ML(N)BICGSTABT: A ML(N)BICGSTAB VARIANT WITH A-TRANSPOSE

MAN-CHUNG YEUNG∗

Abstract. The 1980 IDR method[26] plays an important role in the history of Krylov subspace methods. It started the research of transpose-free Krylov subspace methods. In this paper, we make a first attempt to bring back A-transpose to the research area by presenting a new ML(n)BiCGStab variant that involves A-transpose in its implementation. Comparisons of this new algorithm with the existing ML(n)BiCGStab algorithms will be presented.

Key words. IDR, 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. ML(n)BiCGStab is a transpose-free Krylov subspace method for the solution of linear systems

$$Ax = b$$

where A ∈ C^{N×N} and b ∈ C^N. It was introduced by Yeung and Chan[30] in 1999 and its algorithms were recently reformulated by Yeung[27]. ML(n)BiCGStab is a natural generalization of BiCGStab[25], built from a multiple starting BiCG-like algorithm called ML(n)BiCG, through the Sonneveld-van der Vorst-Lanczos procedure (SVLP), namely, the procedure introduced by Sonneveld[21] and van der Vorst[25] in the construction of CGS and BiCGStab from BiCG[2]. In theory, ML(n)BiCGStab is a method that lies between the Lanczos-based BiCGStab and the Arnoldi-based GMRES/FOM[17]. In fact, it is a BiCGStab when n = 1 and becomes a GMRES/FOM when n = N (see[27, 28]). In computation, ML(n)BiCGStab can be much more stable and converge much faster than BiCGStab. We once tested it on the standard oil reservoir simulation test data called SPE9 which contains a sequence of linear systems and found that it reduced the total computational time by 60% when compared to BiCGStab. Tests made on the data from matrix markets also supported the superiority of ML(n)BiCGStab over BiCGStab. For details, one is referred to[27, 30].

The author once constructed a new version of ML(n)BiCG where the left residuals are not just given by the monomial basis, but are orthogonalized against previous right-hand side residuals. In structure, this new ML(n)BiCG is closer to the classical BiCG than the one in[30] is. Numerical experiments, however, showed that this new ML(n)BiCG was unstable and weaker than the standard BiCG. Moreover, in[29], Yeung and Boley derived a SVLP from a one-sided multiple starting band Lanczos procedure (MSLP) with n left-starting and m right-starting vectors respectively. From their experiments with multi-input multi-output time-invariant linear dynamical systems, they observed that SVLP is more stable than MSLP when m ≠ n. The two examples of comparison hint that, when m ≠ n, a stable multiple starting procedure with A-transpose may come from a modification of a SVLP. In this paper, we make a first step in this direction by introducing A-transpose into ML(n)BiCGStab. We call the resulting algorithm ML(n)BiCGStabt, standing for ML(n)BiCGStab with transpose.

∗Dept. 3036, 1000 East University Avenue, Laramie, WY 82071. E-mail: myeung@uwyo.edu.
There exist two ML(n)BiCGStab algorithms, labeled as Algorithms 4.1 and 5.1 respectively in [27], derived from different definitions of the residual vectors \( r_k \). While both algorithms are numerically stable in general, one is relatively more stable than the other. ML(n)BiCGStabt is a modified version of Algorithm 5.1 so that it enjoys the same level of stability with Algorithm 4.1.

Other extensions of IDR, CGS and BiCGStab exist. Among them are BiCGStab[6], BiCGStab(l)[15], GPBi-CG[31], IDR(s)[23, 4], IDRstab[19], and GBi-CGSTAB(s, l)[24]. Related articles include [1, 9, 10, 22].

The outline of the paper is as follows. In §2, index functions in [29] are introduced. They are helpful in the construction of a ML(n)BiCGStab algorithm. In §3, we present the ML(n)BiCG algorithm from [30]. The derivation of every ML(n)BiCGStab algorithm is based on it. In §4, we introduce the ML(n)BiCGStabt algorithm and its properties. In §5, numerical experiments are presented, and in §6, concluding remarks are given.

2. Index Functions. Let be given a \( n \in \mathbb{N} \), the set of positive integers. For all \( k \in \mathbb{Z} \), the set of all integers, we define

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

where \( \lfloor \cdot \rfloor \) rounds its argument to the nearest integer towards minus infinity. We call \( g_n \) and \( r_n \) index functions; they are defined on \( \mathbb{Z} \) with ranges \( \mathbb{Z} \) and \( \{1, 2, \ldots, n\} \), respectively.

If we write

\[
k = j n + i
\]

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

\[
g_n(j n + i) = j \quad \text{and} \quad r_n(j n + i) = i.
\]

3. ML(n)BiCG. Analogously to the derivation of BiCGStab from BiCG, ML(n)BiCGStab algorithms were derived from a BiCG-like algorithm 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 ML(n)BiCG algorithm from [30].

