ML($\lambda$)BICGSTAB: REFORMULATION, ANALYSIS AND IMPLEMENTATION
MAN-CHUNG YEUNG*

Abstract. With the aid of index functions, we re-derive the ML($n$)BICGSTab algorithm in [35] in a more systematic way. It turns out that there are $n$ ways to define the ML($n$)BICGSTab residual vector. Each definition will lead 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 connecting BiCG and FOM while ML($n$)BiCG a bridge connecting BiCG and FOM. We also analyze the convergence of ML($n$)BiCG/ML($n$)BiCGStab from the probabilistic point of view when a singular system is solved, and summarize some of their useful properties. Implementation issues are also addressed.

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

AMS subject classifications. Primary, 65F10, 65F15; Secondary, 65F25, 65F30.

1. Introduction. If we express the BiCG[4, 13] residual as $r_{k}^{BiCG} = p_{k}(A)r_{0}$ in terms of a polynomial, the residual vector $r_{k}$ of a Lanczos-type product method based on BiCG is defined to be $r_{k} = \phi_{k}(A)p_{k}(A)r_{0}$ where $\phi_{k}(\lambda)$ is some polynomial of degree $k$ with $\phi_{k}(0) = 1$. In CGS[26], $\phi_{k} = p_{k}$. Since, in every iteration, CGS searches for an approximate solution in a larger Krylov subspace, it often converges much faster than BiCG. However, CGS usually behaves irregularly due to a lack of a smoothing mechanism. In BiCGStab[29], the $\phi_{k}$ is

$$
\phi_{k}(\lambda) = \begin{cases} 
1 & \text{if } k = 0 \\
(p_{k} + 1)\phi_{k-1}(\lambda) & \text{if } k > 0.
\end{cases}
$$

Here $\rho_{k}$ is a free parameter selected to minimize the 2-norm of $r_{k}^{BiCGStab}$ in the $k$-iteration. As a result, BiCGStab is generally more stable and robust than CGS. BiCGStab has been extended to BiCGStab2[6] and BiCGStab($l$)[20, 25] through the use of higher degree minimizing polynomials. In BiCGStab2, the $\phi_{k}$ is defined by the recursion

$$
\phi_{k}(\lambda) = \begin{cases} 
1 & \text{if } k = 0 \\
(p_{k} + 1)\phi_{k-1}(\lambda) & \text{if } k \text{ is odd} \\
((\alpha_{k}\lambda + \beta_{k})(\rho_{k} - 1\lambda + 1) + 1 - \beta_{k})\phi_{k-2}(\lambda) & \text{if } k \text{ is even}
\end{cases}
$$

The parameters are again chosen to minimize BiCGStab2 residuals. Likewise, BiCGStab($l$) defines its $\phi_{k}$ 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 $l$th step. BiCGStab2 and BiCGStab($l$) usually converge faster than BiCGStab because of smaller residuals in magnitude while avoiding near-breakdowns.

*Dept. 3036, 1000 East University Avenue, Laramie, WY 82071. E-mail: myeung@uwyo.edu. This research was supported by Flittie Sabbatical Augmentation Award, University of Wyoming.

1For this type of Krylov subspace methods, one can consult [8]. They are called hybrid BiCG methods in [23].
caused by a possibly too small $\rho_k$. CGS, BiCGStab and BiCGStab2 have been summarized and generalized by GPBi-CG[36]. Here $\phi_k$ is

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

GPBi-CG will become CGS, BiCGStab or BiCGStab2 when the $\alpha, \beta, \rho$ are appropriately chosen. For detailed descriptions of these and other product-type methods, one is referred to [5, 7, 17, 19, 30] and the references therein. Moreover, a history of product-type methods can be found in [9]. The history starts three decades ago with IDR[33] method which can be considered as the predecessor of CGS and BiCGStab[21]. Recently, IDR has been generalized to IDR($s$) with a shadow space of $s$ dimensions, see [21, 28, 32].

Generalizations of BiCGStab to methods based on generalizations of BiCG have been made. For example, BL-BiCGStab[3] is a BiCGStab variant built on the BL-BiCG[14] for the solution of systems with multiple right-hand sides. ML($n$)BiCGStab[35] 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 [35] 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 [34] by Boley for the purpose of simplifying the development of the transpose-free multiple starting Lanczos process, and they proved to be very helpful.

It turns out that the definition of the ML($n$)BiCGStab residual vector $r_k$ in [35] is not unique. There are at least $n$ different ways to define $r_k$. Let $\tilde{r}_k$ be the residual of ML($n$)BiCG and $\phi_k(\lambda)$ as in (1.1). Then, the ML($n$)BiCGStab residual $r_k$ in [35] is defined as

$$r_k = \phi_j(A)\tilde{r}_k$$

where $k = jn + i$, $1 \leq i \leq n, j = 0, 1, 2, \cdots$. Starting from $k = 1$, let us call every $n$ consecutive 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 (1.2) increases the degree of $\phi$ by 1 at the beginning of a cycle. One actually can define $r_k$ by increasing the degree of $\phi$ by 1 anywhere within an iteration cycle. Each definition will lead to a different ML($n$)BiCGStab algorithm. As an illustration, we derive a second ML($n$)BiCGStab algorithm associated with the definition

$$r_{jn+i} = \begin{cases} \phi_j(A)\tilde{r}_{jn+i} & \text{if } 1 \leq i \leq n-1 \\ \phi_{j+1}(A)\tilde{r}_{jn+i} & \text{if } i = n. \end{cases}$$

(1.3) increases the degree of $\phi$ 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 (1.2). However, one drawback with this storage-saving algorithm is that its computed residual $r_k$ easily diverges from the corresponding exact residual when $n$ is moderately large.

Both ML($n$)BiCG and ML($n$)BiCGStab possess a set of left starting vectors (or, shadow vectors) $q_1, \cdots, q_n$ that can be chosen freely. This freedom appears to be an advantage of the methods. It not only helps stabilize the performance
of the algorithms, but also allows to see a connection between the Lanczos-based BiCG/BiCGStab and the Arnoldi-based FOM.

Just like BiCGStab, ML(n)BiCGStab can suffer from three types of breakdown, caused respectively by the failure of the underlying Lanczos process, the non-existence of the LU factorization during the construction of ML(n)BiCG and the parameters $\rho_k$. We prove that the breakdown probability is zero when the shadow vectors are selected randomly.

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 [35], from which ML(n)BiCGStab algorithms are derived. In §4, we rederive the ML(n)BiCGStab algorithm in [35] 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. Concluding remarks are made in §8.

2. Index Functions. Let be given a positive integer $n$. For all integers $k$, we define

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

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 $\{1, 2, \ldots, n\}$, respectively.

If we write

$$k = jn + i$$

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

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

Table 2.1 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 $n$.

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

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

(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 $\max(k - n, 0) \leq s \leq g_n(k)n - 1$.

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

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

(e) $g_n(k + 1) = g_n(k)$ if $r_n(k) < n$.

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

\[
\begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c}
  k & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & \cdots \\
g_n(k) & -1 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 3 & \cdots \\
r_n(k) & 3 & 1 & 2 & 3 & 1 & 2 & 3 & 1 & 2 & 3 & 1 & 2 & \cdots \\
\end{array}
\]

Table 2.1 Simple illustration of the index functions for $n = 3$. 

3. A ML(n)BiCG Algorithm. Parallel to the derivation of BiCGStab from BiCG, ML(n)BiCGStab was derived in [35] 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 [35] and summarize some properties of it.

3.1. The Algorithm. Consider the solution of the linear system

\[ Ax = b \]

where \( A \in \mathbb{C}^{N \times N} \) and \( b \in \mathbb{C}^N \). Throughout the paper we do not assume the matrix \( A \) is nonsingular except where specified.

Let be given \( n \) vectors \( q_1, \ldots, q_n \in \mathbb{C}^N \), which we call left starting vectors or shadow vectors. Set

\[ p_k = (A^H)^{g_n(k)} q_{r_n(k)} \]

for \( k = 1, 2, \ldots \). The following algorithm for solving (3.1) is from [35].

**Algorithm 3.1. ML(n)BiCG**

1. Choose an initial guess \( \tilde{x}_0 \) and \( n \) vectors \( q_1, q_2, \ldots, q_n \).
2. Compute \( \tilde{r}_0 = b - Ax_0 \) and set \( p_1 = q_1 \), \( \tilde{g}_0 = \tilde{r}_0 \).
3. For \( k = 1, 2, \ldots \), until convergence:
   4. \( \alpha_k = p_k^H \tilde{r}_{k-1} / p_k^H A \tilde{g}_{k-1} \);
   5. \( \tilde{x}_k = \tilde{x}_{k-1} + \alpha_k \tilde{g}_{k-1} \);
   6. \( \tilde{r}_k = \tilde{r}_{k-1} - \alpha_k A \tilde{g}_{k-1} \);
   7. For \( s = \max(k - n, 0), \ldots, k - 1 \)
      8. \( \beta_{s(k)} = -p_{s+1}^H A \left( \tilde{r}_k + \sum_{t=\max(k-n,0)}^{s-1} \beta_{t(k)} \tilde{g}_t \right) / p_{s+1}^H A \tilde{g}_s \);
   9. \( \text{End} \)
   10. \( \tilde{g}_k = \tilde{r}_k + \sum_{s=\max(k-n,0)}^{k-1} \beta_{s(k)} \tilde{g}_s \);
   11. Compute \( p_{k+1} \) according to (3.2)
   12. \( \text{End} \)

The ML(n)BiCG algorithm 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 \( q_1, q_2, \ldots, q_n \):

\[ B_k := \text{span} \{q_1, q_2, \ldots, q_n\} \]

\[ = \text{span} \{p_1, p_2, \ldots, p_k\} \]

\[ = \sum_{i=1}^{r_n(k)} K_{g_n(k)+1} (A^H, q_i) + \sum_{i=r_n(k)+1}^{n} K_{g_n(k)} (A^H, q_i) \]

where \( Q := [q_1, q_2, \ldots, q_n] \) and

\[ K_k(M, v) := \text{span} \{v, Mv, \ldots, M^{k-1}v\} \]

Algorithm 3.1 consists of exact mathematical formulas for \( \alpha_k, \beta_{s(k)} \), \( \tilde{x}_k, \tilde{r}_k \) and \( \tilde{g}_k \) obtained in §3 of [35]. Repeated operations should be removed in order to make the algorithm computationally efficient. Moreover, even though the algorithm has not been tested, it is believed to be numerically instable because of Line 11 in which the left starting vectors are repeatedly multiplied by \( A^H \), a type of operation which is highly sensitive to round-off errors. The algorithm has been introduced only for the purpose of developing ML(n)BiCGStab algorithms.
for $M \in \mathbb{C}^{N \times N}$, $\nu \in \mathbb{C}^N$ and $t \in \mathbb{N}$. Moreover, in ML(n)BiCG, the basis used for $\mathcal{B}_k$ is not chosen to be bi-orthogonal, but simply the set $\{p_1, p_2, \ldots, p_k\}$. Therefore, the ML(n)BiCG algorithm can be viewed as a generalization of one-sided Lanczos algorithm (see [8, 16]). 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 $A^H$ will be absorbed by the residuals and direction vectors of ML(n)BiCGStab. For constructing the right-hand side basis consisting of residuals $r_k$, we used recurrences that generalize the coupled two-term recurrences of BiCG, that is, direction vectors $g_k$ are also constructed.

3.2. Properties. Let $\nu$ be the degree of the minimal polynomial $p_{\min}(\lambda; A, \tilde{r}_0)$ of $\tilde{r}_0$ with respect to $A$ (that is, the unique monic polynomial $p(\lambda)$ of minimum degree such that $p(A)\tilde{r}_0 = 0$) and let

$$\check{S}_\nu = [p_1, p_2, \ldots, p_\nu]^H A [\tilde{r}_0, A\tilde{r}_0, \ldots, A^{\nu-1}\tilde{r}_0]$$

and

$$\check{W}_\nu = [p_1, p_2, \ldots, p_\nu]^H [\tilde{r}_0, A\tilde{r}_0, \ldots, A^{\nu-1}\tilde{r}_0].$$

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

**Proposition 3.2.** In infinite precision arithmetic, if $\prod_{i=1}^\nu \text{det}(\check{S}_i) \neq 0$, then Algorithm 3.1 does not break down by zero division for $k = 1, 2, \ldots, \nu$, and $x_\nu$ is the exact solution of (3.1). Moreover, the computed quantities satisfy

(a) $\tilde{x}_k \in \tilde{x}_0 + K_k(A, \tilde{r}_0)$ and $\tilde{r}_k = b - A\tilde{x}_k \in \tilde{r}_0 + A K_k(A, \tilde{r}_0)$ for $1 \leq k \leq \nu$.
(b) span$\{\tilde{r}_0, \tilde{r}_1, \ldots, \tilde{r}_{\nu-1}\} = K_\nu(A, \tilde{r}_0)$ for $1 \leq k \leq \nu$.
(c) span$\{A\tilde{r}_0, A\tilde{r}_1, \ldots, A\tilde{r}_{\nu-1}\} = K_\nu(A, \tilde{r}_0)$.
(d) $\tilde{r}_k \perp \mathcal{B}_k$ and $\tilde{r}_k \not\perp p_{k+1}$ for $0 \leq k \leq \nu - 1$.
(e) span$\{\tilde{g}_0, \tilde{g}_1, \ldots, \tilde{g}_{\nu-1}\} = K_\nu(A, \tilde{r}_0)$ for $1 \leq k \leq \nu$.
(f) span$\{A\tilde{g}_0, A\tilde{g}_1, \ldots, A\tilde{g}_{\nu-1}\} = K_\nu(A, \tilde{r}_0)$.
(g) $A\tilde{g}_k \perp \mathcal{B}_k$ and $A\tilde{g}_k \not\perp p_{k+1}$ for $0 \leq k \leq \nu - 1$.

Because of Proposition 3.2(a) and (d), ML(n)BiCG is an oblique projection Krylov subspace method[17].

**Remarks:**

(i) The matrices $\check{S}_t$ and $\check{W}_t$ have already appeared in [11, 12] where they were called moment matrices. Proposition 3.2 can be regarded as a generalization of Theorem 2 in [12] 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 $p_k^H A\tilde{g}_{k-1} = 0$. The condition $\prod_{i=1}^{\nu} \text{det}(\check{W}_i) \neq 0$ guarantees that the underlying Lanczos process works without breakdown, and the condition $\prod_{i=1}^{\nu} \text{det}(\check{S}_i) \neq 0$ ensures that the LU factorizations exist.
(iii) $\text{det}(\check{S}_\nu) \neq 0$ implies that $p_{\min}(0; A, \tilde{r}_0) \neq 0$ which, in turn, implies that (3.1) is consistent and has a solution lying in $\tilde{x}_0 + K_\nu(A, \tilde{r}_0)$.

We say that $u \perp v$ if $u^H v = 0$. 
The derivation of ML(n)BiCGStab will require the following result.

**Corollary 3.3.** Let \( s \in \mathcal{N} \) and

\[
\psi_{g_0(s)}(\lambda) = c_{g_0(s)}\lambda^{g_0(s)} + c_{g_0(s)-1}\lambda^{g_0(s)-1} + \cdots + c_0
\]

be any polynomial of exact degree \( g_0(s) \). Then, under the assumptions of Proposition 3.2,

\[
p_s^H \tilde{r}_k = \frac{1}{c_{g_0(s)}} q_{r_0(s)}^H \psi_{g_0(s)}(A) \tilde{r}_k \quad \text{and} \quad p_s^H A \tilde{g}_k = \frac{1}{c_{g_0(s)}} q_{r_0(s)}^H A \psi_{g_0(s)}(A) \tilde{g}_k
\]

if \( 0 \leq k \leq \nu - 1 \) and \( s \leq k + n \).

**Proof.** It is easy to verify that

\[
p_s = \frac{1}{c_{g_0(s)}} \bar{\psi}_{g_0(s)}(A^H) q_{r_0(s)} \in \mathcal{B}_k
\]

by Proposition 2.1(a) and (3.2), where the overbar denotes complex conjugation. The corollary then follows from Proposition 3.2(d) and (g).

Corollary 3.3 essentially says that adding to \( p_s \) a vector from \( \mathcal{B}_k \) does not change the inner products \( p_s^H \tilde{r}_k \) and \( p_s^H A \tilde{g}_k \).

Examples exist where the condition \( \prod_{i=1}^n \det(\tilde{W}_i) \det(\tilde{S}_i) \neq 0 \) in Proposition 3.2 holds, as shown below.

**Lemma 3.4.** Consider the case where \( n = 1, \tilde{r}_0 \in \mathcal{R}^N, \tilde{r}_0 \neq 0 \) and \( A \in \mathcal{R}^{N \times N} \) is nonsingular. If \( q_1 \in \mathcal{R}^N \) is a random vector with independent and identically distributed elements from \( N(0,1) \), the normal distribution with mean 0 and variance 1, then \( \text{Prob} \left( \prod_{i=1}^n \det(\tilde{W}_i) \det(\tilde{S}_i) = 0 \right) = 0 \).

