
15.
16.     G(:,1) = M\r; W(:,1) = A*G(:,1); c(1) = Q(:,1)'*W(:,1);
17.     if c(1) == 0, flag = -1; return, end
18.     e = Q(:,1)'*r;
19.
20.     for j = 0 : max_it
21.         for i = 1 : n-1
22.             alpha = e/c(i); x = x + alpha*G(:,i);  r = r - alpha*W(:,i);
23.             err = norm(r)/bnrm2; iter = iter + 1;
24.             if err < tol, flag = 0; return, end
25.             if iter >= max_it, return, end
26.
27.             e = Q(:,i+1)'*r;
28.             if j >= 1
29.                 beta = -e/c(i+1); W(:,i+1) = r + beta*W(:,i+1);
30.                 G(:,i+1) = beta*G(:,i+1);
31.                 for s = i+1 : n-1
32.                     beta = -Q(:,s+1)'*W(:,i+1)/c(s+1);
33.                     W(:,i+1) = W(:,i+1) + beta*W(:,s+1);
34.                     G(:,i+1) = G(:,i+1) + beta*G(:,s+1);
35.                 end
36.                 G(:,i+1) = (M\W(:,i+1)) - (1/omega)*G(:,i+1);
37.             else
38.                 G(:,i+1) = M\r;
39.             end
40.             W(:,i+1) = A*G(:,i+1);
41.             for s = 0 : i-1
42.                 beta = -Q(:,s+1)'*W(:,i+1)/c(s+1);
43.                 W(:,i+1) = W(:,i+1) + beta*W(:,s+1);
44.                 G(:,i+1) = G(:,i+1) + beta*G(:,s+1);
45.             end
46.             c(i+1) = Q(:,i+1)'*W(:,i+1);
47.             if c(i+1) == 0, flag = -1; return, end
48.         end
```

```
49.       alpha = e/c(n);  x = x + alpha * G(:,n);
50.       r = r - alpha * W(:,n);  err = norm(r)/bnrm2;
51.       if err < tol,  flag = 0;  iter = iter + 1;  return,  end
52.       u_t = M\r;  z = A * u_t;  omega = z' * z;
53.       if omega == 0,  flag = -1;  return,  end
54.       rho = z' * r;  omega = rho/omega;
55.       if kappa > 0
56.           rho = rho/(norm(z) * norm(r));  abs_rho = abs(rho);
57.           if (abs_rho < kappa) & (abs_rho ~= 0)
58.               omega = omega * kappa/abs_rho;
59.           end
60.       end
61.       if omega == 0,  flag = -1;  return,  end
62.       x = x + omega * u_t;  r = r - omega * z;
63.       err = norm(r)/bnrm2;  iter = iter + 1;
64.       if err < tol,  flag = 0;  return,  end
65.       if iter >= max_it,  return,  end
66.
67.       e = Q(:,1)' * r;  beta = -e/c(1);
68.       W(:,1) = r + beta * W(:,1);  G(:,1) = beta * G(:,1);
69.       for s = 1 : n - 1
70.           beta = -Q(:,s + 1)' * W(:,1)/c(s + 1);
71.           W(:,1) = W(:,1) + beta * W(:,s + 1);
72.           G(:,1) = G(:,1) + beta * G(:,s + 1);
73.       end
74.       G(:,1) = (M\W(:,1)) - (1/omega) * G(:,1);  W(:,1) = A * G(:,1);
75.       c(1) = Q(:,1)' * W(:,1);
76.       if c(1) == 0,  flag = -1;  return,  end
77.   end
```

9.3. Sample Executions of ML(n)BiCGStab

We provide two sample executions: Code #3 for a single system and Code #4 for a sequence of systems.

Code #3: **A sample run of ML(n)BiCGstab**

```
1. N = 100; A = randn(N); M = randn(N); b = randn(N,1);
2. n = 10; tol = 10^-7; max_it = 3 * N; kappa = 0;    % or, say, kappa = 0.7
3. Q = sign(randn(N,n)); x = zeros(N,1); Q(:,1) = b - A * x;
4. [x,err,iter,flag] = mlbicgstab(A,x,b,Q,M,max_it,tol,kappa);
```

Code #4: **Solution of a sequence of linear systems**

```
% Suppose ml(n)bicgstab is used to solve a sequence of m systems A_i x = b_i with
% preconditioners M_i. This code dynamically searches for n in an user-provided
% interval [n_min, n_max] so that the time per iteration is as small as possible.

n_min = 2; n_max = 20; step = 3;    % step size, an integer >= 1.
max_it = 3*N; tol = 10^-7; kappa = 0;    % or, say, kappa = 0.7
Q = sign(randn(N, n_max));       % a random sign shadow matrix.
n = 10;          % initial value for n, an integer picked from [n_min, n_max].
walk = 1;        % walk = 1, search forward; = -1, search backward.
t1 = inf;        % solution time of the previous system.