Consider the solution of (1.1). Throughout the paper we do not assume the coefficient matrix \( A \) is nonsingular. In [27], we proved that ML(n)BiCG/ML(n)BiCGStab can solve a singular system almost surely provided that the underlining Krylov subspace contains a solution of (1.1).

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

\[
p_k = (A^H)^{g_n(k)} q_{r_n(k)}, \quad k \in \mathbb{N}.
\]

The following algorithm for the solution of (1.1) is from [30].

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 \( \tilde{r}_0 = b - A \tilde{x}_0 \) 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. End
10. $\hat{g}_k = \tilde{r}_k + \sum_{s=\max(k-n, 0)}^{k-1} \beta_s^{(k)} \tilde{g}_s$;
11. Compute $p_{k+1}$ according to [3.7]
12. End

This ML(n)BiCG algorithm is a variation of the classical BiCG algorithm with the left-hand side (shadow) Krylov subspace of BiCG being replaced by the block Krylov subspace

$$B_k \equiv \text{span of the first } k \text{ columns of } [Q, AHQ, (AH)^2 Q, \ldots]$$

$$= \text{span} \{ p_1, p_2, \ldots, p_k \}$$

$$= \sum_{i=1}^{r_n(k)} K_{g_n(k)+1}(AH, q_i) + \sum_{i=r_n(k)+1}^{n} K_{g_n(k)}(AH, q_i)$$

where $Q \equiv [q_1, q_2, \ldots, q_n]$, $K_0(M, v) = \{ 0 \}$ and $K_t(M, v) \equiv \text{span} \{ v, M v, \ldots, M^{t-1} v \}$

for $M \in \mathbb{C}^{N \times N}$, $v \in \mathbb{C}^N$ and $t \in \mathbb{N}$. Moreover, in this ML(n)BiCG, the basis used for $B_k$ is not chosen to be bi-orthogonal, but simply the set $\{ p_1, p_2, \ldots, p_k \}$. Therefore, it can be viewed as a generalization of a one-sided Lanczos algorithm (see [8, 16]).

It can be shown that the quantities of ML(n)BiCG satisfy the properties (see [27])

(a) $\tilde{x}_k \in \tilde{x}_0 + K_k(A, \tilde{r}_0)$, $\tilde{r}_k \in \tilde{r}_0 + A K_k(A, \tilde{r}_0)$.
(b) $\tilde{r}_k \perp \text{span} \{ p_1, p_2, \ldots, p_k \}$ and $\tilde{r}_k \not\perp p_{k+1}$.
(c) $A \tilde{g}_k \perp \text{span} \{ p_1, p_2, \ldots, p_k \}$ and $A \tilde{g}_k \not\perp p_{k+1}$.

4. ML(n)BiCGStab. The derivation of a ML(n)BiCGStab algorithm from ML(n)BiCG essentially is a Sonneveld-van der Vorst-Lanczos procedure. The central idea of this procedure is the remarkable observation: inner products $p^H \tilde{r}$ and $p^H A \tilde{g}$ in BiCG can be replaced by inner products of the forms $q^H \psi(A) \tilde{r}$ and $q^H A \psi(A) \tilde{g}$ respectively, where $\psi$ is an arbitrary polynomial with some suitable degree. This observation can also applied to ML(n)BiCG because of properties (b) and (c) stated in [3].

4.1. Algorithm. In [27], Yeung presented two ML(n)BiCGStab algorithms, labeled as Algorithms 4.1 and 5.1 respectively. Let $\phi_k$ be the polynomial of degree $k$, recursively defined by

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

where $\omega_k$ is a free parameter. Then the quantities in Algorithm 4.1 are defined by

$$r_k = \phi_{g_n(k)+1}(A) \tilde{r}_k, \quad u_k = \phi_{g_n(k)}(A) \tilde{r}_k, \quad g_k = \phi_{g_n(k)+1}(A) \tilde{g}_k, \quad d_k = -\omega_{g_n(k)+1} A \phi_{g_n(k)}(A) \tilde{g}_k,$$

for $k > 0$, and those in Algorithm 5.1 defined as

$$r_k = \phi_{g_n(k)+1}(A) \tilde{r}_k, \quad u_k = \phi_{g_n(k)}(A) \tilde{r}_k, \quad g_k = \phi_{g_n(k)+1}(A) \tilde{g}_k, \quad w_k = A \tilde{g}_k,$$

(4.1)
for \( k > 0 \). When \( k = 0 \), both algorithms set
\[
\mathbf{r}_0 = \mathbf{r}_0 \quad \text{and} \quad \mathbf{g}_0 = \mathbf{g}_0.
\]
Here \( \mathbf{r}_k \) is the residual of the \( k \)th approximate solution \( \mathbf{x}_k \). Numerical experiments in \cite{Yeung2006} indicated that the \( \mathbf{r}_k \) computed by Algorithm 4.1 is generally closer to the true residual \( \mathbf{b} - \mathbf{A}\mathbf{x}_k \) than the \( \mathbf{r}_k \) computed by Algorithm 5.1 is. A close examination of the algorithms can explain this difference in stability.