**Proof.** Since \( p_k = A^{q_0(k)} q_{r_0(k)} = A^{k-1} q_1 \) when \( n = 1 \), both \( \tilde{S}_\nu \) and \( \tilde{W}_\nu \) are Hankel matrices

\[
\tilde{S}_\nu = \begin{bmatrix}
\hat{s}_1 & \hat{s}_2 & \cdots & \hat{s}_\nu \\
\hat{s}_2 & \hat{s}_3 & \cdots & \hat{s}_{\nu+1} \\
\cdots & \cdots & \cdots & \cdots \\
\hat{s}_\nu & \hat{s}_{\nu+1} & \cdots & \hat{s}_{2\nu-1}
\end{bmatrix}, \quad \tilde{W}_\nu = \begin{bmatrix}
\hat{w}_1 & \hat{w}_2 & \cdots & \hat{w}_\nu \\
\hat{w}_2 & \hat{w}_3 & \cdots & \hat{w}_{\nu+1} \\
\cdots & \cdots & \cdots & \cdots \\
\hat{w}_\nu & \hat{w}_{\nu+1} & \cdots & \hat{w}_{2\nu-1}
\end{bmatrix}
\]

where \( \hat{s}_t = q_1^T A^t \tilde{r}_0 \) and \( \hat{w}_t = q_1^T A^{t-1} \tilde{r}_0 \) for \( t = 1, 2, \cdots, 2\nu - 1 \).

We first prove

\[
\text{Prob} \left( \det(\tilde{W}_1) = 0 \right) = 0
\]

for any fixed \( l \) with \( 1 \leq l \leq \nu \). It is trivial that (3.3) holds when \( l = 1 \) and we therefore assume \( l \geq 2 \) in the following discussion.

By assumption, \( \nu \) is the degree of the minimal polynomial of \( \tilde{r}_0 \) with respect to \( A \). This implies that \( \mathcal{K} := \text{span}\{A^t \tilde{r}_0 \mid t \in \mathcal{N}_0\} \) is a \( \nu \)-dimensional space with \( \{\tilde{r}_0, A \tilde{r}_0, \cdots, A^{\nu-1} \tilde{r}_0\} \) as a basis. Since \( A \) is nonsingular, \( \{A^{l-1} \tilde{r}_0, A^l \tilde{r}_0, \cdots, A^{l+\nu-2} \tilde{r}_0\} \) is another basis of \( \mathcal{K} \).

---

4In Lemma 3.4 and Theorem 3.5, \( \tilde{r}_0 \) can be any non-zero vector in \( \mathcal{R}^N \), not necessary to be a residual vector like \( \tilde{r}_0 = b - A \tilde{x}_0 \).
Perform an orthogonal factorization of the $N \times \nu$ matrix

$$[A^{l-1}R_0, A^{l\nu}R_0, \ldots, A^{l+\nu-2}R_0] = QR$$

where $Q \in \mathcal{R}^{N \times N}$ is orthogonal and $R \in \mathcal{R}^{N \times \nu}$ is upper triangular with positive main diagonal elements $r_{11}, r_{22}, \ldots, r_{\nu \nu}$. Clearly, the first $\nu$ columns of $Q$ form a basis of $\mathcal{K}$ and the last $N - \nu$ columns belong to $\mathcal{K}^\perp$.

Write

$$A^{l-2}R_0 = \xi_1 A^{l-1}R_0 + \xi_2 A^lR_0 + \cdots + \xi_{\nu} A^{l+\nu-2}R_0$$

for some scalars $\xi_1, \xi_2, \ldots, \xi_{\nu} \in \mathcal{R}$, where $\xi = [\xi_1, \xi_2, \ldots, \xi_{\nu}]^T \in \mathcal{R}^\nu$ and $\eta = [\eta_1, \eta_2, \ldots, \eta_N]^T = R\xi \in \mathcal{R}^N$. Since $A$ is nonsingular and $\{R_0, AR_0, \ldots, A^{l-1}R_0\}$ linearly independent, we have $\xi_{\nu} \neq 0$ and hence $\eta_{\nu} = r_{\nu \nu} \xi_{\nu} \neq 0$. Let $\theta = [\theta_1, \theta_2, \ldots, \theta_N]^T = Q^T \eta$. Then $\theta$ is a random vector with iid elements from $N(0, 1)|2]$. We now express $\det(W_l)$ in terms of the elements of $\theta$. Let us write

$$\hat{w}_1, \hat{w}_2, \ldots, \hat{w}_{l-1}, \hat{w}_l, \ldots, \hat{w}_{2l-1}$$

as

$$\begin{align*}
\hat{w}_1 & = \begin{cases}
\text{a linear combination of } \theta_1, \theta_2, \ldots, \theta_\nu \text{ if } 1 \leq t \leq l - 2,
\eta_1 \theta_1 + \eta_2 \theta_2 + \cdots + \eta_\nu \theta_\nu \text{ with } \eta_\nu \neq 0 \text{ if } t = l - 1,
\theta_1, \theta_{l-1}, \theta_{l+1}, \ldots, \theta_{2l-1} \text{ if } l \leq t \leq 2l - 1 - \nu,
\end{cases}
\end{align*}$$

with $r_{t-1-t+l-\nu} \neq 0$ if $l \leq t \leq 2l - 1$.

This shows that none of the random variables $\theta_{\nu+1}, \theta_{\nu+2}, \ldots, \theta_N$ is involved in any of the $\hat{w}_i$’s. In more detail, when $l < \nu$, $\hat{w}_t = \hat{w}_t(\theta_1, \theta_2, \ldots, \theta_\nu)$ if $1 \leq t \leq l - 1$ and $\hat{w}_t = \hat{w}_t(\theta_1, \theta_2, \ldots, \theta_{\nu-1})$ if $l \leq t \leq 2l - 1$; when $l = \nu$, $\hat{w}_t = \hat{w}_t(\theta_1, \theta_2, \ldots, \theta_{\nu})$ if $1 \leq t \leq \nu - 1$ or $l = 2\nu - 1$ and $\hat{w}_t = \hat{w}_t(\theta_1, \theta_2, \ldots, \theta_{\nu-1})$ if $\nu \leq t < 2\nu - 1$.

We now expand $\det(W_l)$ by minors down its last column and write it into a polynomial in $\theta_\nu$. This yields

$$\begin{align*}
& (-1)^{\frac{1}{2}l(l+1)+1} \det(W_l) \\
& = \hat{w}_{2l-1} \hat{w}_{l-1}^{-1} + \cdots \\
& = \begin{cases}
\left(\sum_{t=1}^{\nu-1} r_{s+t} \theta_s \right) \eta_{l-1}^{-1} \theta_{l-1}^{-1} + c_1 \eta_2 \theta_{\nu-1}^{-1} + \cdots + c_\nu \theta_{\nu-1}^{-1} + c_0 & \text{if } 2 \leq l < \nu, \\
\sum_{t=1}^{\nu} r_{\nu+t} \eta_{\nu-t} \theta_{\nu-t}^{-1} \theta_0 + d_0 & \text{if } l = \nu,
\end{cases}
\end{align*}$$

where the coefficients $c_0, \ldots, c_\nu, d_0$ are polynomials in $\theta_1, \theta_2, \ldots, \theta_{\nu-1}$. Now (3.3) follows from the facts that $r_{\nu} \neq 0, r_{\nu \nu} \neq 0, \eta_{\nu} \neq 0$ and $\theta_1, \theta_2, \ldots, \theta_{\nu-1}$ are independent random variables.

Note that $\nu$ is also the degree of the minimal polynomial of $AR_0$ with respect to $A$ when $A$ is nonsingular. With $R_0$ replaced by $AR_0$ in (3.3), we then have

$$\text{Prob} \left( \det(S_l) = 0 \right) = 0$$
for any \( l \) with \( 1 \leq l \leq \nu \).

Now, (3.3) and (3.4) together imply that

\[
Prob \left( \prod_{t=1}^\nu \det (\hat{W}_t) \det (\hat{S}_t) = 0 \right) 
\leq \sum_{t=1}^\nu Prob \left( \det (\hat{W}_t) = 0 \right) + \sum_{t=1}^\nu Prob \left( \det (\hat{S}_t) = 0 \right) = 0
\]

and the lemma is proved. \( \blacksquare \)

The \( A \) in Lemma 3.4 is assumed nonsingular. For a general \( A \), we have

**Theorem 3.5.** Consider the case where \( n = 1, \tilde{r}_0 \in \mathbb{R}^N, \tilde{r}_0 \neq 0 \) and \( A \in \mathbb{R}^{N \times N} \). If \( q_1 \in \mathbb{R}^N \) is a random vector with independent and identically distributed elements from \( N(0,1) \), then \( Prob \left( \prod_{t=1}^\nu \det (\hat{W}_t) \det (\hat{S}_t) = 0 \right) = 0 \) if and only if \( p_{\min}(0; A, \tilde{r}_0) \neq 0 \).

**Proof.** If \( p_{\min}(0; A, \tilde{r}_0) = 0 \), then \( A^t \tilde{r}_0 \) is a linear combination of \( A \tilde{r}_0, \ldots, A^{\nu-1} \tilde{r}_0 \) or \( A^t \tilde{r}_0 = 0 \) in the case when \( \nu = 1 \). Hence det(\( \hat{S}_\nu \)) = 0 no matter what \( q_1 \) is and therefore \( Prob \left( \prod_{t=1}^\nu \det (\hat{W}_t) \det (\hat{S}_t) = 0 \right) = 1 \).

We now suppose \( p_{\min}(0; A, \tilde{r}_0) \neq 0 \). By the real version of the Schur’s unitary triangularization theorem (see, for instance, [10]), \( A \) can be decomposed as

\[
A = Q^T \begin{bmatrix} B_{11} & B_{12} \\ 0 & B_{22} \end{bmatrix} Q \equiv Q^T B Q
\]

where \( Q \in \mathbb{R}^{N \times N} \) is orthogonal, \( B_{11} \in \mathbb{R}^{N_1 \times N_1} \) nonsingular and \( B_{22} \in \mathbb{R}^{N_2 \times N_2} \) strictly upper triangular (namely, an upper triangular matrix with its main diagonal elements zero). Let \( \tilde{r}_0 = Q \tilde{r}_0 = Q^{\top} [ \tilde{r}_{01}, \tilde{r}_{02} ]^T \) where \( \tilde{r}_{01} \in \mathbb{R}^{N_1} \) and \( \tilde{r}_{02} \in \mathbb{R}^{N_2} \). Then \( p_{\min}(B; A, \tilde{r}_0) \tilde{r}_0 = Q p_{\min}(A; A, \tilde{r}_0) \tilde{r}_0 = 0 \). Note that

\[
(3.5) \quad B^k = \begin{bmatrix} B_{11}^k & * \\ 0 & B_{22}^k \end{bmatrix}
\]

for \( k \in \mathcal{N} \), we have

\[
p_{\min}(B; A, \tilde{r}_0) = \begin{bmatrix} p_{\min}(B_{11}; A, \tilde{r}_0) \\ 0 \end{bmatrix} = \begin{bmatrix} p_{\min}(B_{22}; A, \tilde{r}_0) \end{bmatrix}.
\]

Thus, \( p_{\min}(B; A, \tilde{r}_0) \tilde{r}_0 = 0 \) implies that \( p_{\min}(B_{22}; A, \tilde{r}_0) \tilde{r}_{02} = 0 \). If we write \( p_{\min}(\lambda; A, \tilde{r}_0) = \sum_{t=0}^{\nu} c_t \lambda^t \) with \( c_0 \neq 0 \), then \( \sum_{t=1}^{\nu} c_t B_{22}^t \) is a strictly upper triangular matrix. Thus, \( p_{\min}(B_{22}; A, \tilde{r}_0) = (\sum_{t=1}^{\nu} c_t B_{22}^t) + c_0 I \) is an upper triangular matrix whose main diagonal elements are \( c_0 \). So, \( p_{\min}(B_{22}; A, \tilde{r}_0) \tilde{r}_{02} = 0 \) yields \( \tilde{r}_{02} = 0 \). Since \( \tilde{r}_0 \neq 0 \) due to \( \tilde{r}_0 \neq 0 \) by assumption, \( \tilde{r}_{02} \neq \tilde{r}_0 \). In other words, \( N_2 < N \) or \( B_{11} \) is not a null matrix.

Now that \( \tilde{r}_{02} = 0 \), (3.5) implies that

\[
(3.6) \quad B^k \tilde{r}_0 = \begin{bmatrix} B_{11}^k \tilde{r}_{01} \\ 0 \end{bmatrix}
\]

for \( k \in \mathcal{N} \). Therefore, \( p(B) \tilde{r}_0 = [p(B_{11}) \tilde{r}_{01}]^T, 0^T]^T \) for any polynomial \( p(\lambda) \). Thus, the minimal polynomial of \( \tilde{r}_0 \) with respect to \( B \) is equal to the minimal polynomial of
$\mathbf{r}_{01}$ with respect to $\mathbf{B}_{11}$. This implies that, $\nu$, the degree of the minimal polynomial of $\mathbf{r}_0$ with respect to $\mathbf{A}$, is also the degree of the minimal polynomial of $\mathbf{r}_{01}$ with respect to $\mathbf{B}_{11}$.

We now set $\theta = \mathbf{Q}\mathbf{q}_1 = [\theta_1^T, \theta_2^T]^T$ where $\theta_1 \in \mathcal{R}^{N_1}$ and $\theta_2 \in \mathcal{R}^{N_2}$. Since $\mathbf{q}_1$ is random with iid elements from $N(0, 1)$, so is $\theta$. By (3.6),

$$\mathbf{q}_1^T \mathbf{A}^{k}\mathbf{r}_0 = \theta^T \mathbf{B}^{k}\mathbf{r}_0 = \theta_1^T \mathbf{B}_{11}\mathbf{r}_{01} \quad \text{and} \quad \mathbf{q}_1^T \mathbf{r}_0 = \theta_1^T \mathbf{r}_{01}$$

where $k \in \mathcal{N}$. Thus

$$\tilde{\mathbf{S}}_\nu(\mathbf{A}, \mathbf{r}_0, \mathbf{q}_1) = \tilde{\mathbf{S}}_\nu(\mathbf{B}_{11}, \mathbf{r}_{01}, \theta_1) \quad \text{and} \quad \tilde{\mathbf{W}}_\nu(\mathbf{A}, \mathbf{r}_0, \mathbf{q}_1) = \tilde{\mathbf{W}}_\nu(\mathbf{B}_{11}, \mathbf{r}_{01}, \theta_1).$$

Now, the desired probability follows from Lemma 3.4 because $\mathbf{B}_{11}$ is nonsingular, $\theta_1$ is iid $N(0, 1)$ random and $\nu$ is the degree of the minimal polynomial of $\mathbf{r}_{01}$ with respect to $\mathbf{B}_{11}$.

Extension of the theorem to the general case should be possible, namely, $n \geq 1, \mathbf{A} \in \mathcal{C}^{N \times N}, \mathbf{r}_0 \in \mathcal{C}^N$ and $[\mathbf{q}_1, \ldots, \mathbf{q}_n]$ is a Gaussian matrix. We remark that, when $\mathbf{A}$ is non-defective, the general case has been proved in the proof of Theorem 3 of [27]. The proof was based on the observation that, if a polynomial $p(\lambda_1, \ldots, \lambda_l) \neq 0$, then $\mathcal{P} \{p(\lambda_1, \ldots, \lambda_l) = 0 \}$ when $\lambda_1, \ldots, \lambda_l$ are randomly chosen.

Remark: $p_{\min}(0; \mathbf{A}, \mathbf{r}_0) \neq 0$ if and only if the affine space $\tilde{\mathbf{x}}_0 + \text{span}\{\mathbf{A}^T\mathbf{r}_0 | t \in \mathcal{N}_0\}$ contains a solution to (3.1).

The following corollary then follows from Proposition 3.2 and Theorem 3.5.

**Corollary 3.6.** In the case where $n = 1$, (3.1) is a real system and $\mathbf{q}_1 \in \mathcal{R}^N$ is a random vector with iid elements from $N(0, 1)$, Algorithm 3.1 almost surely works without breakdown by zero division to find a solution from the affine space $\tilde{\mathbf{x}}_0 + \text{span}\{\mathbf{A}^T\mathbf{r}_0 | t \in \mathcal{N}_0\}$ provided that $\tilde{\mathbf{x}}_0 \in \mathcal{R}^N$ is chosen such that the affine space contains a solution to (3.1).

Remarks:

(i) The initial guess $\tilde{\mathbf{x}}_0$ in Corollary 3.6 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 $\tilde{\mathbf{x}}_0$ for the new system.

(ii) If we pick $\tilde{\mathbf{x}}_0 \in \mathcal{R}^N$ randomly and set $\mathbf{q}_1 = \mathbf{b} - \mathbf{A}\tilde{\mathbf{x}}_0$, then Algorithm 3.1 with $n = 1$, or equivalently in mathematics, the standard BiCG (see §6), almost surely solves (3.1) without breakdown by zero division for all, but a certain small class of, nonsingular $\mathbf{A} \in \mathcal{R}^{N \times N}$. For details, see [12].