for i = 1 : m
    x = zeros(N, 1);       % choose an initial guess for the ith system.
    Q(:, 1) = b_i - A_i * x;
    tic
    [x, err, iter, flag] = mlbicgstab(A_i, x, b_i, Q(:, 1 : n), M_i, max_it, tol, kappa);
    t2 = toc / iter;    % time per iteration of the current system.
    if walk == 1
        if t2 < t1
            n = min(n + step, n_max); t1 = t2;
        else
            n = max(n - step, n_min); t1 = t2; walk = -1;
        end
    else
        if t2 < t1
            n = max(n - step, n_min); t1 = t2;
        else
            n = min(n + step, n_max); t1 = t2; walk = 1;
        end
    end
end
```

Acknowledgements The index functions were introduced by Prof. Daniel Boley in [38]. Without the index functions, this research work would have become too complicated to be possible. The author is grateful to Prof. Martin Gutknecht for reading the manuscript carefully, helping him revise it and pointing out some useful references. Special thanks go to Prof. Martin Gijzen, Prof. Peter Sonneveld and Dr. Jens-Peter M. Zemke for their reading of the whole manuscript and making helpful comments and suggestions. The author also acknowledges Dr. Jens-Peter M. Zemke for the helpful communications and up-to-date information about Krylov subspace methods. The author is grateful to the referees for many valuable suggestions that improve this paper. The author is grateful to his Ph.D. advisor Prof. Tony Chan for guiding him into the field of Krylov subspace methods. The method was originally named MLBiCGStab(n) by him. The current and more accurate name was from Prof. Henk A. van der Vorst. Part of this work was done during the visit of

the author at Delft University of Technology, The Netherlands, in June 2009.

References

[1] J. Aliaga, D. Boley, R. Freund and V. Hernández, *A Lanczos-type method for multiple starting vectors*, Math. Comp. 69 (2000), pp. 1577-1601.

[2] P. N. Brown and H. F. Walker, *GMRES on (nearly) singular systems*, SIAM J. Matrix Anal. Appl., 18(1997), pp. 37-51.

[3] A. El Guennouni, K. Jbilou and H. Sadok, *A block version of BiCGSTAB for linear systems with multiple right-hand sides,* ETNA 16 (2003), pp. 129-142.

[4] R. Fletcher, *Conjugate Gradient Methods for Indefinite Systems,* volume 506 of Lecture Notes Math., pp. 73-89. Springer-Verlag, Berlin-Heidelberg-New York, 1976.

[5] A. Gaul, M. Gutknecht, J. Liesen and R. Nabben, *Deflated and augmented Krylov subspace methods: basic facts and a breakdown-free deflated MINRES*, Preprint, DFG Research Center Matheon, 2011.

[6] M. H. Gutknecht, *A completed theory of the unsymmetric Lanczos process and related algorithms. Part I.*, SIAM J. Matrix Anal. Appl. 1992, 13:594-639.

[7] ——, *Variants of BICGStab for matrices with complex spectrum,* SIAM J. Sci. Comput., 14 (1993), pp. 1020–1033.

[8] ——, *A completed theory of the unsymmetric Lanczos process and related algorithms. Part II.*, SIAM J. Matrix Anal. Appl. 1994, 15:15-58.

[9] ——, *Lanczos-type solvers for nonsymmetric linear systems of equations,* Acta Numerica, 6 (1997), pp. 271-397.

[10] ——, *IDR Explained,* ETNA 36 (2010), pp. 126–148.

[11] R. Horn and C. Johnson, *Matrix Analysis*, Cambridge University Press, 1985.

[12] W. D. Joubert, *Generalized Conjugate Gradient and Lanczos Methods for the Solution of Nonsymmetric Systems of Linear Equations*, Ph.D. Thesis and Tech. Report CNA-238, Center for Numerical Analysis, University of Texas, Austin, TX, 1990.

[13] ——, *Lanczos methods for the solution of nonsymmetric systems of linear equations*, SIAM Journal on Matrix Analysis and Applications 1992; 13:926-943.

[14] E. F. Kaasschieter, *Preconditioned conjugate gradients for solving singular systems*, Journal of Computational and Applied Mathematics 1988; 24:265-275.

[15] C. Lanczos, *Solution of systems of linear equations by minimized iterations,* J. Research Nat. Bureau of Standards, 49 (1952), pp. 33–53.