In both algorithms, the \( \mathbf{x}_k \) and \( \mathbf{r}_k \) are updated by the recursive relations
\[
\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k \mathbf{g}_{k-1}, \quad \mathbf{r}_k = \mathbf{r}_{k-1} - \alpha_k \mathbf{w}_{k-1}
\]
in most \( k \)-iterations, where \( \alpha_k \) is a scalar. The true residual of the computed \( \mathbf{x}_k \) is therefore
\[
\mathbf{b} - \mathbf{A}\mathbf{x}_k = \mathbf{b} - \mathbf{A}(\mathbf{x}_{k-1} + \alpha_k \mathbf{g}_{k-1}) = (\mathbf{b} - \mathbf{A}\mathbf{x}_{k-1}) - \alpha_k \mathbf{A}\mathbf{g}_{k-1}. \quad (4.3)
\]
In Algorithm 4.1, \( \mathbf{w}_k \) is updated by \( \mathbf{w}_k = \mathbf{A}\mathbf{g}_k \) (as it is defined in \cite{Yeung2006}) for all \( k \). In Algorithm 5.1, however, \( \mathbf{w}_k \) is updated by \( \mathbf{w}_k = \mathbf{A}\mathbf{g}_k \) only when \( r_n(k) = n \). In other words, the update for \( \mathbf{r}_k \) in Algorithm 4.1 follows closer to (4.3) than the \( \mathbf{r}_k \) computed by Algorithm 5.1 is. Because of the observation, we expect that Algorithm 5.1 should be as stable as Algorithm 4.1 if we could modify the algorithm so that its \( \mathbf{w}_k \) were updated by \( \mathbf{w}_k = \mathbf{A}\mathbf{g}_k \) in all the iterations — this is the goal that we develop ML(\( n \))BiCGStab.

The derivation of Algorithm 5.1 in \cite{Yeung2006} was divided into several stages, starting from ML(\( n \))BiCG. The following is a copy of its Derivation Stage #8 which is a list of equations that the quantities in \cite{Yeung2006} satisfy.

**Derivation Stage #8 in \cite{Yeung2006}**.

1. For \( k = 1, 2, \ldots \), until convergence:
2. \[ \alpha_k = q^H_{r_n(k)} r_{k-1} / q^H_{r_n(k)} w_{k-1} \]
3. If \( r_n(k) < n \)
4. \[ \mathbf{r}_k = \mathbf{r}_{k-1} - \alpha_k \mathbf{w}_{k-1} \]
5. For \( s = \max(k - n, 0), \ldots, n \)
6. \[ \beta_s(k) = q^H_{r_n(s+1)} (\mathbf{r}_k - \omega g_n(1) \sum_{t=\max(k-n,0)}^{s-1} \beta_t(k) w_t) / \omega g_n(1) q^H_{r_n(s+1)} w_s \]
7. End
8. For \( s = g_n(k)n, \ldots, k - 1 \)
9. \[ \beta_s(k) = -q^H_{r_n(s+1)} (\mathbf{A}\mathbf{r}_k + \sum_{t=\max(k-n,0)}^{n-1} \beta_t(k) (\mathbf{I} - \omega g_n(k+1) \mathbf{A}) w_t + \sum_{t=g_n(k)n}^{s-1} \omega g_n(k)n \beta_t(k) w_t) / q^H_{r_n(s+1)} w_s \]
10. End
11. \[ \mathbf{g}_k = \mathbf{r}_k - \omega g_n(1) \sum_{s=\max(k-n,0)}^{g_n(k)n-1} \beta_s(k) w_s + \sum_{s=\max(k-n,0)}^{s=g_n(k)n-1} \omega g_n(k+1) \beta_s(k) g_s + \sum_{s=g_n(k)n}^{k-1} \beta_s(k) g_s \]
12. Else
13. \[ \mathbf{u}_k = \mathbf{r}_{k-1} - \alpha_k \mathbf{w}_{k-1} \]
14. \[ \mathbf{r}_k = (\mathbf{I} - \omega g_n(k+1) \mathbf{A}) \mathbf{u}_k \]
15. For \( s = g_n(k)n, \ldots, k - 1 \)
16. \[ \beta_s(k) = q^H_{r_n(s+1)} (\mathbf{r}_k - \omega g_n(1) \sum_{t=g_n(k)n}^{s-1} \beta_t(k) w_t) / \omega g_n(1) q^H_{r_n(s+1)} w_s \]