**4. A ML(n)BiCGStab Algorithm.** An algorithm for the ML(n)BiCGStab method has been derived from ML(n)BiCG in [35] (Algorithm 2 without preconditioning and Algorithm 3 with preconditioning in [35]), 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 the polynomial of degree $k$ defined by (1.1). If expressed in terms of the power basis

$$(4.1) \quad \phi_k(\lambda) = c_k^{(k)} \lambda^k + \cdots + c_1^{(k)} \lambda + c_0^{(k)},$$
it is clear that \( c_k^{(k)} = \rho_1 \rho_2 \cdots \rho_k \) and \( c_0^{(k)} = 1 \). Thus,
\[
(4.2) \quad c_k^{(k)} = \rho_k c_{k-1}^{(k-1)}.
\]
In ML(\( n \))BiCGStab, we construct the following vectors: for \( k \in \mathcal{N} \),
\[
\begin{align*}
    r_k &= \phi_{g_k + (k+1)}(A) \tilde{r}_k, \quad u_k = \phi_{g_k + (k)}(A) \tilde{r}_k, \\
    g_k &= \phi_{g_k + (k+1)}(A) \tilde{g}_k, \quad d_k = \rho_{g_k + (k+1)}A \phi_{g_k + (k)}(A) \tilde{g}_k, \\
    w_k &= A g_k
\end{align*}
\]
and for \( k = 0 \), set
\[
(4.4) \quad r_0 = \tilde{r}_0 \quad \text{and} \quad g_0 = \tilde{g}_0.
\]
The vectors \( r_k \) will be the residual vectors of the approximate solutions \( x_k \) computed in the following ML(\( n \))BiCGStab algorithm.

4.2. Algorithm Derivation. The derivation parallels the one of BiCGStab from BCG. We first replace all the inner products \( p^H \tilde{r} \) and \( p^H A \tilde{g} \) in ML(\( n \))BiCG respectively by inner products of the forms \( q^H \phi(A) \tilde{r} \) and \( q^H A \phi(A) \tilde{g} \), where \( \phi \) is the polynomial (1.1). Corollary 3.3 guarantees that the inner products remain unchanged with such replacements. Then we compile recurrences for the new residuals \( r_k \) and the corresponding iterates. The overall derivation is best described and verified in stages, and depends on Proposition 2.1 and Corollary 3.3.

The derivation is complicated by the fact that the recurrences in the \( k \)th iteration in ML(\( n \))BiCG involve \( n \) terms which stretch from \( k - n \) to \( k - 1 \). Note that \( k - n \leq g_n(k)n \leq k - 1 \). 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 2.1). Therefore, our first task in the derivation 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 \( k = 1, 2, \ldots \), until convergence:
2. If \( r_n(k) = 1 \)
3. \( \alpha_k = p^H \tilde{r}_k - 1 / p^H_k A \tilde{g}_{k-1} \)
4. \( \tilde{r}_k = \tilde{r}_{k-1} - \alpha_k A \tilde{g}_{k-1} \)
5. Else
6. \( \alpha_k = p^H \tilde{r}_k - 1 / p^H_k A \tilde{g}_{k-1} \)
7. \( \tilde{r}_k = \tilde{r}_{k-1} - \alpha_k A \tilde{g}_{k-1} \)
8. End
9. If \( r_n(k) < n \)
10. For \( s = \max(k - n, 0) \), \ldots , \( g_n(k)n - 1 \)
11. \( \beta_s^{(k)} = - p^H p_{s+1} A \left( \tilde{r}_k + \sum_{t=\max(k-n,0)}^{s-1} \beta_t^{(k)} \tilde{g}_t \right) / p^H_{s+1} A \tilde{g}_s \)
12. End
13. \( \beta_{g_n(k)}(k) = - p^H p_{g_n(k)n+1} A \left( \tilde{r}_k + \sum_{t=\max(k-n,0)}^{g_n(k)n-1} \beta_t^{(k)} \tilde{g}_t \right) / p^H_{g_n(k)n+1} A \tilde{g}_{g_n(k)n} \)
14. For \( s = g_n(k)n + 1, \ldots , k - 1 \)
15. \( \beta_s^{(k)} = - p^H p_{s+1} A \left( \tilde{r}_k + \sum_{t=\max(k-n,0)}^{g_n(k)n} \beta_t^{(k)} \tilde{g}_t + \sum_{t=g_n(k)n+1}^{s-1} \beta_t^{(k)} \tilde{g}_t \right) / p^H_{s+1} A \tilde{g}_s \)
16. 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 $\mathbf{p}^H \mathbf{r}$ and $\mathbf{p}^H \mathbf{A} \tilde{g}$ by inner products of the forms $\mathbf{q}^H \phi(\mathbf{A}) \mathbf{r}$ and $\mathbf{q}^H \mathbf{A} \phi(\mathbf{A}) \tilde{g}$ respectively. That is, the factor $(\mathbf{A}^H g_n)$ that is hidden in the left basis vector $\mathbf{p}_k$ is moved to the right-hand side space and replaced by the factor $\phi_{g_n}(\mathbf{A})$. Formally, by Corollary 3.3 together with (3.2), (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 $k = 1, 2, \ldots$, until convergence:
   2. If $r_n(k) = 1$
      3. $\alpha_k = q_n^H(\phi_{g_n}(\mathbf{A}) \mathbf{r}_k - 1)/q_n^H(\alpha_k \mathbf{A} \phi_{g_n}(\mathbf{A}) \tilde{g}_k - 1)$
      4. $\phi_{g_n}(\mathbf{A}) \mathbf{r}_k = \phi_{g_n}(\mathbf{A}) \mathbf{r}_k - 1 - \alpha_k \mathbf{A} \phi_{g_n}(\mathbf{A}) \tilde{g}_k - 1$
      5. $\phi_{g_n}(\mathbf{A}) \mathbf{r}_k = (\rho_{g_n}(k+1) + 1) \phi_{g_n}(\mathbf{A}) \mathbf{r}_k$
   6. Else
      7. $\alpha_k = q_n^H(\phi_{g_n}(\mathbf{A}) \mathbf{r}_k - 1)/q_n^H(\phi_{g_n}(\mathbf{A}) \mathbf{r}_k - 1 - \alpha_k \mathbf{A} \phi_{g_n}(\mathbf{A}) \tilde{g}_k - 1)$
      8. $\phi_{g_n}(\mathbf{A}) \mathbf{r}_k = \phi_{g_n}(\mathbf{A}) \mathbf{r}_k - 1 - \alpha_k \mathbf{A} \phi_{g_n}(\mathbf{A}) \tilde{g}_k - 1$
      9. $\phi_{g_n}(\mathbf{A}) \mathbf{r}_k = \phi_{g_n}(\mathbf{A}) \mathbf{r}_k - 1 - \alpha_k \mathbf{A} \phi_{g_n}(\mathbf{A}) \mathbf{r}_k - 1$
   10. End
   11. If $r_n(k) < n$
      12. For $s = \max(k-n, 0), \ldots, g_n(k) - 1$
         13. $\beta_s(k) = -q_n^H(\phi_{g_n}(\mathbf{A}) \mathbf{r}_k + \sum_{t=\max(k-n, 0)}^{g_n(k) - 1} \beta_t(k) \phi_{g_n}(\mathbf{A}) \mathbf{r}_k + \sum_{t=g_n(k) + 1}^{g_n(k) + 1} \beta_t(k) \phi_{g_n}(\mathbf{A}) \mathbf{r}_k) / / \rho_{g_n}(k+1) q_n^H(\phi_{g_n}(\mathbf{A}) \tilde{g}_k)$
      14. End
      15. $\beta_s(g_n(k)) = q_n^H(\phi_{g_n}(\mathbf{A}) \mathbf{r}_k + \sum_{t=\max(k-n, 0)}^{g_n(k) - 1} \beta_t(k) \phi_{g_n}(\mathbf{A}) \mathbf{r}_k + \sum_{t=g_n(k) + 1}^{g_n(k) + 1} \beta_t(k) \phi_{g_n}(\mathbf{A}) \mathbf{r}_k) / / \rho_{g_n}(k+1) q_n^H(\phi_{g_n}(\mathbf{A}) \tilde{g}_k)$
      16. For $s = g_n(k) - n, \ldots, k - 1$
         17. $\beta_s(k) = -q_n^H(\phi_{g_n}(\mathbf{A}) \mathbf{r}_k + \sum_{t=\max(k-n, 0)}^{g_n(k) - 1} \beta_t(k) \phi_{g_n}(\mathbf{A}) \mathbf{r}_k + \sum_{t=g_n(k) + 1}^{g_n(k) + 1} \beta_t(k) \phi_{g_n}(\mathbf{A}) \mathbf{r}_k) / / \rho_{g_n}(k+1) q_n^H(\phi_{g_n}(\mathbf{A}) \tilde{g}_k)$
         18. End
      19. $\phi_{g_n}(\mathbf{A}) \tilde{g}_k = \phi_{g_n}(\mathbf{A}) \mathbf{r}_k + \sum_{s=g_n(k) - 1}^{g_n(k) + 1} \beta_s(k) \phi_{g_n}(\mathbf{A}) \tilde{g}_k$
By Corollary 3.3, 
\[ \sum_{s=g_n(k)+1}^{k-1} \beta_{g_n(k)+1}(A) \tilde{r}_k; \]

for \( s = g_n(k) n + 1, \ldots, k - 1 \)

By (3.2) and Proposition 2.1(a), we have

\[ \beta_{g_n(k)}(k) \]

To this end, we further transform DS#2 into the following version. This time, we

\[ p \]

term

transformation of Line 3, DS#1, into Line 3, DS#2 and (ii) the

\[ \phi \]

respectively. Line 5, DS#2, is a direct result of the definition (1.1) of

DS#1, through a multiplication by

\[ H \]

in (4.3) can be applied. Again, further explanations are given after the listing.

22. Else

23. For

24. End

25. End

26. \[ \phi_{g_n(k)+1}(A) \tilde{g}_k = \phi_{g_n(k)+1}(A) \tilde{r}_k + \phi_{g_n(k)}(k) \phi_{g_n(k)+1}(A) \tilde{g}_k; \]

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 \( \phi_{g_n(k)}(A) \), \( \phi_{g_n(k)+1}(A) \) and \( \rho_{g_n(k)+1} A \phi_{g_n(k)}(A) \)
respectively. Line 5, DS#2, is a direct result of the definition (1.1) of \( \phi \). 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 \( p_{g_n(k)+1} H A \tilde{r}_k \) on Line 13, DS#1, into the term \( q_{n+1} H \phi_{g_n(k)+1}(A) \tilde{r}_k \) on Line 15, DS#2, as follows.

(i) By Corollary 3.3,

\[ \alpha_k = \frac{p_{g_n(k)+1} H A \tilde{r}_k}{q_{n+1} H A \tilde{g}_k} = \frac{1}{\phi_{g_n(k)}(A)} q_{n+1} H A \phi_{g_n(k)}(A) \tilde{r}_k = \frac{q_{n+1} H A \phi_{g_n(k)}(A) \tilde{r}_k}{q_{n+1} H A \phi_{g_n(k)}(A) \tilde{g}_k} \]

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

(ii) By (3.2) and Proposition 2.1(a), we have

\[ A^H p_{g_n(k)+1} = (A^H)_{g_n(k)+1} q_{g_n(k)n+1} = (A^H)_{g_n(k)+1} q_{g_n(k)(n+1)+1} \]

Hence \( p_{g_n(k)+1}^H A \tilde{r}_k = p_{g_n(k)+1}^H A \tilde{r}_k \). Since \( (g_n(k) + 1)n + 1 \leq k + n \), an
application of Corollary 3.3 to \( p_{g_n(k)+1}^H A \tilde{r}_k \) thus yields

\[ p_{g_n(k)+1}^H A \tilde{r}_k = \frac{1}{\phi_{g_n(k)+1}(A)} q_{g_n(k)(n+1)+1} \phi_{g_n(k)+1}(A) \tilde{r}_k \]

The second equation above follows from (2.2). The coefficient \( 1/c_{g_n(k)+1} \) 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 $k = 1, 2, \ldots$, until convergence:
2.  If $r_n(k) = 1$
3. \[ \alpha_k = q_n^H(k)\phi_{g_n(k-1)+1}(A)\tilde{r}_{k-1}/q_n^H(k)A\phi_{g_n(k-1)+1}(A)\hat{g}_{k-1}; \]
4. \[ \phi_{g_n(k)}(A)\tilde{r}_k = \phi_{g_n(k-1)+1}(A)\tilde{r}_{k-1} - \alpha_kA\phi_{g_n(k-1)+1}(A)\hat{g}_{k-1}; \]
5. \[ \phi_{g_n(k)+1}(A)\tilde{r}_k = (\rho_{g_n(k)+1}A + I)\phi_{g_n(k)}(A)\tilde{r}_k; \]
6. Else
7. \[ \alpha_k = q_n^H(k)\phi_{g_n(k-1)}(A)\tilde{r}_{k-1}/q_n^H(k)A\phi_{g_n(k-1)}(A)\hat{g}_{k-1}; \]
8. \[ \phi_{g_n(k)}(A)\tilde{r}_k = \phi_{g_n(k-1)}(A)\tilde{r}_{k-1} - \alpha_kA\phi_{g_n(k-1)}(A)\hat{g}_{k-1}; \]
9. \[ \phi_{g_n(k)+1}(A)\tilde{r}_k = \phi_{g_n(k-1)+1}(A)\tilde{r}_{k-1} - \alpha_kA\phi_{g_n(k-1)+1}(A)\hat{g}_{k-1}; \]
10. End
11. If $r_n(k) < n$
12.  For $s = \max(k-n,0), \ldots, g_n(k)n - 1$
13. \[ \beta^s(k) = -q_n^H(s+1)\phi_{g_n(k)}(A)\tilde{r}_k + \sum_{t=\max(k-n,0)}^{s-1} \phi_{g_n(k)+1}(A)\tilde{r}_k + 1\rho_n(t)+\sum_{t=\max(k-n,0)}^{s} \phi_{g_n(k)+1}(A)\tilde{r}_k + 1\rho_n(t); \]
14. End
15. \[ \beta^r_{g_n(k)}(n) = -q_n^H\phi_{g_n(k)+1}(A)\tilde{r}_k + \sum_{t=\max(k-n,0)}^{g_n(k)n - 1} \phi_{g_n(k)+1}(A)\tilde{r}_k + 1\rho_n(t); \]
16. For $s = g_n(k)n - 1, \ldots, k - 1$
17. \[ \beta^s(k) = -q_n^H(s+1)\phi_{g_n(k)}(A)\tilde{r}_k + \sum_{t=\max(k-n,0)}^{g_n(k)n} \phi_{g_n(k)+1}(A)\tilde{r}_k + 1\rho_n(t); \]
18. End
19. \[ \phi_{g_n(k)+1}(A)\hat{g}_{k} = \phi_{g_n(k)+1}(A)\tilde{r}_k + \sum_{s=\max(k-n,0)}^{g_n(k)n} \phi_{g_n(k)+1}(A)\hat{g}_{k}; \]
20. Else
21. \[ \beta^r_{g_n(k)}(n) = -q_n^H\phi_{g_n(k)+1}(A)\tilde{r}_k + 1\rho_n(t); \]
22. For $s = g_n(k)n - 1, \ldots, k - 1$
23. \[ \beta^s(k) = -q_n^H(s+1)\phi_{g_n(k)+1}(A)\tilde{r}_k + \beta^r_{g_n(k)}(n)\rho_{g_n(k)+1}(A)\phi_{g_n(k)+1}(A)\hat{g}_{k} + \sum_{t=\max(k-n,0)}^{s-1} \phi_{g_n(k)+1}(A)\tilde{r}_k + 1\rho_n(t); \]
24. End
25. \[ \phi_{g_n(k)+1}(A)\hat{g}_{k} = \phi_{g_n(k)+1}(A)\tilde{r}_k + \beta^r_{g_n(k)}(n)\rho_{g_n(k)+1}(A)\phi_{g_n(k)+1}(A)\hat{g}_{k} + \sum_{s=\max(k-n,0)}^{k-1} \beta^s(k)\phi_{g_n(s)+1}(A)\hat{g}_{k}; \]
26. End
27. End
28. End

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

$$\max(k-n,0) \leq s, t \leq g_n(k)n - 1,$$
we have 
\[ g_n(s + 1) = g_n(k + 1) - 1 = g_n(t + 1) \]
by Proposition 2.1(b). Now that \( g_n(k) > 0 \), \( \max(k - n, 0) = k - n \) and hence \( k - n \leq t \leq g_n(k) - 1 \).

Let \( k = jn + i \) as in (2.1). Then (4.5) is
\[ (j - 1)n + i \leq t \leq (j - 1)n + n - 1 \]
which implies that \( r_n(t) < n \). Now, Proposition 2.1(d) yields \( g_n(t + 1) = g_n(t) \) and therefore we have \( g_n(s + 1) = g_n(t) \).

Now we are ready to use the residuals \( r_k \) of ML(n)BiCGStab, the corresponding direction vectors \( g_k \), and some auxiliary vectors which were defined in (4.3) and (4.4). Substituting these vectors into DS\#3 leads to the following stage.