[16] D. O'Leary, *The block conjugate gradient algorithm and related methods,* Linear Algebra Appl., 29(1980), pp. 293-322.

[17] K. Moriya and T. Nodera, *Breakdown-free ML(k)BiCGStab algorithm for non-Hermitian linear systems,* O. Gervasi et al. (Eds.): ICCSA 2005, LNCS 3483, pp. 978-988, 2005.

[18] L. Reichel and Q. Ye, *Breakdown-free GMRES for singular systems*, SIAM Journal on Matrix Analysis and Applications 2005; 26:1001-1021.

[19] Y. Saad, *The Lanczos biorthogonalization algorithm and other oblique projection methods for solving large unsymmetric systems*, SIAM Journal on Numerical Analysis, 19(1982), pp. 485-506.

[20] ——, *Iterative methods for sparse linear systems*, 2nd edition, SIAM, Philadelphia, PA, 2003.

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

[22] Y. Saad and H. A. van der Vorst, *Iterative solution of linear systems in the 20-th century,* J. Comp. and Appl. Math., 123(1-2):1-33, 2000.

[23] G. L. G. SLEIJPEN AND D. R. FOKKEMA, *BiCGSTAB(l) for linear equations involving unsymmetric matrices with complex spectrum,* ETNA, 1:11-32, 1993.
[24] G. L.G. SLEIJPEN, P. SONNEVELD, AND M. B. VAN GIJZEN, *Bi-CGSTAB as an induced dimension reduction method,* Applied Numerical Mathematics. Vol 60, pp. 1100-1114, 2010.
[25] G. L. G. SLEIJPEN AND H. A. VAN DER VORST, *Maintaining convergence properties of BiCGSTAB methods in finite precision arithmetic,* Numer. Algorithms, 10 (1995), pp. 203–223.
[26] ——, *Reliable updated residuals in hybrid Bi-CG methods,* Computing, 56 (1996), pp. 141–163.
[27] G. L. G. SLEIJPEN, H. A. VAN DER VORST, AND D. R. FOKKEMA, *BiCGstab(l) and other hybrid Bi-CG methods,* Numerical Algorithms, 7 (1994), pp. 75-109. Received Oct. 29, 1993.
[28] P. SONNEVELD, *CGS, a fast Lanczos-type solver for nonsymmetric linear systems,* SIAM J. Sci. Statist. Comput., 10 (1989), pp. 36–52.
[29] P. SONNEVELD AND M. VAN GIJZEN, *IDR(s): a family of simple and fast algorithms for solving large nonsymmetric linear systems*, Delft University of Technology, Reports of the Department of Applied Mathematical Analysis, Report 07-07.
[30] ——, *IDR(s): a family of simple and fast algorithms for solving large nonsymmetric linear systems,* SIAM J. Sci. Comput. Vol. 31, No. 2, pp. 1035-1062.
[31] 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. Statist. Comput., 12 (1992), pp. 631–644.
[32] ——, *Iterative Krylov Methods for Large Linear Systems,* Cambridge University Press, Cambridge, April 2003.
[33] H. VAN DER VORST AND Q. YE, *Residual Replacement Strategies for Krylov Subspace Iterative Methods for the Convergence of True Residuals*, SIAM J. Sci. Comput., 22 (2000):836-852.
[34] M. VAN GIJZEN AND P. SONNEVELD, *An elegant IDR(s) variant that efficiently exploits bi-orthogonality properties*, to appear in ACM Trans. Math. Software.
[35] Y. WEI AND H. WU, *Convergence properties of Krylov subspace methods for singular linear systems with arbitrary index*, Journal of Computational and Applied Mathematics 2000; 114:305-318.
[36] P. WESSELING AND P. SONNEVELD, *Numerical experiments with a multiple grid and a preconditioned Lanczos type method*, in Approximation methods for Navier-Stokes problems (Proc. Sympos., Univ. Paderborn, Paderborn, 1979), vol. 771 of Lecture Notes in Math., Springer, Berlin, 1980, pp. 543-562.
[37] M. YEUNG, *An introduction to ML(n)BiCGStab*, available at http://arxiv.org/abs/1106.3678. Proceedings of Boundary Elements and Other Mesh Reduction Methods XXXIV, edited by Brebbia & Poljak 2012, WITpress.
[38] M. YEUNG AND D. BOLEY, *Transpose-free multiple Lanczos and its application in Padé approximation*, Journal of Computational and Applied Mathematics, Vol 177/1 pp. 101-127, 2005.
[39] M. YEUNG AND T. CHAN, *ML(k)BiCGSTAB: A BiCGSTAB variant based on multiple Lanczos starting vectors*, SIAM J. Sci. Comput., Vol. 21, No. 4, pp. 1263-1290, 1999.
[40] S.-L. ZHANG, *GPBi-CG: Generalized product-type methods based on Bi-CG for solving nonsymmetric linear systems*, SIAM J. Sci. Comput., 18:537-551, 1997.