\footnote{There is a similar comment on BiCGStab(l) \cite{Yeung2008} p.27 when compared to BiCGStab2 \cite{Yeung2008}.}
17. \[ g_k = r_k - \omega g_{n(k)} \sum_{s=g_{n(k)}n}^{k-1} \beta_s^{(k)} w_s + \sum_{s=g_{n(k)}n}^{k-1} \beta_s^{(k)} g_s; \]
18. End
19. End
20. End

According to (4.2), the equation in Line 9 can be rewritten as
\[
\beta_s^{(k)} = -q^H_{r_{n(s+1)}} A \left( r_k + \sum_{t=\max(k-n,0)}^{g_{n(k)}n-1} \beta_t^{(k)} (I - \omega g_{n(k+1)} A) g_t \right) \\
+ \sum_{t=g_{n(k)}n}^{s-1} \beta_t^{(k)} g_t / q^H_{r_{n(s+1)}} w_s \\
= -q^H_{r_{n(s+1)}} A \left( r_k + \sum_{t=\max(k-n,0)}^{g_{n(k)}n-1} \beta_t^{(k)} (g_t - \omega g_{n(k+1)} w_t) \right) \\
+ \sum_{t=g_{n(k)}n}^{s-1} \beta_t^{(k)} g_t / q^H_{r_{n(s+1)}} w_s.
\]

It is because of the \( A \) before the parentheses, we can not update \( w_k \) by \( w_k = Ag_k \) in Algorithm 5.1 while keeping the average number of matrix-vector multiplications as low as \( 1 + 1/n \) per iteration. If, however, the vector \( f_{r_{n(s+1)}} = A^H q_{r_{n(s+1)}} \) is available, then Line 9 will become
\[
\beta_s^{(k)} = -f^H_{r_{n(s+1)}} \left( r_k + \sum_{t=\max(k-n,0)}^{g_{n(k)}n-1} \beta_t^{(k)} (g_t - \omega g_{n(k+1)} w_t) \right) \\
+ \sum_{t=g_{n(k)}n}^{s-1} \beta_t^{(k)} g_t / q^H_{r_{n(s+1)}} w_s.
\]

and the troubling \( A \) is gone. It is the observation that leads to the ML(n)BiCGStabT algorithm.

Replace Line 9 in Derivation Stage #8 with (4.4) and suppose
\[ F = A^H Q = [A^H q_1, A^H q_2, \ldots, A^H q_n] \]
is available. Recalling that \( x_k \) is the residual of \( x_k \), to be consistent with Lines 4, 13 and 14, we update the approximate solution \( x_k \) as
\[
x_k = \begin{cases} 
  x_{k-1} + \alpha_k g_{k-1}, & \text{if } r_{n(k)} < n \\
  \omega g_{n(k+1)} x_{k-1} + x_{k-1} + \alpha_k g_{k-1}, & \text{if } r_{n(k)} = n.
\end{cases}
\] (4.5)

Now adding (4.3) and \( w_k = Ag_k \) to the derivation stage, then simplifying the operations appropriately, we arrive at the following algorithm. The free parameter \( \omega_{g_{n(k+1)}} \) is chosen to minimize the 2-norm of \( r_k \).

**Algorithm 4.1. ML(n)BiCGStabT without preconditioning**

1. Choose an initial guess \( x_0 \) and \( n \) vectors \( q_1, q_2, \ldots, q_n \).
2. Compute \( [f_1, \ldots, f_{n-1}] = A^H [q_1, \ldots, q_{n-1}] \).
3. Compute \( r_0 = b - Ax_0 \) and \( g_0 = r_0 \), \( w_0 = Ag_0 \), \( c_0 = q^H_1 w_0 \), \( \omega_0 = 1 \).
4. For \( k = 1, 2, \ldots, \) until convergence:
5. \( \alpha_k = q^H_{r_{n(k)}} r_{k-1} / c_{k-1}; \)
6. If \( r_{n(k)} < n \)
7. \( x_k = x_{k-1} + \alpha_k g_{k-1}; \)
8. \( g_k = \frac{r_{k-1} - \alpha_k w_{k-1}}{c_k}; \)
9. For \( s = \max(k-n, 0), \ldots, g_{n(k)n} - 1 \)
10. \( \tilde{\beta}_s^{(k)} = -q^H_{r_{n(s+1)}} z_w / c_s; \)
11. \( z_w = z_w + \tilde{\beta}_s^{(k)} w_s; \)
Table 4.1
Average cost per k-iteration of the preconditioned ML(n)BiCGStabt Algorithm [7.1] and its storage. This table does not count the cost in Lines 1-2 of the algorithm.