**Derivation Stage \#4.**

1. For \( k = 1, 2, \ldots, \) until convergence:
   2. If \( r_n(k) = 1 \)
   3. \( \alpha_k = q^H_{r_n(k)} r_{k-1} / q^H_{r_n(k)} A g_{k-1} \);
   4. \( u_k = r_{k-1} - \alpha_k A g_{k-1} \);
   5. \( r_k = \rho_{g_n(k)+1} A u_k + u_k \);
   6. Else
   7. \( \alpha_k = \rho_{g_n(k)-1+1} q^H_{r_n(k)} u_{k-1} / q^H_{r_n(k)} d_{k-1} \);
   8. \( u_k = u_{k-1} - (\alpha_k / \rho_{g_n(k)-1+1}) d_{k-1} \);
   9. \( r_k = r_{k-1} - \alpha_k A g_{k-1} \);
   10. End
    11. If \( r_n(k) < n \)
    12. For \( s = \max(k - n, 0), \ldots, g_n(k) - 1 \)
    13. \( \beta^{(k)}_s = -q^H_{r_n(s+1)} (u_k + \sum_{t=\max(k-n,0)}^{s-1} \beta^{(k)}_t d_t) / q^H_{r_n(s+1)} d_s \);
    14. End
    15. \( \beta^{(k)}_{g_n(k)n} = -q^H_{r_n(s+1)} (r_k + \rho_{g_n(k)+1} \sum_{t=\max(k-n,0)}^{g_n(k)n-1} \beta^{(k)}_t A g_t) / \rho_{g_n(k)+1} q^H_{r_n(k)} A g_{g_n(k)n} \);
    16. For \( s = g_n(k)n + 1, \ldots, k - 1 \)
    17. \( \beta^{(k)}_s = -q^H_{r_n(s+1)} (r_k + \rho_{g_n(k)+1} \sum_{t=\max(k-n,0)}^{g_n(k)n} \beta^{(k)}_t A g_t + \sum_{t=g_n(k)n+1}^{s-1} \beta^{(k)}_t d_t) / q^H_{r_n(s+1)} d_s \);
    18. End
    19. \( d_k = r_k - u_k + \rho_{g_n(k)+1} \sum_{s=\max(k-n,0)}^{g_n(k)n} \beta^{(k)}_s A g_s + \sum_{s=g_n(k)n+1}^{k-1} \beta^{(k)}_s d_s \);
    20. \( g_k = r_k + \sum_{s=\max(k-n,0)}^{g_n(k)n} \beta^{(k)}_s (\rho_{g_n(k)+1} A + 1) g_s + \sum_{s=g_n(k)n+1}^{k-1} \beta^{(k)}_s g_s \);
    21. Else
    22. \( \beta^{(k)}_{g_n(k)n} = -q^H_{r_n(s+1)} r_k / \rho_{g_n(k)+1} q^H_{r_n(k)} A g_{g_n(k)n} \);
    23. For \( s = g_n(k)n + 1, \ldots, k - 1 \)
    24. \( \beta^{(k)}_s = -q^H_{r_n(s+1)} (r_k + \rho_{g_n(k)+1} \sum_{t=g_n(k)n+1}^{g_n(k)n} A g_{g_n(k)n} + \sum_{t=g_n(k)n+1}^{s-1} \beta^{(k)}_t d_t) / q^H_{r_n(s+1)} d_s \);
    25. End
    26. \( g_k = r_k + \beta^{(k)}_{g_n(k)n} (\rho_{g_n(k)+1} A + 1) g_{g_n(k)n} + \sum_{s=g_n(k)n+1}^{k-1} \beta^{(k)}_s g_s \);
    27. End
    28. End
We consider $r_k$ to be the residual of the $k$th approximate solution $x_k$. Updating relations for $x_k$ can be obtained from Lines 4, 5 and 9 respectively:

\[(4.6) \quad x_k = \begin{cases} x_{k-1} - \rho g_n(k)u_k + \alpha_k g_k, & \text{if } r_n(k) = 1 \\ x_{k-1} + \alpha_k g_k, & \text{if } r_n(k) > 1. \end{cases} \]

After adding (4.6) to DS#4 and simplifying the operations appropriately, we arrive at the following ML($n$)BiCGStab algorithm. Just like BiCGStab, the free parameter $\rho g_n(k)+1$ on Line 5, DS#4, is chosen to minimize the 2-norm of $r_k$.

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

1. Choose an initial guess $x_0$ and $n$ vectors $q_1, q_2, \cdots, q_n$.
2. Compute $r_0 = b - Ax_0$ and set $g_0 = r_0$. Compute $w_0 = Ag_0$, $c_0 = q_1^H w_0$.
3. For $k = 1, 2, \cdots$, until convergence:
   
   4. If $r_n(k) = 1$
      5. $\alpha_k = q_n(k)^H r_{k-1}/c_{k-1}$
      6. $u_k = r_{k-1} - \alpha_k w_{k-1}$
      7. $x_k = x_{k-1} + \alpha_k g_{k-1}$
      8. $\rho g_n(k)+1 = - (A u_k)^H u_k / ||A u_k||_2^2$
      9. $x_k = x_k - \rho g_n(k)+1 u_k$
      10. $r_k = \rho g_n(k)+1 A u_k + u_k$
      11. Else
          12. $\tilde{\alpha}_k = q_n(k)^H u_{k-1}/c_{k-1}$
          13. If $r_n(k) < n$
             14. $u_k = u_{k-1} - \tilde{\alpha}_k d_{k-1}$
          15. End
          16. $x_k = x_{k-1} + \rho g_n(k)+1 \tilde{\alpha}_k g_{k-1}$
          17. $r_k = r_{k-1} - \rho g_n(k)+1 \tilde{\alpha}_k w_{k-1}$
          18. End
          19. If $r_n(k) < n$
             20. $z_d = u_k$, $g_d = 0$, $z_w = 0$;
             21. For $s = k - n, \cdots, g_n(k)n - 1$ and $g_n(k) \geq 1$
                22. $\beta_s(k) = -q_{n,s+1}^H z_d/c_s$
                23. $z_d = z_d + \beta_s(k) d_s$
                24. $g_s = g_s + \beta_s(k) g_s$
                25. $z_w = z_w + \beta_s(k) w_s$
             26. End
             27. $z_w = r_k + \rho g_n(k)+1 z_w$
                28. $\tilde{\beta}_n(k)n = -q_{n,k}^H z_w / c_{n,n}$
                29. $z_w = z_w + \tilde{\beta}_n(k)n w_{g_n(k)n}$
                30. $g_s = g_s + z_w + (\tilde{\beta}_n(k)n/\rho g_n(k)+1) g_{g_n(k)n}$
                31. For $s = g_n(k)n+1, \cdots, k - 1$
                    32. $\beta_s(k) = -q_{n,s}^H z_w/c_s$
                    33. $g_s = g_s + \beta_s(k) g_s$
                    34. $z_w = z_w + \beta_s(k) d_s$
                35. End
             36. $d_k = z_w - u_k$;
Table 4.1

| Preconditioning (M⁻¹v) | 1 + \frac{1}{n} | Vector addition (u ± v) | 2 - \frac{2}{n} |
|------------------------|-----------------|------------------------|-----------------|
| Matvec: (Av)           | 1 + \frac{1}{n} | Saxpy (u + ov)          | \max(2.5n + 2.5 - \frac{2}{n}, 6) |
| dot product (uHv)      | n + 1 + \frac{2}{n} | Storage                | \mathbf{A} + \mathbf{M} + (4n + 5)N + O(n) |

37. \( c_k = q_{r_n(k+1)}^H d_k; \)
38. \( w_k = Ag_k; \)
39. Else
40. \( \beta_n^{(k)} = -q_{r_n(k)}^H r_k / c_{g_n(k)n}; \% \beta_n^{(k)} = \rho_{g_n(k)+1} / \rho_{g_n(k)} \)
41. \( z_w = r_k + \beta_n^{(k)} w_{g_n(k)n}; \)
42. \( g_k = z_w + (\beta_n^{(k)}) g_{g_n(k)n}; \)
43. For \( s = g_n(k)n + 1, \ldots, k - 1 \)
44. \( \beta_s^{(k)} = -q_{r_n(s+1)}^H z_w / c_s; \)
45. \( g_k = g_k + \beta_s^{(k)} g_s; \)
46. \( z_w = z_w + \beta_s^{(k)} d_s; \)
47. End
48. \( w_k = Ag_k; \)
49. \( c_k = q_{r_n(k+1)}^H w_k; \)
50. End
51. End

Remarks:

(i) Algorithm 4.1 does not compute the quantities \( u_k \) and \( d_k \) when \( r_n(k) = n \) (see Lines 13-15 and Lines 39-50).

(ii) if the \( u_k \) on Line 6 happens to be zero, then the \( \rho_{g_n(k)+1} \) on Line 8 and therefore the \( x_k \) and \( r_k \) on Lines 9 and 10 will not be computable. In this case, however, the \( x_k \) on Line 7 will be the exact solution to system (3.1) and Algorithm 4.1 stops there.

We now compare Algorithm 4.1 with the ML(n)BiCGStab algorithm in [35]. First, the definitions of \( r_k, u_k \) and \( g_k \) are the same in both algorithms, but \( d_k \) is defined differently. In [35], \( d_k = g_{g_n(k)}(\mathbf{A}) \mathbf{g}_k \). In exact arithmetic, however, both algorithms compute the same \( \rho_{g_n(k)+1}, r_k \) and \( x_k \). 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 [35] can be seen and removed and some arithmetics are simplified. For example, the vectors \( d_k, u_k \) are computed in every iteration in Algorithm 2 of [35]. They are now computed only when \( r_n(k) < n \). Also, the expression of \( \beta_{g_n(k)n}^{(k)} \) on 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 4.1. Since the vectors \( \{q_1, \ldots, q_n\}, \{d_{g_n(k)n-1}, d_{g_n(k)n+1}, \ldots, d_{k-1}\}, \{g_{k-n}, \ldots, g_{k-1}\} \) and \( \{w_{k-n}, \ldots, w_{g_n(k)n}, w_{k-1}\} \) 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 $4nN$.

### 4.3. Properties

We summarize the properties of Algorithm 4.1 in the following proposition. Since $r_0 = \mathbf{r}_0$ by (4.4), $\nu$ (see §3.2) is also the degree of the minimal polynomial of $r_0$ with respect to $A$.

**Proposition 4.2.** Under the assumptions of Proposition 3.2, if $\rho_{g_n(k)+1} \neq 0$ and $-1/\rho_{g_n(k)+1} \not\in \sigma(A)$ for $1 \leq k \leq \nu - 1$, where $\sigma(A)$ is the spectrum of $A$, then Algorithm 4.1 does not break down by zero division for $k = 1, 2, \cdots, \nu$, and $x_\nu$ is the exact solution of (3.1). Moreover, the computed quantities satisfy

(a) $x_k \in x_0 + K_{g_n(k)+1}(A, r_0)$ and $r_k = b - Ax_k \in r_0 + AK_{g_n(k)+1}(A, r_0)$ for $1 \leq k \leq \nu - 1$.

(b) $r_k \neq 0$ for $1 \leq k \leq \nu - 1$ and $r_\nu = 0$.

(c) $u_k \perp \text{span}\{q_1, q_2, \cdots, q_{r_n(k)}\}$ and $u_k \not\in q_{r_n(k)+1}$ for $1 \leq k \leq \nu - 1$ with $r_n(k) < n$.

(d) $d_k \perp \text{span}\{q_1, q_2, \cdots, q_{r_n(k)}\}$ and $d_k \not\in q_{r_n(k)+1}$ for $1 \leq k \leq \nu - 1$ with $r_n(k) < n$.

Proof. The divisors in Algorithm 4.1 are $c_k, \|Au_k\|_2^2$ and $\rho_{g_n(k)+1}$ respectively, where the $\rho$’s have been assumed to be nonzero. By Proposition 3.2(c), we have $\mathbf{A} \mathbf{r}_k \neq 0$ for $1 \leq k \leq \nu - 1$. Since $-1/\rho \not\in \sigma(A)$ by assumption, $\phi_{g_n(k)}(A)$ is nonsingular. Hence $Au_k = \phi_{g_n(k)}(A) A \mathbf{r}_k \neq 0$ (see (4.3) for the first equation). Therefore, $\|Au_k\|_2 \neq 0$ for $1 \leq k \leq \nu - 1$.

$c_k$ is defined respectively on Lines 37 and 49 in the algorithm. When $r_n(k) < n$, we have $c_k = q^H_{r_n(k)+1} \mathbf{d}_k$. In this case, $c_k = \rho_{g_n(k)+1} q^H_{r_n(k)+1} A \phi_{g_n(k)}(A) g_k = \rho_{g_n(k)+1} q^H_{r_n(k)+1} A \phi_{g_n(k)+1}^{\nu_{g_n(k)+1}} p^H_{r_n(k)+1} A g_k = \rho_{g_n(k)+1} q^H_{r_n(k)+1} A \phi_{g_n(k)+1}^{\nu_{g_n(k)+1}} p^H_{r_n(k)+1} A g_k = c_{g_n(k)+1} p^H_{r_n(k)+1} A \mathbf{g}_k$ by (4.3), Corollary 3.3, (4.1) and (4.2). Since the $\rho$’s are nonzero and $p^H_{r_n(k)+1} A \mathbf{g}_k \neq 0$ by Proposition 3.2(g), we have $c_{g_n(k)+1} \neq 0$ and hence $c_k \neq 0$. When $r_n(k) = n$, on the other hand, $c_k = q^H_{r_n(k)+1} w_k = q^H_{r_n(k)+1} A \phi_{g_n(k)+1}^{\nu_{g_n(k)+1}} p^H_{r_n(k)+1} A g_k = c_{g_n(k)+1} p^H_{r_n(k)+1} A \mathbf{g}_k = c_{g_n(k)+1} p^H_{r_n(k)+1} A \mathbf{g}_k \neq 0$. Therefore, in either case, we always have $c_k \neq 0$ for $1 \leq k \leq \nu - 1$. Moreover, $c_0 = q^H_0 w_0 = q^H_0 A g_0$ according to Line 2 of the algorithm. Since $p_1 = q_1$ by (3.2) and $g_0 = \mathbf{g}_0$ by (4.4), $c_0 \neq 0$ by Proposition 3.2(g).

Now that $\|Au_k\|_2 \neq 0$ and $\rho_{g_n(k)+1} \neq 0$ 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 $k = \nu$, $u_k = u_\nu = \phi_{g_n(\nu)}(A) \mathbf{r}_\nu = 0$ and $r_k = \mathbf{r}_\nu = \phi_{g_n(\nu)}(A) \mathbf{r}_\nu = 0$ due to $\mathbf{r}_\nu = 0$ by Proposition 3.2. If it happens that $\nu_{g_n(\nu)} = 1$, then the $x_\nu (= x_\nu)$ on Line 7 is the exact solution to system (3.1) because its residual $u_\nu$ is zero. So, the algorithm stops there. Otherwise, the $x_k (= x_\nu)$ on Line 16 will be exact with residual $r_\nu = 0$ and where the algorithm stops.

Part (a) follows from the definition of $r_k$ in (4.3) and Proposition 3.2(a).

Since $r_k \neq 0$ for $1 \leq k \leq \nu - 1$ by Proposition 3.2(b) and $\phi_{g_n(k)+1}(A)$ is nonsingular due to $-1/\rho \not\in \sigma(A)$, we have $r_k = \phi_{g_n(k)+1}(A) \mathbf{r}_k \neq 0$. Therefore, Part (b) holds.

For Part (c), write $k = jn + n$ with $0 \leq j \leq j$. By (4.3), (4.1) and Corollary 3.3, we have $q^H_0 \mathbf{r}_k = q^H_0 \phi_{g_n(k)+1}(A) \mathbf{r}_k = q^H_0 \phi_{g_n((j+1)n)+1}(A) \mathbf{r}_k = q^H_0 \phi_{g_n((j+1)n)+1}(A) \mathbf{r}_k$
If we write the beginning of every cycle. For example, consider that and a breakdown due to the overflow of \( c \). Algorithm 4.1 are we terminate the algorithm when \( \| r \| = 0 \). Now Part (c) follows from Proposition 3.2(d) and \( c_{\phi_n(k)+1} \neq 0 \).

For the proof of Part (d), we first note that Algorithm 4.1 does not compute \( u_k \) 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 3.3, we have \( q^H_k u_k = q^H_k \phi_{\phi_n(k)}(A) \hat{r}_k = q^H_{r_n(jn+t)} \phi_{\phi_n(jn+t)}(A) \hat{r}_k = c_{\phi_n(jn+t)}^2 p_{jn+t}^H \hat{r}_k \). Since \( p_{jn+t}^H \hat{r}_k = 0 \) by Proposition 3.2(d), \( q^H_k u_k = 0 \) for \( 1 \leq t \leq i \). Similarly, \( q^H_{k+1} u_k = c_{\phi_n(k)}^2 p_{jn+i}^H \hat{r}_k = c_{\phi_n(k)}^2 p_{k+1}^H \hat{r}_k \) (the validity of the first equation requires \( i < n \)). Because of Proposition 3.2(d) and \( c_{\phi_n(k)}^2 \neq 0, q^H_{k+1} u_k \neq 0 \). Similar to the quantity \( u_k \), Algorithm 4.1 does not compute \( d_k \) when \( r_n(k) = n \) (see Lines 40 - 49). By (4.3), \( d_k = \rho_{\phi_n(k)+1} \phi_{\phi_n(k)}(A) \hat{g}_k \) and the proof of Part (e) is parallel to that of Part (d).

The conditions of \( \rho_{\phi_n(k)+1} \neq 0 \) and \(-1/\rho_{\phi_n(k)+1} \not\in \sigma(A)\) can be easily made satisfied. For example, one can add some small random noise (e.g., \( N(0, \delta) \) with \( \delta \ll 1 \)) to \( \rho_{\phi_n(k)+1} \) after it is computed.

**Corollary 4.3.** Consider the case where \( n = 1, (3.1) \) is a real system and \( q_1 \in \mathcal{R}^N \) is a random vector with iid elements from \( N(0,1) \). If some small random number is added to \( \rho_{\phi_n(k)+1} \) after it is computed so that \( \rho_{\phi_n(k)+1} \not\neq 0 \) and \(-1/\rho_{\phi_n(k)+1} \not\in \sigma(A)\), then Algorithm 4.1 will work almost surely without breakdown by zero division to find a solution of (3.1) from the affine space \( x_0 + \text{span} \{ A^t r_0 \mid t \in \mathbb{N}_0 \} \) provided that \( x_0 \in \mathcal{R}^N \) is chosen such that the affine space contains a solution to (3.1).