| Preconditioning $M^{-1}v$ | 1 + 1/n | $u \pm v$, $\alpha v$ | 1 |
|----------------------------|---------|-------------------|---|
| Matvec $Av$                | 1 + 1/n | $u \pm v$, $\alpha v$ | 1.5$n + 2.5 + 2/n$ |
| dot product $u^Tv$        | $n + 1/2/n$ | Storage | $(A + M + (4n + 4)N + O(n)$ |

12. $g_k = g_k + \beta_k^{(k)} s_k$;
13. End
14. $g_k = z_w - \frac{1}{\omega_{g_n(k+1)|g_k|} g_k}$;
15. For $s = g_n(k) n, \ldots, k - 1$
16. $\beta_k^{(k)} = -r_{r_n(s+1)}/c_s$;
17. $g_k = g_k + \beta_k^{(k)} s_k$;
18. End
19. Else
20. $x_k = x_{k-1} + \alpha_k g_{k-1}$;
21. $u_k = r_{k-1} - \alpha_k w_{k-1}$;
22. $\omega_{g_n(k+1)} = (Au_k)^H u_k / \|Au_k\|^2$;
23. $x_k = x_k + \omega_{g_n(k+1)} u_k$; $r_k = -\omega_{g_n(k+1)} A u_k + u_k$;
24. $z_w = r_k$; $g_k = 0$;
25. For $s = g_n(k) n, \ldots, k - 1$
26. $\beta_s^{(k)} = -q_{r_n(s+1)}^H z_w / c_s$; $\beta_k^{(k)} = -\omega_{g_n(k+1)} \beta_s^{(k)}$
27. $z_w = z_w + \beta_s^{(k)} w_s$;
28. $g_k = g_k + \beta_s^{(k)} g_s$;
29. End
30. $g_k = z_w - \frac{1}{\omega_{g_n(k+1)}} g_k$;
31. End
32. $w_k = A g_k$; $c_k = q_{r_n(s+1)}^H w_k$;
33. End

Line 32 indicates that $w_k$ is computed by $w_k = A g_k$ for all $k$-iterations. Therefore the updates for $x_k$ and $r_k$ in the above Algorithm [4.1] are

$$x_k = x_{k-1} + \alpha_k g_{k-1}, \quad r_k = r_{k-1} - \alpha_k A g_{k-1}$$

which meets the goal that we set right before Derivation Stage #8 on improving the stability of Algorithm 5.1 in [27]. The stability of updates of the type (4.6) has been studied in detail by Neumaier [14] and Sleijpen and van der Vorst [20].

We remark that (i) Algorithm [4.1] does not compute $u_k$ when $r_n(k) < n$. In fact, $u_k = r_k$ when $r_n(k) < n$ from (4.2); (ii) if the $u_k$ in Line 21 happens to be zero, then the $x_k$ in Line 20 will be the exact solution to system [1.1] and the algorithm stops there.

Computational and storage cost based on the preconditioned ML(n)BiCGStabt (see Algorithm [4.1]) is presented in Table 4.1. Note that we do not need to store both $A$ and $A^H$ since $A^H$ is only used in Line 2. Compared with Algorithm 5.1 in [27], the computational cost of ML(n)BiCGStabt is slightly cheaper.

Theoretically, it can be guaranteed that an exact breakdown in Algorithm [4.1]
is almost impossible (see [27] for a detailed analysis). The algorithm, however, can encounter a near breakdown in its implementation. The divisors in the algorithm are \(c_k, \|Au_k\|_2\) and \(\omega_{g_n(k+1)}\). If \(\|Au_k\|_2 \approx 0\) and the \(x_k\) in Line 20 is an approximate solution. If \(\omega_{g_n(k+1)} \approx 0\), we can add some small perturbation to it so that it is relatively far from 0. About \(c_k\), it can be showed that it is a quantity that relates to \(\omega_{g_n(k+1)}\) and the ML(\(n\))BiCG divisor \(p_{k+1}^HA_{g_k}\). The ML(\(n\))BiCG divisor \(p_{k+1}^HA_{g_k}\) is in turn related to the underlying Lanczos breakdown and the breakdown caused by the non-existence of the LU factorization of the Hessenberg matrix of the recurrence coefficients. But, as indicated in [6], in most cases such breakdowns can be overcome by a look-ahead step, see [3, 5, 7, 15] and further references cited there. Moreover, for how to avoid a breakdown in a nonsymmetric block Lanczos algorithm, one can consult [13].

4.2. Properties. Since the quantities of ML(\(n\))BiCGStabt are defined exactly the same as those of Algorithm 5.1 in [27], ML(\(n\))BiCGStabt shares the same properties with Algorithm 5.1.

Let \(\nu\) be the degree of the minimal polynomial \(p_{\min}(\lambda; A, r_0)\) of \(r_0\) with respect to \(A\), namely, the unique monic polynomial \(p(\lambda)\) of minimum degree such that \(p(A)r_0 = 0\), and let

\[
S_\nu = [p_1, p_2, \ldots, p_\nu]^H A[r_0, A r_0, \ldots, A^{\nu-1} r_0]
\]

and

\[
W_\nu = [p_1, p_2, \ldots, p_\nu]^H [r_0, A r_0, \ldots, A^{\nu-1} r_0].
\]

Denote by \(S_j\) and \(W_j\) the \(l \times l\) leading principal submatrices of \(S_\nu\) and \(W_\nu\), respectively. Joubert [11, 12] called these matrices the moment matrices. With the notations, some facts about ML(\(n\))BiCGStabt (Algorithm 4.1) are summarized as follows.

**Proposition 4.2.** [27, Prop. 5.1] In infinite precision arithmetic, if \(\prod_{i=1}^\nu \det(S_i) \neq 0\), \(\omega_{g_n(k+1)} \neq 0\) and \(1/\omega_{g_n(k+1)} \notin \sigma(A)\) 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 \(k = 1, 2, \ldots, \nu\), and the approximate solution \(x_\nu\) at step \(k = \nu\) is exact to the system \([\nu]\). Moreover, the computed quantities satisfy

(a) \(x_k \in x_0 + \text{span}\{r_0, A r_0, \ldots, A^{g_n(k+1)+k-1} r_0\}\) and \(r_k = b - A x_k \in r_0 + \text{span}\{A r_0, A^2 r_0, \ldots, A^{g_n(k+1)+k} r_0\}\) for \(1 \leq k \leq \nu - 1\).

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

(c) \(r_k \perp \text{span}\{q_1, q_2, \ldots, q_{r_n(k)}\}\) and \(r_k \notin q_{r_n(k)+1} \) for \(1 \leq k \leq \nu - 1\) with \(r_n(k) < n; r_k \notin q_1 \) for \(1 \leq k \leq \nu - 1\) with \(r_n(k) = n\).

(d) \(u_k \perp \text{span}\{q_1, q_2, \ldots, q_{r_n(k)}\}\) for \(1 \leq k \leq \nu\) with \(r_n(k) = n\).

(e) \(A g_k \perp \text{span}\{q_1, q_2, \ldots, q_{r_n(k)}\}\) and \(A g_k \notin q_{r_n(k)+1} \) for \(1 \leq k \leq \nu - 1\) with \(r_n(k) < n; A g_k \notin q_1 \) for \(1 \leq k \leq \nu - 1\) with \(r_n(k) = n\).

In §6.2 of [27], relations of Algorithm 5.1 in [27] to some existing methods were presented. The same arguments applied to ML(\(n\))BiCGStabt imply that

(a) ML(\(n\))BiCGStabt is a FOM algorithm, but involving \(A^H\) in its implementation, if we set \(n \geq \nu\) and \(q_k = r_{k-1}\).

(b) ML(\(n\))BiCGStabt is a BiCGStab algorithm if we set \(n = 1\).

(c) ML(\(n\))BiCGStabt is a IDR(\(s\)) algorithm with \(s = n\), but involving \(A^H\) in its implementation.
5. Numerical experiments. A preconditioned ML(n)BiCGStabt algorithm can be obtained by applying Algorithm 4.1 to the system

$$AM^{-1}y = b$$

where M is nonsingular, then recovering x through $$x = M^{-1}y$$. The resulting algorithm, Algorithm 7.1 together with its Matlab code are presented in §7. 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 much as possible in the resulting preconditioned algorithm.

We compared ML(n)BiCGStabt with BiCG, BiCGStab and two algorithms of ML(n)BiCGStab: Algorithms 4.1 and 5.1 in [27]. All test data were downloaded from The University of Florida Sparse Matrix Collection, and the computing was done in Matlab Version 7.1 on a Windows XP machine with a Pentium 4 processor.

In all the experiments, we chose the initial guess $$x_0 = 0$$, the stopping criterion $$\|r_k\|/\|b\| < 10^{-7}$$ where $$r_k$$ was the computed residual, and the Sleijpen-van der Vorst minimization control parameter (see [27]) $$\kappa = 0$$. As for the shadow vectors, we chose $$Q = [r_0, \text{sign(randn}(N,n-1))]$$. When a data did not provide a right-hand side, we set $$b = Ae$$ where e is the vector of ones.

Example 1. We ran all the methods on the selected group of matrices in Table 5.1. No preconditioner was used. The results are summarized in Tables 5.2 - 5.4. The “True error” columns in the tables contain the true relative errors $$\|b - Ax\|/\|b\|$$ where x is the computed solution output by an algorithm when it converges. In this experiment, we observe that ML(n)BiCGStabt and ML(n)BiCGStab generally outperform BiCG and BiCGStab in terms of computational time. As an improved version of Algorithm 5.1 in [27], ML(n)BiCGStabt has the same stability with Algorithm 4.1 in [27] and is slightly more stable than Algorithm 5.1.