Proposition 4.2 indicates that exact solution can only be found at iteration \( k = \nu \). It is possible, however, that \( \| r_k \|_2 \) can become very small for some \( k < \nu \). In practice, we terminate the algorithm when \( \| r_k \|_2 \) falls within a given tolerance.

As in the case of BiCGStab, ML(n)BiCGStab can encounter a breakdown in its implementation. ML(n)BiCGStab, besides the two types of breakdown of ML(n)BiCG, has one more type of breakdown caused by \( \rho_{\phi_n(k)+1} \). In more detail, the divisors in Algorithm 4.1 are \( c_k, \| A u_k \|_2^2 \) and \( \rho_{\phi_n(k)+1} \). If \( \| A u_k \|_2 = 0 \), then \( \rho_{\phi_n(k)+1} = \infty \) and a breakdown due to the overflow of \( \rho_{\phi_n(k)+1} \) occurs. Under the assumptions of Proposition 4.2, on the other hand, it can be shown (see the proof of the proposition) that \( c_k = c_{\phi_n(k)+1}^2 p_{k+1}^H A \hat{g}_k \), where \( c_{\phi_n(k)+1} \) is the leading coefficient of \( \phi_{\phi_n(k)+1}(A) \) (see (4.1)). So, \( c_k \) is a quantity that relates to \( \rho_{\phi_n(k)+1} \) and the ML(n)BiCG divisor \( p_{k+1}^H A \hat{g}_k \). Thus, either \( \rho_{\phi_n(k)+1} = 0 \) or \( p_{k+1}^H A \hat{g}_k = 0 \) can cause \( c_k = 0 \).

**5. A Second ML(n)BiCGStab Algorithm.** If we write \( k = jn + i \) as in (2.1), the \( r_k \) defined by (4.4) then becomes

\[
(5.1) \quad r_{jn+i} = \phi_{i+1}(A) \hat{r}_{jn+i}
\]

where \( i = 1, 2, \ldots, n \) and \( j = 0, 1, 2, \ldots \).

Starting with \( k = 1 \), let us call every \( n \) consecutive \( k \)-iterations a “cycle”, namely, iterations \( k = 1, 2, \ldots, n \) form the first cycle, iterations \( k = n + 1, n + 2, \ldots, n + n \) the second cycle and so on. Then (5.1) increases the degree of the polynomial \( \phi \) by 1 at the beginning of every cycle. For example, consider \( n = 3 \). Then (5.1) implies that

\[
\begin{align*}
\mathbf{r}_1 & = \phi_1(A) \mathbf{r}_1, & \mathbf{r}_4 & = \phi_2(A) \mathbf{r}_4, & \mathbf{r}_7 & = \phi_3(A) \mathbf{r}_7, \\
\mathbf{r}_2 & = \phi_1(A) \mathbf{r}_2, & \mathbf{r}_5 & = \phi_2(A) \mathbf{r}_5, & \mathbf{r}_8 & = \phi_3(A) \mathbf{r}_8, \\
\mathbf{r}_3 & = \phi_1(A) \mathbf{r}_3, & \mathbf{r}_6 & = \phi_2(A) \mathbf{r}_6, & \mathbf{r}_9 & = \phi_3(A) \mathbf{r}_9.
\end{align*}
\]
Iteration $k = 4$ is the first iteration of the second cycle and the degree of $\phi$ is increased from 1 to 2 there.

One can define $r_k$ by increasing the degree of $\phi$ by one anywhere within a cycle. Correspondingly, we derive the definition will lead to a different algorithm of ML$(n)$BiCGStab. As an illustration, let us increase the degree of $\phi$ at the end of every cycle and derive the algorithm associated with it.

### 5.1. Notation and Definitions

Let $\phi_k(\lambda)$ be defined as in (1.1). For $k \in \mathbb{N}$, define

\[
\begin{align*}
\mathbf{r}_k &= \phi_{g_n(k+1)}(A)\hat{\mathbf{r}}_k, \\
\mathbf{u}_k &= \phi_{g_n(k)}(A)\hat{\mathbf{r}}_k, \\
\mathbf{w}_k &= A\hat{\mathbf{g}}_k,
\end{align*}
\]

and set

\[
\begin{align*}
\mathbf{r}_0 &= \hat{\mathbf{r}}_0 & \text{and} & \quad \mathbf{g}_0 &= \hat{\mathbf{g}}_0.
\end{align*}
\]

The vector $\mathbf{r}_k$ is considered to be the residual of the approximate solution $\mathbf{x}_k$ computed. 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 $\phi$ at the end of a cycle. To see this, let $n = 3$. Then (5.2) yields

\[
\begin{align*}
\mathbf{r}_1 &= \phi_0(A)\hat{\mathbf{r}}_1, \\
\mathbf{r}_4 &= \phi_1(A)\hat{\mathbf{r}}_4, \\
\mathbf{r}_7 &= \phi_2(A)\hat{\mathbf{r}}_7, \\
\mathbf{r}_2 &= \phi_0(A)\hat{\mathbf{r}}_2, \\
\mathbf{r}_5 &= \phi_1(A)\hat{\mathbf{r}}_5, \\
\mathbf{r}_8 &= \phi_2(A)\hat{\mathbf{r}}_8, \\
\mathbf{r}_3 &= \phi_1(A)\hat{\mathbf{r}}_3, \\
\mathbf{r}_6 &= \phi_2(A)\hat{\mathbf{r}}_6, \\
\mathbf{r}_9 &= \phi_3(A)\hat{\mathbf{r}}_9.
\end{align*}
\]

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

1. For $k = 1, 2, \ldots$, until convergence:
2. \[ \alpha_k = q_{r_n(k)}^H \phi_{g_n(k)}(A) \hat{r}_{k-1}/q_{r_n(k)}^H A \phi_{g_n(k)}(A) \hat{g}_{k-1}; \]
3. If $r_n(k) < n$
4. \[ \phi_{g_n(k)}(A) \hat{r}_k = \phi_{g_n(k)}(A) \hat{r}_{k-1} - \alpha_k A \phi_{g_n(k)}(A) \hat{g}_{k-1}; \]
5. For $s = \max(k - n, 0), \ldots, g_n(k) - 1$
6. \[ \beta_s^{(k)} = -q_{r_n(s+1)}^H \left( \phi_{g_n(s+1)+1}(A) \hat{r}_k \right) \]
   \[ + \rho_{g_n(s+1)+1} \sum_{t=\max(k-n,0)}^{s-1} \beta_t^{(k)} \phi_{g_n(s+1)}(A) \hat{g}_t \]}
7. End
8. For $s = g_n(k) n, \ldots, k - 1$
9. \[ \beta_s^{(k)} = -q_{r_n(s+1)}^H \left( \phi_{g_n(s+1)+1}(A) \hat{r}_k + \sum_{t=\max(k-n,0)}^{s-1} \beta_t^{(k)} \phi_{g_n(s+1)}(A) \hat{g}_t \right) \]
   \[ + \sum_{t=g_n(k)n}^{s-1} \beta_t^{(k)} \phi_{g_n(s+1)}(A) \hat{g}_t \]}
10. End
11. \[ \phi_{g_n(k+1)}(A) \hat{g}_k = \phi_{g_n(k+1)}(A) \hat{r}_k + (\rho_{g_n(k+1)} A + I) \sum_{s=\max(k-n,0)}^{g_n(k)n-1} \beta_s^{(k)} \phi_{g_n(k+1)-1}(A) \hat{g}_s + \sum_{s=g_n(k)n}^{k-1} \beta_s^{(k)} \phi_{g_n(k+1)}(A) \hat{g}_s; \]
12. Else
13. \[ \phi_{g_n(k)}(A) \hat{r}_k = \phi_{g_n(k)}(A) \hat{r}_{k-1} - \alpha_k A \phi_{g_n(k)}(A) \hat{g}_{k-1}; \]
14. \[ \phi_{g_n(k+1)}(A) \hat{r}_k = (\rho_{g_n(k+1)} A + I) \phi_{g_n(k+1)-1}(A) \hat{r}_k; \]
15. For $s = g_n(k) n, \ldots, k - 1$
16. \[ \beta_s^{(k)} = -q_{r_n(s+1)}^H \left( \phi_{g_n(s+1)+1}(A) \hat{r}_k + \rho_{g_n(s+1)+1} \sum_{t=g_n(k)n}^{s-1} \beta_t^{(k)} \phi_{g_n(s+1)}(A) \hat{g}_t \right) / \rho_{g_n(s+1)+1} q_{r_n(s+1)}^H A \phi_{g_n(s+1)}(A) \hat{g}_s; \]
17. End
18. \[ \phi_{g_n(k+1)}(A) \hat{g}_k = \phi_{g_n(k+1)}(A) \hat{r}_k + (\rho_{g_n(k+1)} A + I) \sum_{s=g_n(k)n}^{k-1} \beta_s^{(k)} \phi_{g_n(k+1)-1}(A) \hat{g}_s; \]
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 $\phi_{g_n(k)}(A)$ and $\phi_{g_n(k+1)}(A)$ respectively. Line 14, DS#6, is a direct result of the definition (1.1) of $\phi$.

Now we use Proposition 2.1 to write DS#6 as

Derivation Stage #7.

1. For $k = 1, 2, \ldots$, until convergence:
2. \[ \alpha_k = q_{r_n(k)}^H \phi_{g_n(k)}(A) \hat{r}_{k-1}/q_{r_n(k)}^H A \phi_{g_n(k)}(A) \hat{g}_{k-1}; \]
3. If $r_n(k) < n$
4. \[ \phi_{g_n(k+1)}(A) \hat{r}_k = \phi_{g_n(k)}(A) \hat{r}_{k-1} - \alpha_k A \phi_{g_n(k)}(A) \hat{g}_{k-1}; \]
5. For $s = \max(k - n, 0), \ldots, g_n(k) - 1$
6. \[ \beta_s^{(k)} = -q_{r_n(s+1)}^H \left( \phi_{g_n(s+1)}(A) \hat{r}_k \right) \]
   \[ + \rho_{g_n(s+1)} \sum_{t=\max(k-n,0)}^{s-1} \beta_t^{(k)} \phi_{g_n(s+1)}(A) \hat{g}_t \]}
   \[ + \rho_{g_n(s+1)} q_{r_n(s+1)}^H A \phi_{g_n(s+1)}(A) \hat{g}_s; \]
\begin{align*}
\text{End} \\
\text{For } s = g_n(k)n, \ldots, k - 1 \\
\beta_s^{(k)} &= -a_{r_n(s+1)}^H(A)\bar{r}_k + \sum_{t=\max(k-n,0)}^{g_n(k)n-1} \beta_t^{(k)} \phi_{g_n(t)+1}(A)\bar{g}_t \\
&+ \sum_{t=g_n(k)n}^{s-1} \beta_t^{(k)} \phi_{g_n(t)+1}(A)\bar{g}_t / q_{r_n(s+1)}^H(A)\phi_{g_n(s+1)}(A)\bar{g}_s; \\
\text{End} \\
\phi_{g_n(k+1)}(A)\bar{g}_k &= \phi_{g_n(k+1)}(A)\bar{r}_k + (\rho_{g_n(k+1)} + I) \sum_{s=\max(k-n,0)}^{g_n(k)n-1} \beta_s^{(k)} \phi_{g_n(s+1)}(A)\bar{g}_s + \sum_{s=g_n(k)n}^{k-1} \beta_s^{(k)} \phi_{g_n(s+1)}(A)\bar{g}_s; \\
\text{Else} \\
\phi_{g_n(k)}(A)\bar{r}_k &= \phi_{g_n(k)}(A)\bar{r}_k - \alpha_k A \phi_{g_n(k)}(A)\bar{g}_k - 1; \\
\phi_{g_n(k+1)}(A)\bar{r}_k &= (\rho_{g_n(k+1)} + I) \phi_{g_n(k)}(A)\bar{r}_k; \\
\text{For } s = g_n(k)n, \ldots, k - 1 \\
\beta_s^{(k)} &= -a_{r_n(s+1)}^H(\phi_{g_n(k)}(A)\bar{r}_k + \rho_{g_n(k+1)} \sum_{t=\max(k-n,0)}^{s-1} \beta_t^{(k)} A \phi_{g_n(t)+1}(A)\bar{g}_t / q_{r_n(s+1)}^H(A)\phi_{g_n(s+1)}(A)\bar{g}_s; \\
\phi_{g_n(k+1)}(A)\bar{g}_k &= \phi_{g_n(k+1)}(A)\bar{r}_k + (\rho_{g_n(k+1)} + I) \sum_{s=\max(k-n,0)}^{k-1} \beta_s^{(k)} \phi_{g_n(s+1)}(A)\bar{g}_s; \\
\text{End} \\
\phi_{g_n(t+1)+1}(A)\bar{g}_t &= (\rho_{g_n(t+1)+1} + I) \phi_{g_n(t+1)}(A)\bar{g}_t = (\rho_{g_n(k+1)} + I) \phi_{g_n(k)}(A)\bar{g}_t. \\
\end{align*}

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

\begin{equation}
\phi_{g_n(t+1)+1}(A)\bar{g}_t = (\rho_{g_n(t+1)+1} + I) \phi_{g_n(t+1)}(A)\bar{g}_t = (\rho_{g_n(k+1)} + I) \phi_{g_n(k)}(A)\bar{g}_t.
\end{equation}

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.**

\begin{enumerate}
\item For $s = 1, 2, \ldots$, until convergence:
\item $\alpha_k = a_{r_n(k)}^H r_k - 1 / q_{r_n(k)}^H w_k - 1$;
\item If $r_n(k) < n$ \\
\begin{align*}
\text{For } s = \max(k-n,0), \ldots, g_n(k)n - 1 \\
\beta_s^{(k)} &= -a_{r_n(s+1)}^H r_k + \rho_{g_n(k)+1} \sum_{t=\max(k-n,0)}^{s-1} \beta_t^{(k)} w_t / q_{r_n(s+1)}^H w_s;
\end{align*}
\item $\text{End}$ \\
\text{For } s = g_n(k)n, \ldots, k - 1 \\
\phi_{g_n(k)}(A) = r_k + \rho_{g_n(k+1)} \sum_{s=\max(k-n,0)}^{g_n(k)n-1} \beta_s^{(k)} w_s + \sum_{s=\max(k-n,0)}^{k-1} \beta_s^{(k)} g_s + \sum_{s=g_n(k)n}^{k-1} \beta_s^{(k)} g_s;
\item Else \\
\begin{align*}
\beta_s^{(k)} &= -a_{r_n(s+1)}^H (r_k + \rho_{g_n(k+1)} \sum_{t=\max(k-n,0)}^{s-1} \beta_t^{(k)} w_t) / q_{r_n(s+1)}^H w_s;
\end{align*}
\item $\text{End}$
\end{enumerate}
to minimize the 2-norm of \( r \) with definition (5.2) 

\[
\rho \end{equation}

\[
g \]

\[
x \]

\[
\alpha \]

\[
\beta \]

\[
\gamma \]

\[
\delta \]

\[
\epsilon \]

\[
\zeta \]

\[
\eta \]

\[
\theta \]

\[
\iota \]

\[
\kappa \]

\[
\lambda \]

\[
\mu \]

\[
\nu \]

\[
\xi \]

\[
\omicron \]

\[
\pi \]

\[
\rho \]

\[
\sigma \]

\[
\tau \]

\[
\upsilon \]

\[
\phi \]

\[
\chi \]

\[
\psi \]

\[
\omega \]

\[
\Theta \]

\[
\Delta \]

\[
\Gamma \]

\[
\Lambda \]

\[
\Xi \]

\[
\Pi \]

\[
\Sigma \]

\[
\Upsilon \]

\[
\Phi \]

\[
\Psi \]

\[
\Omega \]

\[
\text{Algorithm 5.1. ML(n)BiCGStab without preconditioning associated with definition (5.2)}
\]

1. Choose an initial guess \( x_0 \) and \( n \) vectors \( q_1, q_2, \cdots, q_n \).
2. Compute \( r_0 = b - Ax_0 \) and \( g_0 = r_0, w_0 = Ag_0, c_0 = q_0^H w_0 \).
3. For \( k = 1, 2, \cdots \), until convergence:
   4. \( \alpha_k = q_k^H r_k / c_k \); 
   5. If \( r_n(k) < n \)
   6. \( x_k = x_{k-1} + \alpha_k g_{k-1} \); 
   7. \( r_k = r_{k-1} - \alpha_k w_{k-1} \); 
   8. \( z_w = r_k, g_k = 0 \); 
   9. For \( s = \max(k-n,0), \cdots, g_n(k) n - 1 \)
   10. \( \beta_k(s) = -q_k^H r_{k+1} z_w / c_s \); 
   11. \( z_w = z_w + \beta_k(s) w_s \); 
   12. \( g_k = g_k + \beta_k(s) g_s \); 
   13. End
   14. \( g_k = z_w + \beta_k(s+1) g_{k+1} \); 
   15. \( w_k = A g_k \); 
   16. For \( s = g_n(k) n, \cdots, k - 1 \)
   17. \( \beta_k(s) = -q_k^H r_{k+1} w_k / c_s \); 
   18. \( w_k = w_k + \beta_k(s) w_s \); 
   19. \( g_k = g_k + \beta_k(s) g_s \); 
   20. End
   21. Else
Table 5.1
Average cost per \((k-)\)iteration step of Algorithm 9.2 and its storage requirement.

| Preconditioning \((M^{-1}v)\) | \(1 + \frac{1}{n}\) | Vector addition \((u \pm v)\) | \(1\) |
|-------------------------------|-----------------|-------------------------|------|
| Matvec: \((Av)\)              | \(1 + \frac{1}{n}\) | Saxpy \((u + \alpha v)\) | \(2n + 2 + \frac{1}{n}\) |
| dot product \((u^Tv)\)        | \(n + 1 + \frac{2}{n}\) | Storage                  | \(A + M + (3n + 5)N + O(n)\) |

\[
x_k = x_{k-1} + \alpha_k g_{k-1};
\]
\[
u_k = r_{k-1} - \alpha_k w_{k-1};
\]
\[
\rho_{g_a(k+1)} = -(Au_k)^H u_k / \|Au_k\|^2;
\]
\[
x_k = x_k - \rho_{g_a(k+1)} u_k;
\]
\[
r_k = \rho_{g_a(k+1)} A u_k + u_k;
\]
\[
z_w = r_k, g_k = 0;
\]
\[
\text{For } s = g_a(n) n, \ldots, k - 1
\]
\[
\beta_s^{(k)} = -\frac{\alpha_{s+1}}{\alpha_s} z_w / c_s; \quad \beta_s^{(k)} = \beta_s^{(k)} \rho_{g_a(k+1)}
\]
\[
z_w = z_w + \beta_s^{(k)} w_s;
\]
\[
g_k = g_k + \beta_s^{(k)} g_s;
\]
\[
\text{End}
\]
\[
g_k = z_w + \frac{1}{\rho_{g_a(k+1)}} g_k;
\]
\[
w_k = A g_k;
\]
\[
\text{End}
\]
\[
c_k = q_{l+1}^{H} w_k;
\]
\[
\text{End}
\]

We remark that (i) the algorithm does not compute \(u_k\) when \(r_a(k) < n\). In fact, \(u_k = r_k\) when \(r_a(k) < n\) (see the remark right after (5.2)); (ii) if the \(u_k\) on Line 23 happens to be zero, then the \(x_k\) on Line 22 will be the exact solution to system (3.1) and the algorithm stops there.