Example 2. Our experience with the Florida collection has shown that Algorithm 5.1 in [27] is overall a stable algorithm. But still, one can find one or two matrices where it is unstable. Consider the data

1. e0r0100, a 2D/3D problem from the Shen group. The coefficient matrix is a 17281-by-17281 real unsymmetric matrix with 553,562 nonzero entries.
2. utm5940, an electromagnetics problem from the TOKAMAK group. The coefficient matrix is a 5940-by-5940 real unsymmetric matrix with 83,842 nonzero entries.

In this experiment, ILU preconditioners generated by the Matlab command [L,U,P] = luinc(A, 1e-3) were used. For the ease of presentation, we introduce the true relative error function $$E(n) = \|b - Ax\|/\|b\|$$ where x is the computed solution output by a ML(n)BiCGStab algorithm when it converges. The graphs of $$E(n)$$ are plotted in Figure 5.1. It can be seen that the computed relative errors $$\|r_k\|/\|b\|$$ by Algorithm 5.1 significantly diverge from their exact counterparts. By contrast, however, the computed $$\|r_k\|/\|b\|$$ by ML(n)BiCGStabt and Algorithm 4.1 in [27] well approximate their corresponding true relative errors. In this experiment, the improvement on stability of ML(n)BiCGStabt over Algorithm 5.1 is significant.

6. Concluding Remarks. The original motivation of developing ML(n)BiCGStabt was to improve the stability of Algorithm 5.1 in [27]. From our experiments, the improvement can sometimes be significant. Since, however, the two algorithms are

\[\text{http://www.cise.ufl.edu/research/sparse/matrices/}\]
essentially the same in structure, they basically share the same theoretical and numerical properties. A generalization of ML(n)BiCGStabt to ML(n)BiCGStabt2 and ML(n)BiCGStabtl are being carried out. They are clearly different from ML(n)BiCGStabt in structure and thereby we expect different properties that these algorithms will have.

Now, it can be seen that ML(n)BiCGStabt should be the first method getting involved in its implementation in the area of product-type or hybrid BiCG methods since the 1980 IDR method was published.

7. Appendix. In this section, we present a preconditioned ML(n)BiCGStabt algorithm together with its Matlab code.
Algorithm 7.1. ML(n)BiCGStab with preconditioning

1. Choose an initial guess $x_0$ and $n$ vectors $q_1, q_2, \ldots, q_n$.
2. Compute $f_1, \ldots, f_{n-1} = M^{-1/2}A^H[q_1, \ldots, q_{n-1}]$, $r_0 = b - Ax_0$ and $g_0 = r_0$.
   Compute $g_0 = M^{-1}r_0$, $w_0 = A\hat{g}_0$, $c_0 = q_i^Hw_0$, $c_0 = q_i^Hr_0$.
3. For $j = 0, 1, 2, \ldots$
4. For $i = 1, 2, \ldots, n - 1$
5. $\alpha_{jn+i} = c_{jn+i}/c_{jn+i-1}$;
6. $x_{jn+i} = x_{jn+i-1} + \alpha_{jn+i}g_{jn+i-1}$;
7. $r_{jn+i} = r_{jn+i-1} - \alpha_{jn+i}w_{jn+i-1}$;
8. $c_{jn+i} = q_i^Hr_{jn+i}$;
9. If $j \geq 1$
10. $\beta_{(j-1)n+i} = -c_{(j-1)n+i}/c_{(j-1)n+i-1}$;
11. $z_w = r_{jn+i} + \beta_{(j-1)n+i}w_{(j-1)n+i}$;
12. $g_{jn+i} = \beta_{(j-1)n+i}g_{(j-1)n+i}$;
13. For $s = i + 1, \ldots, n - 1$
14. $\beta_{(j-1)n+s} = -q_i^Hz_w/c_{(j-1)n+s}$;
15. $z_w = z_w + \beta_{(j-1)n+s}w_{(j-1)n+s}$;
16. $g_{jn+i} = g_{jn+i} + \beta_{(j-1)n+s}g_{(j-1)n+s}$;
17. End
18. $g_{jn+i} = z_w - \frac{1}{\omega_j}g_{jn+i}$;
19. For $s = 0, \ldots, i - 1$
20. $\beta_{jn+s} = -f_i^Hg_{jn+s}/c_{jn+s}$;

| No. | Iter | Time (s) | True error | Iter | Time (s) | True error |
|-----|------|----------|------------|------|----------|------------|
| 1   | 190  | 0.3823   | 8.6940 x 10⁻⁸ | 243  | 0.2611   | 2.4796 x 10⁻⁸ |
### Table 5.3
Experimental results run on the data in Table 5.2. "−" means no convergence within 10N k-iterations.