The cost and storage requirement, obtained from its preconditioned version, Algorithm 9.2 in §9, are listed in Table 5.1. Compared to Algorithm 4.1, Algorithm 5.1 saves about 20% in saxpy. Since only three sets of vectors \(\{q_1, \ldots, q_n\}, \{g_{k-n}, \ldots, g_k-1\}\) and \(\{w_{k-n}, \ldots, w_{k-1}\}\) are needed in iteration \(k\), the storage is about \(3nN\) besides storing \(A\) and \(M\).

Numerical experiments have shown that Algorithm 5.1 converges faster than Algorithm 4.1 in general in terms of time. However, a disadvantage of the algorithm is that the computed relative error \(\|r_k\|_2/\|b\|_2\) easily diverges from the exact error \(\|b - Ax_k\|_2/\|b\|_2\) (see §7), especially when \(n\) is not small.

5.3. Properties. We summarize the properties about Algorithm 5.1 below. Their proofs are similar to those in Proposition 4.2. Since \(r_0 = r_0\) by (5.3), \(\nu\) is also the degree of the minimal polynomial of \(r_0\) with respect to \(A\).

**Proposition 5.2.** Under the assumptions of Proposition 3.2, if \(\rho_{g_a(k+1)} \neq 0\) and \(-1/\rho_{g_a(k+1)} \notin \sigma(A)\) for \(1 \leq k \leq \nu - 1\), where \(\sigma(A)\) is the spectrum of \(A\), then Algorithm 5.1 does not break down by zero division for \(k = 1, 2, \ldots, \nu\), and the approximate solution \(x_k\) at step \(k = \nu\) is exact to the system (3.1). Moreover, the computed quantities satisfy...
6. Relations to Some Other Methods. In this section, we discuss the relations of ML(n)BiCGStab with the FOM, BiCGStab and IDR(n).

6.1. Algorithm 4.1.

1. Relation with FOM[18]. Consider the case where \( n \geq \nu \). In this case, \( g_n(k) = 0 \) and \( r_n(k) = k \) for \( k = 1, 2, \ldots, \nu \). Hence \( p_k = q_k \) by (3.2). If we choose \( q_k = \tilde{r}_{k-1} \) in Algorithm 3.1 (it is possible since \( \tilde{r}_{k-1} \) is computed before \( q_k \) is used), then the \( \hat{x}_k \) and \( \tilde{r}_k \) computed by the algorithm satisfy

\[
\begin{align*}
\hat{x}_k &= \hat{x}_0 + \text{span}\{\tilde{r}_0, A\tilde{r}_0, \ldots, A^{k-1}\tilde{r}_0\}, \\
\tilde{r}_k &= \text{span}\{\tilde{r}_0, A\tilde{r}_0, \ldots, A^{k-1}\tilde{r}_0\}
\end{align*}
\]

for \( 1 \leq k \leq \nu \) by Proposition 3.2(a), (d). (6.1) is what the FOM approximate solution \( x_k^{FOM} \) needs to satisfy. Therefore, when \( n \geq \nu \) and with the choice \( q_k = \tilde{r}_{k-1} \), Algorithm 3.1 is mathematically equivalent to FOM.

Now, from (4.3), the \( r_k \) computed by Algorithm 4.1 satisfies

\[
r_k = \phi_{g_n(k)+1}(A) \tilde{r}_k = \phi_1(A) \tilde{r}_k = (\rho_1 A + I) \tilde{r}_k.
\]

Note that \( u_k = \phi_{g_n(k)}(A) \tilde{r}_k = \phi_0(A) \tilde{r}_k = \tilde{r}_k \). Thus, for \( 1 \leq k \leq \nu \), \( r_k \) is the factor \( \rho_1 A + I \) times the FOM residual \( u_k \) if we set \( q_1 = r_0 \) and \( q_{k+1} = u_k \) in Algorithm 4.1.\(^5\)

2. Relation with BiCGStab[29]. When \( n = 1 \), we have \( g_n(k) = k - 1 \) and \( r_n(k) = 1 \) for \( k \in \mathcal{N} \). Hence \( p_k = (A^H)^{k-1} q_1 \) by (3.2). By Proposition 3.2(a) and (d), the \( \hat{x}_k \) and \( \tilde{r}_k \) computed by Algorithm 3.1 satisfy

\[
\begin{align*}
\hat{x}_k &= \hat{x}_0 + \text{span}\{\tilde{r}_0, A\tilde{r}_0, \ldots, A^{k-1}\tilde{r}_0\}, \\
\tilde{r}_k &= \text{span}\{\tilde{r}_0, A^H q_1, \ldots, (A^H)^{k-1} q_1\}
\end{align*}
\]

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

Now, from (4.3), the \( r_k \) computed by Algorithm 4.1 satisfies

\[
r_k = \phi_{g_n(k)+1}(A) \tilde{r}_k = \phi_k(A) \tilde{r}_k
\]

\(^5\)In [35], a remark immediately following Theorem 4.1 states that, when \( n \geq \nu \) and with the choice that \( q_1 = \phi_1(A^H) r_0 \) and \( q_k = \phi_k(A^H) r_{k-1} \) for \( k \geq 2 \), the \( x_k \) and \( r_k \) 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 this remark is not correct. The author remembers that the referees of [35] were skeptical about the argument.
which is the definition of the BiCGStab residuals. Thus Algorithm 4.1 is mathematically equivalent to BiCGStab when \( n = 1 \).

3. Relation with IDR(\( n \))[28]. Write \( k = jn + i \) as in (2.1) with \( 1 \leq i \leq n, 0 \leq j \). Let \( G_0 = K(A, r_0) \) be the complete Krylov space and let \( S = \text{span}\{q_1, q_2, \ldots, q_n\}^\perp \). Define the Sonneveld spaces

\[
G_{j+1} = (\rho_{j+1}A + I)(G_j \cap S) = (\rho_{g_{n(k+1)}}A + I)(G_j \cap S)
\]

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

\[
r_{jn+i} = \phi_{j+1}(A)\tilde{r}_{jn+i} = (\rho_{j+1}A + I)\phi_j(A)\tilde{r}_{jn+i} = (\rho_{j+1}A + I)u_{jn+i}.
\]

From Proposition 4.2(d), \( u_{jn+i} \notin q_{i+1} \) if \( i < n \). Hence \( u_{jn+i} \notin G_j \cap S \) and therefore \( r_{jn+i} \notin G_{j+1} \) when \( i < n \). From this point of view, Algorithm 4.1 is not a IDR(\( n \)) algorithm.

6.2. Algorithm 5.1.

1. Relation with FOM. When \( n \geq \nu, g_n(k) = 0 \) and \( r_n(k) = k \) for \( 1 \leq k \leq \nu \) and Algorithm 3.1, with the choice \( q_k = \tilde{r}_{k-1} \), is a FOM algorithm as seen in \$6.1. Now, from (5.2), the \( r_k \) computed by Algorithm 5.1 satisfies

\[
r_k = \phi_{g_n(k+1)}(A)\tilde{r}_k = \phi_0(A)\tilde{r}_k = \tilde{r}_k.
\]

Thus Algorithm 5.1 is a FOM algorithm when we set \( q_k = r_{k-1} \).

2. Relation with BiCGStab. When \( n = 1 \), we have \( g_n(k) = k - 1 \) and \( r_n(k) = 1 \) for \( k \in N \) and Algorithm 3.1 is a BiCG algorithm. Now, from (5.2), the \( r_k \) computed by Algorithm 5.1 satisfies

\[
r_k = \phi_{g_n(k+1)}(A)\tilde{r}_k = \phi_k(A)\tilde{r}_k
\]

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

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

\[
(6.3) r_{jn+i} = \phi_{g_{n(jn+i+1)}}(A)\tilde{r}_{jn+i} = \begin{cases} \phi_j(A)\tilde{r}_{jn+i} & \text{if } 1 \leq i < n, \\ \phi_{j+1}(A)\tilde{r}_{jn+i} & \text{if } i = n. \end{cases}
\]

By (3.2) and Proposition 3.2(d), we have

\[
\begin{align*}
\phi_j(A)\tilde{r}_{jn+i} \in S & \quad \text{if } 1 \leq i < n \text{ and } 0 \leq t < j, \\
\phi_j(A)\tilde{r}_{jn+n} \in S & \quad \text{if } 0 \leq t \leq j.
\end{align*}
\]

Thus, by induction on \( t \), we have

\[
\begin{align*}
\phi_j(A)\tilde{r}_{jn+i} \in G_t \cap S & \quad \text{if } 1 \leq i < n \text{ and } 0 \leq t < j, \\
\phi_j(A)\tilde{r}_{jn+n} \in G_t \cap S & \quad \text{if } 0 \leq t \leq j.
\end{align*}
\]

Therefore, by (6.3),

\[
\begin{align*}
r_{jn+i} = \tilde{r}_i & \in G_0 \quad \text{if } 1 \leq i < n, j = 0, \\
r_{jn+i} = (\rho_jA + I)\phi_{j-1}(A)\tilde{r}_{jn+i} & \in G_i \quad \text{if } 1 \leq i < n, j = 1, \\
r_{jn+n} = (\rho_{j+1}A + I)\phi_j(A)\tilde{r}_{jn+n} & \in G_{j+1}.
\end{align*}
\]

So, the residuals in (6.3) lie in the Sonneveld spaces \( G \) and therefore Algorithm 5.1 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

\[ AM^{-1}y = b \]

where \( M \) is nonsingular, then recovering \( x \) through \( x = M^{-1}y \). The resulting algorithms, Algorithm 9.1 and Algorithm 9.2, together with their Matlab codes are presented in §9.1 and §9.2 respectively. 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 \( 1 \leq i \leq n, 0 \leq j \). Moreover, we have optimized the operations as possible as we can in the resulting preconditioned algorithms.

Since we have compared ML\((n)\)BiCGStab with some existing methods in [35], we will only concentrate on the performance of ML\((n)\)BiCGStab itself. The following test data were downloaded from Matrix Market.\(^6\)

1. \( e20r0100 \), DRIVCAV Fluid Dynamics. \( e20r0100 \) contains a \( 4241 \times 4241 \) real unsymmetric matrix \( A \) with 131,556 nonzero entries and a real right-hand side \( b \).

2. \( qc2534 \), H2PLUS Quantum Chemistry, NEP Collection. \( qc2534 \) contains a \( 2534 \times 2534 \) complex symmetric indefinite matrix with 463,360 nonzero entries, but does not provide the right-hand side \( b \). Following [21], we set \( b = A1 \) with \( 1 = [1, 1, \cdots, 1]^T \).

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

Experiments were performed in Matlab Version 7.1 on a Windows XP machine with a Pentium 4 processor. \( ILU(0) \) preconditioner (p.294, [17]) has been used in all the experiments. For \( e20r0100 \), the \( U \)-factor of the \( ILU(0) \) decomposition of \( A \) has some zeros along its main diagonal. In that experiment, we replaced those zeros with 1 so that the \( U \)-factor was invertible.

In all the experiments, initial guess \( x_0 = 0 \) and stopping criterion is

\[ \|r_k\|_2/\|b\|_2 < 10^{-7} \]

where \( r_k \) is the computed residual. Except where specified, auxiliary vectors \( Q \equiv [q_1, q_2, \cdots, q_n] \) are chosen to be \( Q = [r_0, randn(N, n-1)] \) for \( e20r0100 \) and \( utm5940 \) and \( Q = [r_0, randn(N, n-1) + sqrt(-1) * randn(N, n-1)] \) 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 iterations that a ML\((n)\)BiCGStab algorithm takes to converge.

(c) \( T_{iter}(n) := T_{conv}(n)/I_{conv}(n) \) is the time per iteration of a ML\((n)\)BiCGStab algorithm.

(d) \( E(n) := \|b - Ax\|_2/\|b\|_2 \) is the true relative error of \( x \) where \( x \) is the computed solution output by a ML\((n)\)BiCGStab algorithm when it converges.

\(^6\)http://math.nist.gov/MatrixMarket/data/
7.1. Stability. We plot the graphs of $I_{\text{conv}}(n)$ in Figures 7.1(a), 7.2(a) and 7.3(a). For e20r0100 and qc2534, $I_{\text{conv}}(n)$ decreases as $n$ increases. However, the $I_{\text{conv}}(n)$ for utm5940 behaves very irregularly due to some of the $\rho$’s are too small. Recall that ML(n)BiCGStab performs $1 + 1/n$ matrix-vector multiplications (MVs) per iteration on average. In terms of the number of MVs, both Algorithms 9.1 and 9.2 are considerably faster than BiCGStab. BiCGStab required 455 MVs to converge.

The graphs of $E(n)$ are plotted in Figures 7.1(b), 7.2(b) and 7.3(b). It can be seen that the computed $r_k$ in Algorithm 9.2 easily diverges from its exact counterpart $b - A x_k$. This divergence becomes significant when $n \geq 15$ for e20r0100 and $n \geq 4$.
for utm5940. By contrast, the computed relative errors $\frac{\|r_k\|_2}{\|b\|_2}$ 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. However, Algorithm 9.2 taken twice to form a predictor-corrector pair can be an efficient and stable algorithm. We remark that the issues of divergence of computed residuals and corresponding remedy techniques have been discussed in details in [15, 24, 31].

### 7.2. Choice of $n$

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

From the experiments in [35] and this paper, we have observed that ML($n$)BiCGStab behaves more and more robust as $n$ is increased. So, for an ill-conditioned problem, we would tend to suggest a large $n$ for ML($n$)BiCGStab.

ML($n$)BiCGStab minimizes $\|r_k\|_2$ once every $n$ iterations. The convergence of a well-conditioned problem is usually accelerated by the minimization steps. So, when a problem is well-conditioned, we would suggest a small $n$, say, $n \leq 3$. $n = 2$ may be a good choice since it reduces the MV cost by 25% per iteration while keeping the minimization performed with a high frequency.

One can also choose $n$ so that $T_{iter}(n)$ is minimized. ML($n$)BiCGStab mainly involves three types of operation: preconditioner system solving and matrix-vector multiplication (AM$^{-1}$v), dot product (u$^H$v), and saxpy (u + αv). The algorithm requires $1 + 1/n$ of AM$^{-1}$v per iteration on the average. For smaller values of $n$, the overall computational effort is dominated by AM$^{-1}$v, and as $n$ is increased, the computations of u$^H$v and u + αv will start to take dominant. Therefore, the graph of $T_{iter}(n)$ typically has a parabolic shape, as illustrated in Figures 7.4(a), 7.5(a), 7.6(a). As a result, the minimizer $n_{opt}$ of $T_{iter}(n)$ always exists. On the other hand, however, $n_{opt}$ does not necessarily minimize $T_{conv}(n)$. There should be no absolute connection between $n_{opt}$ and the minimization of $T_{conv}(n)$.

We also plot the graphs of $T_{conv}(n)$ in Figures 7.4(b), 7.5(b) and 7.6(b) to provide more information on how $n$ affects the performance of ML($n$)BiCGStab.
7.3. Choice of $\rho$. The standard choice for the $\rho_{j+1}$ in Algorithm 9.1 is

$$\rho_{j+1} = -(A\hat{u}_{j+1})H u_{j+1}/\|A\hat{u}_{j+1}\|_2^2. \tag{7.1}$$

This choice of $\rho_{j+1}$ minimizes the 2-norm of $r_{j+1} = \rho_{j+1}A\hat{u}_{j+1} + u_{j+1}$, but sometimes can cause instability due to that it can be very small during an implementation. A remedy as follows has been suggested in [22]:

$$\rho_{j+1} = -(A\hat{u}_{j+1})H u_{j+1}/\|A\hat{u}_{j+1}\|_2^2; \quad \omega = (A\hat{u}_{j+1})H u_{j+1}/(\|A\hat{u}_{j+1}\|_2 \|u_{j+1}\|_2);$$

$$\text{if } |\omega| < \kappa, \rho_{j+1} = \kappa \rho_{j+1}/|\omega|; \text{ end} \tag{7.2}$$

where $\kappa$ is a user-defined parameter. In Figures 7.7(a)(b), we compare the performances of Algorithm 9.1 with (7.1) and (7.2) respectively (we only plot the results
Fig. 7.6. utm5940: (a) Graph of $T_{\text{iter}}(n)$ of Algorithm 9.1 against $n$. $T_{\text{iter}}(n)$ reaches its minimum at $n = 7$. (b) Graph of $T_{\text{conv}}(n)$ of Algorithm 9.1 against $n$. $T_{\text{conv}}(n)$ reaches its minimum at $n = 6$.

of qc2534 and utm5940. The result of e20r0100 with $\kappa = 0.1$ is analogous to Figure 7.7(a)). Also, see the numerical experiments in [28] for more information about these $\rho$ choices.

Fig. 7.7. (a) qc2534: Graphs of $I_{\text{conv}}(n)$ of Algorithm 9.1 against $n$ with choices (7.1) and (7.2) for $\rho$ respectively. In this experiment, we picked $\kappa = 0.7$. $\rho$ with (7.1): x-mark, $\rho$ with (7.2): o-mark. (b) utm5940: Graphs of $I_{\text{conv}}(n)$ of Algorithm 9.1 against $n$ with choices (7.1) and (7.2) for $\rho$ respectively. In this experiment, we picked $\kappa = 0.7$. $\rho$ with (7.1): x-mark, $\rho$ with (7.2): o-mark.

7.4. Choice of $q$'s. We usually pick $Q = [q_1, q_2, \cdots, q_n]$ as

(7.3) \[ Q = [r_0, \text{randn}(N, n - 1)] \]

for a real problem and

(7.4) \[ Q = [r_0, \text{randn}(N, n - 1) + \text{sqrt}(-1) \ast \text{randn}(N, n - 1)] \]
for a complex problem. In our experiments, however, we observed a comparable performance when we chose

\[(7.5) \quad Q = [r_0, \text{sign}(\text{randn}(N, n - 1))].\]

or

\[(7.6) \quad Q = [r_0, \text{sign}(\text{randn}(N, n - 1)) + \sqrt{-1} \times \text{sign}(\text{randn}(N, n - 1))].\]

See Figures 7.8(a)(b) (we only plot the results of qc2534 and utm5940. The result of e20r0100 is analogous to Figure 7.8(a)).

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

For other choices for Q, one is referred to [28].

![Fig. 7.8. (a) qc2534: Graphs of \(I_{\text{conv}}(n)\) of Algorithm 9.1 against n with choices (7.4) and (7.6) for Q respectively. Q with (7.4): x-mark, Q with (7.6): o-mark. (b) utm5940: Graphs of \(I_{\text{conv}}(n)\) of Algorithm 9.1 against n with choices (7.3) and (7.5) for Q respectively. Q with (7.3): x-mark, Q with (7.5): o-mark.]

8. Concluding Remarks. With the help of index functions, we re-derived the ML(n)BiCGStab algorithm in [35] 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 realized that there are \(n\) ways to define the ML(n)BiCGStab residuals \(r_k\). Each of the definitions will lead to a different algorithm. We presented two definitions together with their associated algorithms, namely, (i) definition (4.3), increasing the degree of \(\phi\) at the beginning of an iteration cycle, and the associated Algorithm 4.1; (ii) definition (5.2), increasing the degree of \(\phi\) at the end of an iteration cycle, and the associated Algorithm 5.1. By comparison, Algorithm 5.1 is cheaper in storage and in computational cost, faster to converge, but its computed residuals \(r_k\) can easily diverge from the true residuals, except that \(n\) is chosen small. For other definitions of \(r_k\) that increase the degree of \(\phi\) somewhere within a cycle, we expect that the associated algorithms would lie between Algorithms 4.1 and 5.1 in terms of computational cost, storage and performance.
In this section, we present the preconditioned ML (4.3). The following algorithm is a
ML(\text{BiCGStab}) with Definition (4.3). The following algorithm is a
algorithm together with their Matlab codes.

In this paper, we did not assume that $A$ is a nonsingular matrix. When a singular
system (3.1) is solved, selecting an appropriate initial guess $x_0$ is crucial. If $x_0$ is
selected such that the affine space $x_0 + \text{span}\{A^T r_0 \mid r_0 \in \mathcal{N}_0\}$ contains a solution to the
system (3.1), ML($n$)BiCG will almost surely converge (see Corollary 3.6). Otherwise,
we shall have $p_{\text{max}}(0, A, r_0) = 0$ (see the remark before Corollary 3.6) which yields
$\det(\tilde{S}_0) = 0$ (see the remark after Proposition 3.2). In this case, $\prod_{i=1}^{n} \det(\tilde{S}_i) = 0$
and therefore there is no guarantee that the $LU$-factorizations in the construction of
ML($n$)BiCG exist (see the remark after Proposition 3.2). As a result, it is likely that
$\|\tilde{r}_k\|_2$ blows up to $\infty$. A similar remark also applies to ML($n$)BiCGStab.

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

9.1. ML($n$)BiCGStab with Definition (4.3). The following algorithm is a
preconditioned version of Algorithm 4.1.

\begin{algorithm}
\caption{ML($n$)BiCGStab with preconditioning associated with (4.3).}
\begin{enumerate}
\item Choose an initial guess $x_0$ and $n$ vectors $q_1, q_2, \cdots, q_n$.
\item Compute $r_0 = b - A x_0$ and set $g_0 = r_0$.
Compute $\bar{g}_0 = M^{-1} g_0$, $w_0 = A \bar{g}_0$, $c_0 = q_1^H w_0$ and $e_0 = q_1^H r_0$.
\item For $j = 0, 1, 2, \cdots$
\begin{enumerate}
\item $\alpha_{jn+1} = e_{(j-1)n+1}/c_{(j-1)n+1};$
\item $u_{jn+1} = r_{(j-1)n+1} - \alpha_{jn+1} w_{(j-1)n+1};$
\item $x_{jn+1} = x_{(j-1)n+1} + \alpha_{jn+1} \bar{g}_{(j-1)n+1};$
\item $\bar{u}_{jn+1} = M^{-1} u_{jn+1};$
\item $r_{jn+1} = r_{(j-1)n+1} - \rho_{j+1} u_{jn+1};$
\item For $i = 1, 2, \cdots, n - 1$
\begin{enumerate}
\item $f_{jn+i} = q_i^H u_{jn+1};$
\item If $j \geq 1$
\begin{enumerate}
\item $\beta_{(j-1)n+i} = -f_{jn+i}/c_{(j-1)n+i};$
\item If $i \leq n - 2$
\item $z_d = u_{jn+i} + \beta_{(j-1)n+i} d_{(j-1)n+i};$
\item $g_{jn+i} = g_{(j-1)n+i} + \beta_{(j-1)n+i} \bar{g}_{(j-1)n+i};$
\item $z_w = \beta_{(j-1)n+i} w_{(j-1)n+i};$
\item $\rho_{(j-1)n+i} = -q_i^H z_d/c_{(j-1)n+i+1};$
\item For $s = i + 1, \cdots, n - 2$
\begin{enumerate}
\item $z_d = z_d + \beta_{(j-1)n+i} d_{(j-1)n+s};$
\item $g_{jn+i} = g_{jn+i} + \beta_{(j-1)n+s} \bar{g}_{(j-1)n+s};$
\item $z_w = z_w + \beta_{(j-1)n+s} w_{(j-1)n+s};$
\end{enumerate}
\end{enumerate}
\end{enumerate}
\end{enumerate}
\end{enumerate}
\end{algorithm}

\footnote{For an example where the affine space contains no solution, consider $A = [0, 1; 0, 0], b = [1, 0]^T$
and select $x_0 = [1, 0]^T$. Note that this linear system is consistent.}
$\beta^{(j_{n+1})}_{(j-1)n+s+1} = -q^H_{j+2} z_d / c_{(j-1)n+s+1}$;

End

$g_{jn+i} = g_{jn+i} + \beta^{(j_{n+1})}_{(j-1)n+n-1} g_{(j-1)n+n-1}$;

$z_w = z_w + \beta^{(j_{n+1})}_{(j-1)n+n-1} w_{(j-1)n+n-1}$;

$z_w = r_{jn+i} + \rho_{j+1} z_w$;

Else

$g_{jn+i} = \beta^{(j_{n+1})}_{(j-1)n+n-1} g_{(j-1)n+n-1}$;

$z_w = r_{jn+i} + \rho_{j+1} z_w$;

End

$\tilde{\beta}^{(j_{n+1})}_{(j-1)n+n} = -q^H_{j+1} z_w / c_{(j-1)n+n}$;

$z_w = z_w + \tilde{\beta}^{(j_{n+1})}_{(j-1)n+n} w_{(j-1)n+n}$;

$g_{jn+i} = g_{jn+i} + \tilde{\beta}^{(j_{n+1})}_{(j-1)n+n} g_{(j-1)n+n}$;

Else

$\beta^{(j_{n+1})}_{(j-1)n+n} = -q^H_j z_w / c_{jn+s}$;

$g_{jn+i} = g_{jn+i} + \beta^{(j_{n+1})}_{jn+s} g_{jn+s}$;

$z_w = z_w + \beta^{(j_{n+1})}_{jn+s} d_{jn+s}$;

End

For $s = 1, \ldots, i - 1$

$\beta^{(j_{n+1})}_{jn+1} = -q^H_{j+1} z_w / c_{jn+i}$;

$g_{jn+i} = g_{jn+i} + \beta^{(j_{n+1})}_{jn+i} g_{jn+i}$;

$z_w = z_w + \beta^{(j_{n+1})}_{jn+i} d_{jn+i}$;

End

If $i < n - 1$

$d_{jn+i} = z_w - u_{jn+i}$;

$c_{jn+i} = q^H_{j+1} d_{jn+i}$;

$\alpha_{jn+i+1} = f_{jn+i} / c_{jn+i}$;

$u_{jn+i+1} = u_{jn+i} - \alpha_{jn+i+1} d_{jn+i}$;

Else

$c_{jn+i} = q^H_{j+1} (z_w - u_{jn+i})$;

$\alpha_{jn+i+1} = f_{jn+i} / c_{jn+i}$;

$u_{jn+i+1} = u_{jn+i} - \alpha_{jn+i+1} d_{jn+i}$;

End

$\tilde{g}_{jn+i} = M^{-1} g_{jn+i}$;

$w_{jn+i} = A \tilde{g}_{jn+i}$;

$x_{jn+i+1} = x_{jn+i} + \rho_{j+1} \alpha_{jn+i+1} \tilde{g}_{jn+i}$;

$\tilde{r}_{jn+i+1} = \tilde{r}_{jn+i} - \rho_{j+1} \alpha_{jn+i+1} w_{jn+i}$;

End

$c_{jn+n} = q^H_{j+1} r_{jn+n}$;

$\beta^{(j_{n+n})}_{(j-1)n+n} = -c_{jn+n} / c_{(j-1)n+n}$;

$z_w = r_{jn+n} + \beta^{(j_{n+n})}_{(j-1)n+n} w_{(j-1)n+n}$;

$g_{jn+n} = z_w + \beta^{(j_{n+n})}_{(j-1)n+n} g_{(j-1)n+n}$;

If $n \geq 2$

$\beta^{(j_{n+n})}_{jn+1} = -q^H_{j+1} z_w / c_{jn+1}$;

For $s = 1, \ldots, n - 2$

$g_{jn+n} = g_{jn+n} + \beta^{(j_{n+n})}_{jn+s} g_{jn+s}$;
\[ z_w = z_w + \beta^{(j+n)}_{jn+s} d_{jn+s}; \]
\[ \beta^{(j+n)}_{jn+s+1} = -q^H_{s+2} z_w / c_{jn+s+1}; \]

Matlab code of Algorithm 9.1

1. function \([x, err, iter, flag] = mlbimgstab(A, x, b, Q, M, max_it, tol, kappa)\)
2.
3. \(N\)-by-\(N\) matrix
4. \(M\) \(N\)-by-\(N\) preconditioner matrix
5. \(Q\) \(N\)-by-\(n\) auxiliary matrix \([q_1, \cdots, q_n]\)
6. initial guess
7. \(b\) right hand side vector
8. \(max_it\) maximum number of iterations
9. \(tol\) error tolerance.
10. \(kappa\) (real number) minimization step controller:
11. \(kappa = 0\), standard minimization
12. \(kappa > 0\), Sleijpen-van der Vorst minimization
13. solution computed
14. \(err\) error norm
15. number of iterations performed
16. \(flag = 0\), solution found to tolerance
17. \(= 1\), no convergence given \(max_it\) iterations
18. \(= -1\), breakdown.
19. \(D\) \(N \times (n - 2)\) matrix defined only when \(n > 2\)
20. \(G, Q, W\) \(N \times n\) matrices
21. \(A, M\) \(N \times N\) matrices
22. \(x, r, g, u, z, b\) \(N \times 1\) matrices
23. \(c\) \(1 \times n\) matrix.
24.
25. \(N = \text{size}(A, 2); \ n = \text{size}(Q, 2);\)
26. \(G = \text{zeros}(N, n); \ W = \text{zeros}(N, n);\) \(d, g, w\) and \(c\)
27. \(n > 2\), \(D = \text{zeros}(N, n - 2);\) \(c = \text{zeros}(1, n);\)
28. \(\text{end}\)
29. \(\text{end}\)
30. \(iter = 0; \ flag = 1; \ bnrm2 = \text{norm}(b);\)
31. \(\text{end}\)
32. \(r = b - A \times x; \ err = \text{norm}(r) / bnrm2;\)
33. \(\text{return}\)
34. \(\text{end}\)
35. \(G(:, n) = r; \ g = M \backslash r; \ W(:, n) = A \times g;\)
36. \(c(n) = Q(:, 1)' \times W(:, n);\)
37. \(\text{end}\)
38. \(\text{return}\)
39. \(\text{end}\)
39. \[ e = Q(:, 1)' * r; \]
40. for \( j = 0 : \max_\text{it} \)
41. \[ \alpha = e / c(n); \]
42. \[ x = x + \alpha * g_J; \]
43. \[ u = r - \alpha * W(:, n); \]
44. \[ \text{err} = \text{norm}(u) / \text{bcrm2}; \]
45. if \( \text{err} < \text{tol} \), \( \text{flag} = 0; \) \( \text{iter} = \text{iter} + 1; \) return, end
46. \[ g_J = M \setminus u; \]
47. \[ z = A * g_J; \]
48. \[ \rho = z' * z; \]
49. if \( \rho \) \( \approx \) 0, \( \text{flag} = -1; \) return, end
50. \[ \omega = z' * u; \]
51. if \( \omega \) \( \approx \) 0, \( \text{flag} = -1; \) return, end
52. \[ \rho = -\omega / \rho; \]
53. if \( \kappa > 0 \)
54. \[ \omega = \omega / (\text{norm}(z) * \text{norm}(u)); \]
55. \[ \text{abs}_\text{om} = \text{abs}(\omega); \]
56. if \( \text{abs}_\text{om} < \kappa \), \( \rho = \rho * \kappa / \text{abs}_\text{om}; \) end
57. end
58. \[ x = x - \rho * g_J; \]
59. \[ r = \rho * z + u; \]
60. \[ \text{err} = \text{norm}(r) / \text{bcrm2}; \]
61. \[ \text{iter} = \text{iter} + 1; \]
62. if \( \text{err} < \text{tol} \), \( \text{flag} = 0; \) return, end
63. if \( \text{iter} >= \max_\text{it} \), return, end
64. for \( i = 1 : n - 1 \)
65. \[ f = Q(:, i + 1)' * u; \]
66. if \( j >= 1 \)
67. \[ \beta = -f / c(i); \]
68. if \( i <= n - 2 \)
69. \[ D(:, i) = u + \beta * D(:, i); \]
70. \[ G(:, i) = \beta * G(:, i); \]
71. \[ W(:, i) = \beta * W(:, i); \]
72. \[ \beta = -Q(:, i + 2)' * D(:, i) / c(i + 1); \]
73. for \( s = i + 1 : n - 2 \)
74. \[ D(:, i) = D(:, i) + \beta * D(:, s); \]
75. \[ G(:, i) = G(:, i) + \beta * G(:, s); \]
76. \[ W(:, i) = W(:, i) + \beta * W(:, s); \]
77. \[ \beta = -Q(:, s + 2)' * D(:, i) / c(s + 1); \]
78. end
79. \[ G(:, i) = G(:, i) + \beta * G(:, n - 1); \]
80. \[ W(:, i) = W(:, i) + \beta * W(:, n - 1); \]
81. \[ W(:, i) = r + \rho * W(:, i); \]
82. else
83. \[ G(:, i) = \beta * G(:, n - 1); \]
84. \[ W(:, i) = r + (\rho * \beta) * W(:, n - 1); \]
85. end
86. \[ \beta = -Q(:, 1)' * W(:, i) / c(n); \]
\begin{verbatim}
88. W(:,i) = W(:,i) + beta * W(:,n);
89. G(:,i) = G(:,i) + W(:,i) + (beta/rho) * G(:,n);
90. else
91.     beta = -Q(:,1)’ * r/c(n);
92.     W(:,i) = r + beta * W(:,n);
93.     G(:,i) = W(:,i) + (beta/rho) * G(:,n);
94. end
95. for s = 1 : i - 1
96.     beta = -Q(:,s + 1)’ * W(:,i)/c(s);
97.     G(:,i) = G(:,i) + beta * G(:,s);
98.     W(:,i) = W(:,i) + beta * D(:,s);
99. end
100. if i < n - 1
101.     D(:,i) = W(:,i) - u;
102.     c(i) = Q(:,i + 1)’ * D(:,i);
103.     if c(i) == 0, flag = -1; return, end
104.     alpha = f/c(i);
105.     u = u - alpha * D(:,i);
106. else
107.     c(i) = Q(:,i + 1)’ * (W(:,i) - u);
108.     if c(i) == 0, flag = -1; return, end
109.     alpha = f/c(i);
110. end
111. gJ = M\G(:,i); W(:,i) = A * gJ;
112. alpha = rho * alpha;
113. x = x + alpha * gJ;
114. r = r - alpha * W(:,i);
115. err = norm(r)/bmr2;
116. iter = iter + 1;
117. if err < tol, flag = 0; return, end
118. if iter >= max_it, return, end
119. end
120. e = Q(:,1)’ * r; beta = -e/c(n);
121. W(:,n) = r + beta * W(:,n);
122. G(:,n) = W(:,n) + (beta/rho) * G(:,n);
123. if n >= 2
124.     beta = -Q(:,2)’ * W(:,n)/c(1);
125.     for s = 1 : n - 2
126.         G(:,n) = G(:,n) + beta * G(:,s);
127.         W(:,n) = W(:,n) + beta * D(:,s);
128.         beta = -Q(:,s + 2)’ * W(:,n)/c(s + 1);
129.     end
130.     G(:,n) = G(:,n) + beta * G(:,n - 1);
131. end
132. gJ = M\G(:,n); W(:,n) = A * gJ;
133. c(n) = Q(:,1)’ * W(:,n);
134. if c(n) == 0, flag = -1; return, end
135. end
\end{verbatim}
9.2. ML(n)BiCGStab with Definition (5.2). The following algorithm is a preconditioned version of Algorithm 5.1.

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

1. Choose an initial guess $x_0$ and $n$ vectors $q_1, q_2, \ldots, q_n$.
2. Compute $r_0 = b - A x_0$, $\tilde{g}_0 = M^{-1} r_0$, $w_0 = A \tilde{g}_0$, $c_0 = q_i^H w_0$ and $e_0 = q_i^H r_0$.
3. For $j = 0, 1, 2, \ldots$
4. For $i = 1, 2, \ldots, n - 1$
5. $\alpha_{jn+i} = c_{jn+i-1}/c_{jn+i-1}$; 
6. $x_{jn+i} = x_{jn+i-1} + \alpha_{jn+i} \tilde{g}_{jn+i-1}$; 
7. $r_{jn+i} = r_{jn+i-1} - \alpha_{jn+i} w_{jn+i-1}$;
8. $e_{jn+i} = q_i^H (r_{jn+i})$;
9. If $j \geq 1$
10. $\tilde{\beta}_{(j-1)n+i} = -c_{jn+i}/c_{(j-1)n+i}$; 
11. $z_w = r_{jn+i} + \tilde{\beta}_{(j-1)n+i} w_{(j-1)n+i}$;
12. $\tilde{g}_{jn+i} = \tilde{\beta}_{(j-1)n+i} \tilde{g}_{(j-1)n+i}$;
13. For $s = i + 1, \ldots, n - 1$
14. $\tilde{\beta}_{(j-1)n+s} = -q_{jn+i}^H z_w/c_{(j-1)n+s}$; 
15. $z_w = z_w + \tilde{\beta}_{(j-1)n+s} w_{(j-1)n+s}$;
16. $\tilde{g}_{jn+i} = \tilde{g}_{jn+i} + \tilde{\beta}_{(j-1)n+s} \tilde{g}_{(j-1)n+s}$;
17. End
18. $\tilde{g}_{jn+i} = M^{-1} z_w + 1/\rho_j \tilde{g}_{jn+i}$;
19. Else
20. $\tilde{g}_{jn+i} = M^{-1} r_{jn+i}$;
21. End
22. $w_{jn+i} = A \tilde{g}_{jn+i}$;
23. For $s = i, \ldots, i-1$
24. $\beta_{jn+s} = -q_{jn+i}^H w_{jn+i}/c_{jn+s}$;
25. $w_{jn+i} = w_{jn+i} + \beta_{jn+s} w_{jn+s}$;
26. $\tilde{g}_{jn+i} = \tilde{g}_{jn+i} + \beta_{jn+s} \tilde{g}_{jn+s}$;
27. End
28. $e_{jn+i} = q_i^H w_{jn+i}$;
29. End
30. $\alpha_{jn+n} = c_{jn+n-1}/c_{jn+n-1}$;
31. $x_{jn+n} = x_{jn+n-1} + \alpha_{jn+n} \tilde{g}_{jn+n-1}$;
32. $u_{jn+n} = r_{jn+n-1} - \alpha_{jn+n} w_{jn+n-1}$;
33. $\tilde{u}_{jn+n} = M^{-1} u_{jn+n}$;
34. $\rho_{j+1} = -(A \tilde{u}_{jn+n})^H u_{jn+n} / ||A \tilde{u}_{jn+n}||_2^2$;
35. $x_{jn+n} = x_{jn+n} - \rho_{j+1} u_{jn+n}$;
36. $r_{jn+n} = r_{jn+n} - \rho_{j+1} \tilde{u}_{jn+n} + u_{jn+n}$;
37. $c_{jn+n} = q_i^H r_{jn+n}$;
38. $\tilde{\beta}_{(j-1)n+n} = -e_{jn+n}/c_{(j-1)n+n}$; 
39. $\tilde{\beta}_{(j-1)n+n} = \tilde{\beta}_{(j-1)n+n} + \tilde{\beta}_{(j-1)n+n}$;
40. $\tilde{g}_{jn+n} = \tilde{\beta}_{(j-1)n+n} \tilde{g}_{(j-1)n+n}$;

For $s = 1, \ldots, n - 1$

$\tilde{\beta}_{jn+s} = -q_{jn+1}^H z_{w}/c_{jn+s}$;  \hspace{1cm} $\tilde{\beta}_{jn+s} = p_{j+1} \tilde{\beta}_{jn+s}$

$z_w = z_{w} + \tilde{\beta}_{jn+s} w_{jn+s}$;

$\tilde{\gamma}_{jn+s} = \tilde{\gamma}_{jn+s} + \tilde{\beta}_{jn+s} \tilde{\gamma}_{jn+s}$;

End

$g_{jn+n} = M^{-1} z_w + 1/\rho_{j+1} \tilde{g}_{jn+n}$;

$w_{jn+n} = A \tilde{g}_{jn+n}$;

$e_{jn+n} = q_1^H w_{jn+n}$;

End

Matlab code of Algorithm 9.2

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

2. % input: 
3. % $A$ N-by-N matrix
4. % $M$ N-by-N preconditioner matrix
5. % $Q$ N-by-n auxiliary matrix with columns $q_1, \ldots, q_n$
6. % $x$ initial guess
7. % $b$ right hand side vector
8. % $max_it$ maximum number of iterations
9. % $tol$ error tolerance
10. % $kappa$ (real number) minimization step controller:
11. % zero, standard minimization
12. % positive, Sleijpen-van der Vorst minimization
13. % output: 
14. % $x$ solution computed
15. % $err$ error norm
16. % $iter$ number of iterations performed
17. % $flag$ 0 = solution found to tolerance
18. % $-1 = breakdown$
19. % storage: 
20. % $c$ 1 x n matrix
21. % $x, r, b, u, z$ N-by-1 matrices
22. % $A, M$ N-by-N matrices
23. % $Q, G, W$ N-by-n matrices

24. $N = size(A, 2); n = size(Q, 2)$;
25. $G = zeros(N, n); W = zeros(N, n);$  \hspace{1cm} % initialize workspace for $\tilde{g}$, $w$ and $c$
26. $c = zeros(1, n);$  \hspace{1cm} % end initialization
27. $iter = 0; flag = 0; \ bnm2 = norm(b);$  \hspace{1cm} % end initialization
28. if $\ bnm2 == 0.0, \ bnm2 = 1.0; \ end$
29. $r = b - A * x; \ err = norm(r)/\ bnm2;$
30. if $\ err < tol, \ flag = 0; \ return, \ end$
31. $G(:, 1) = M \backslash r; \ W(:, 1) = A * G(:, 1); \ c(1) = Q(:, 1)' * W(:, 1);$  \hspace{1cm} % end initialization
32. if $c(1) == 0, \ flag = -1; \ return, \ end$
33. $e = Q(:, 1)' * r;$
34. for $j = 0 : max_it$
for $i = 1 : n - 1$
\begin{align*}
\alpha &= e/c(i); \\
x &= x + \alpha \ast G(:, i); \\
r &= r - \alpha \ast W(:, i); \\
err &= \text{norm}(r)/\text{bnrm2}; \\
\text{iter} &= \text{iter} + 1; \\
\text{if err < tol, flag = 0; return, end} \\
\text{if iter >= max_it, return, end} \\
\beta &= \frac{-e}{c(i + 1)}; \\
W(:, i + 1) &= r + \beta \ast W(:, i + 1); \\
G(:, i + 1) &= \beta \ast G(:, i + 1); \\
\text{for s = i + 1 : n - 1} \\
\beta &= \frac{-Q(:, s + 1) \ast r}{c(s + 1)}; \\
W(:, i + 1) &= W(:, i + 1) + \beta \ast W(:, s + 1); \\
G(:, i + 1) &= G(:, i + 1) + \beta \ast G(:, s + 1); \\
\text{end} \\
G(:, i + 1) &= (M \backslash W(:, i + 1)) + (1/rho) \ast G(:, i + 1); \\
\text{else} \\
G(:, i + 1) &= M \backslash r; \\
W(:, i + 1) &= A \ast G(:, i + 1); \\
\text{for s = 0 : i - 1} \\
\beta &= \frac{-Q(:, s + 1) \ast r}{c(s + 1)}; \\
W(:, i + 1) &= W(:, i + 1) + \beta \ast W(:, s + 1); \\
G(:, i + 1) &= G(:, i + 1) + \beta \ast G(:, s + 1); \\
\text{end} \\
c(i + 1) &= Q(:, i + 1) \ast r; \\
\text{if c(i + 1) == 0, flag = -1; return, end} \\
\alpha &= e/c(n); \\
x &= x + \alpha \ast G(:, n); \\
r &= r - \alpha \ast W(:, n); \\
err &= \text{norm}(r)/\text{bnrm2}; \\
\text{if err < tol, flag = 0; iter = iter + 1; return, end} \\
\omega = z' \ast r; \\
\text{if omega == 0, flag = -1; return, end} \\
\rho &= -\omega / \text{norm}(z); \\
\text{if kappa > 0} \\
\omega &= \omega / (\text{norm}(z) \ast \text{norm}(r)); \\
\text{abs_om} &= \text{abs}(\omega); \\
\text{if abs_om < kappa} \\
\rho &= \rho \ast \text{kappa} / \text{abs_om}; \\
\text{end} \\
\text{end} \end{align*}
\[ x = x - \rho * u J; \]
\[ r = r + \rho * z; \]
\[ \text{err} = \text{norm}(r)/bnrm2; \]
\[ \text{iter} = \text{iter} + 1; \]
\[ \text{if err < tol, flag = 0; return, end} \]
\[ \text{if iter >= max_it, return, end} \]
\[ e = Q(:,1)' * r; \]
\[ \beta = -e/c(1); \]
\[ W(:,1) = r + \beta * W(:,1); \]
\[ G(:,1) = \beta * G(:,1); \]
\[ \text{for s = 1 : n - 1} \]
\[ \beta = -Q(:,s + 1)' * W(:,1)/c(s + 1); \]
\[ W(:,1) = W(:,1) + \beta * W(:,s + 1); \]
\[ G(:,1) = G(:,1) + \beta * G(:,s + 1); \]
\[ \text{end} \]
\[ G(:,1) = (M\backslash W(:,1)) + (1/\rho) * G(:,1); \]
\[ W(:,1) = A * G(:,1); \]
\[ c(1) = Q(:,1)' * W(:,1); \]
\[ \text{if c(1) == 0, flag = -1; return, end} \]
\[ \text{end} \]

A sample run of ML(n)BiCGstab
1. \( N = 100; A = \text{randn}(N); M = \text{randn}(N); b = \text{randn}(N,1); \)
2. \( n = 10; \kappa = 0.7; \text{tol} = 10^{-7}; \text{max_it} = 3 * N; \)
3. \( Q = \text{sign}(\text{randn}(N,n)); x = \text{zeros}(N,1); Q(:,1) = b - A * x; \)
4. \([x, \text{err}, \text{iter}, \text{flag}] = \text{mlbicgstab}(A,x,b,Q,M,\text{max_it},\text{tol},\kappa); \)

Acknowledgements. The index functions were introduced by Prof. Daniel Boley in [34]. 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 the whole manuscript and making helpful comments and suggestions. The author also acknowledge Dr. Jens-Peter M. Zemke for the helpful communications and up-to-date information about Krylov subspace methods.

The author is grateful to his Ph.D. advisor Prof. Tony Chan for guiding him into the area of Krylov subspace methods. The method was originally named MLBiCGStab(n) by Tony. The current and more accurate name was from Prof. Henk A. van der Vorst.

Part of this work has been done during the visits of the author at Delft University of Technology, The Netherlands, in June 2009, and The Chinese University of Hong Kong during the summers of the past few years.

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] A. Edelman, Eigenvalues and condition numbers of random matrices, SIAM J. Matrix Anal. Appl., 9(1988), 543-560.
[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), 129-142.
[4] R. Fletcher, Conjugate gradient methods for indefinite systems, volume 506 of Lecture Notes Math., pages 73-89. Springer-Verlag, Berlin-Heidelberg-New York, 1976.
[5] 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.
[6] ——, Variants of BiCGStab for matrices with complex spectrum, SIAM J. Sci. Comput., 14 (1993), pp. 1020–1033.
[7] ——, A completed theory of the unsymmetric Lanczos process and related algorithms. Part II., SIAM J. Matrix Anal. Appl. 1994, 15:15-58.
[8] ——, Lanczos-type solvers for nonsymmetric linear systems of equations, Acta Numerica, 6 (1997), pp. 271-397.
[9] ——, IDR Explained, submitted to ETNA.
[10] R. H. Horn and C. Johnson, Matrix analysis, Cambridge University Press, 1985.
[11] 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.
[12] ——, Lanczos methods for the solution of nonsymmetric systems of linear equations, SIAM Journal on Matrix Analysis and Applications 1992; 13:926-943.
[13] C. Lanczos, Solution of systems of linear equations by minimized iterations, J. Research Nat. Bureau of Standards, 49 (1952), pp. 33–53.
[14] D. O'Leary, The block conjugate gradient algorithm and related methods, Linear Algebra Appl., 29(1980), pp. 293-322.
[15] K. Moriya and T. Nodera, Breakdown-free ML(k)/BiCGStab algorithm for non-Hermitian linear systems, O. Gervasi et al. (Eds.): ICCS 2005, LNCS 3433, pp. 978-988, 2005.
[16] 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.
[17] ——, Iterative methods for sparse linear systems, 2nd edition, SIAM, Philadelphia, PA, 2003.
[18] 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.
[19] 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.
[20] G. L. G. Sleijpen and D. R. Fokkema, BiCGSTAB(l) for linear systems involving nonsymmetric matrices with complex spectrum, ETNA, 1:11-32, 1993.
[21] G. L. G. Sleijpen, P. Sonneveld, and M. B. van Gijzen, Bi-CGSTAB as an induced dimension reduction method, Preprint 1369, Dep. Math., University Utrecht, (April, 2008).
[22] G. L. G. Sleijpen and H. A. van der Vorst,维持 convergence properties of BiCGSTAB methods in finite precision arithmetic, Numer. Algorithms, 10 (1995), pp. 203–223.
[23] ——, An overview of approaches for the stable computation of hybrid BiCG methods, Appl. Numer. Math., 19 (1995), pp. 235–254.
[24] ——, Reliable updated residuals in hybrid Bi-CG methods, Computing, 56 (1996), pp. 141–163.
[25] 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.
[26] P. Sonneveld, CGS, a fast Lanczos-type solver for nonsymmetric linear systems, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 36–52.
[27] 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.
[28] ——, IDR(s): a family of simple and fast algorithms for solving large nonsymmetric linear systems, SIAM J. Sci. Comput., 31(2009), pp. 1035-1062.
[29] 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.
[30] ——, Iterative Krylov methods for large linear systems, Cambridge University Press, Cambridge, April 2003.
[31] 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.
[32] M. van Gijzen and P. Sonneveld, An elegant IDR(s) variant that efficiently exploits bi-orthogonality properties, Delft University of Technology, Reports of the Department of Applied Mathematical Analysis, Report 08-21.
[33] 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.
[34] 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.

[35] 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.

[36] Shao-Liang Zhang, *GPBi-CG: Generalized product-type methods based on Bi-CG for solving nonsymmetric linear systems*, SIAM J. Sci. Comput., 18:537-551, 1997.