| No. | n  | ML(n)BiCGStab (Alg. 5.1 in [27]) | ML(n)BiCGStab |
|-----|----|----------------------------------|----------------|
|     |    | Iter | Time [s] | True error | Iter | Time [s] | True error |
| 1   | 8  | 179  | 0.2271  | 8.4558 × 10⁻⁸ | 184  | 0.2283  | 8.1975 × 10⁻⁸ |
| 16  |    | 189  | 0.3422  | 8.8224 × 10⁻⁸ | 189  | 0.3254  | 9.3304 × 10⁻⁸ |
| 2   | 8  | 3,662 | 4.0794  | 9.6577 × 10⁻⁸ | 2,913 | 3.1262  | 9.9816 × 10⁻⁸ |
| 16  |    | 2,473 | 4.2875  | 3.8774 × 10⁻⁸ | 2,382 | 3.7026  | 8.1546 × 10⁻⁸ |
| 3   | 8  | 7,288 | 8.0662  | 6.9185 × 10⁻⁸ | 6,933 | 6.9870  | 6.0790 × 10⁻⁸ |
| 16  |    | 4,108 | 6.7744  | 2.2397 × 10⁻⁶ | 4,224 | 6.5421  | 9.7019 × 10⁻⁸ |
| 4   | 8  | −−    | −−      | −−          | 38,054| 98.2040 | 9.6758 × 10⁻⁸ |
| 16  |    | 15,895| 49.0668 | 8.9919 × 10⁻⁸ | 16,289| 48.1862 | 9.6599 × 10⁻⁸ |
| 5   | 8  | −−    | −−      | −−          |      |         |            |
| 16  |    | 24,388| 31.4974 | 9.9962 × 10⁻⁸ | 23,134| 27.7806 | 7.4456 × 10⁻⁸ |
| 6   | 8  | 2,916 | 5.6495  | 8.9420 × 10⁻⁸ | 2,900 | 5.5693  | 9.0846 × 10⁻⁸ |
| 16  |    | 2,670 | 6.3544  | 9.0912 × 10⁻⁸ | 2,574 | 5.9208  | 9.6812 × 10⁻⁸ |
| 7   | 8  | 2,361 | 2.1554  | 9.5836 × 10⁻⁸ | 2,234 | 1.9393  | 5.3475 × 10⁻⁸ |
| 16  |    | 2,554 | 3.3563  | 7.9615 × 10⁻⁸ | 2,190 | 2.6315  | 7.6172 × 10⁻⁸ |
| 8   | 8  | 324   | 1.4897  | 7.6087 × 10⁻⁸ | 327  | 1.5177  | 3.1248 × 10⁻⁸ |
| 16  |    | 331   | 1.5813  | 8.3862 × 10⁻⁸ | 328  | 1.5712  | 7.4566 × 10⁻⁸ |
| 9   | 8  | 13,334| 23.6233 | 9.7577 × 10⁻⁸ | 9,138 | 15.8272 | 9.1668 × 10⁻⁸ |
| 16  |    | 1,459 | 3.0690  | 9.5718 × 10⁻⁸ | 1,897 | 3.8690  | 8.9256 × 10⁻⁸ |
| 10  | 8  | 8,017 | 17.6347 | 1.8724 × 10⁻⁸ | 7,934 | 17.3321 | 1.5846 × 10⁻⁸ |
| 16  |    | 5,417 | 16.3895 | 9.7412 × 10⁻⁸ | 5,202 | 21.2771 | 4.7796 × 10⁻⁸ |
| 11  | 8  | −−    | −−      | −−          |      |         |            |
| 16  |    | 64    | −−      | −−          | 19,723| 91.8874 | 6.0479 × 10⁻⁸ |
| 12  | 8  | 5,750 | 2.4522  | 9.2938 × 10⁻⁸ | 6,928 | 2.8840  | 8.9248 × 10⁻⁸ |
| 16  |    | 4,554 | 2.7911  | 9.8418 × 10⁻⁸ | 4,254 | 2.4498  | 9.9897 × 10⁻⁸ |
| 13  | 8  | 11,819| 43.8388 | 9.8546 × 10⁻⁸ | 11,807| 43.4337 | 8.1438 × 10⁻⁸ |
| 16  |    | 4,440 | 25.6544 | 1.6265 × 10⁻⁸ | 4,307 | 17.8434 | 8.1273 × 10⁻⁸ |
| 14  | 8  | 10    | 0.0241  | 4.8088 × 10⁻⁸ | 10   | 0.0204  | 4.8088 × 10⁻⁸ |
| 16  |    | 10    | 0.0221  | 7.5853 × 10⁻⁸ | 10   | 0.0172  | 7.5853 × 10⁻⁸ |
| 15  | 8  | 223   | 0.2419  | 8.4699 × 10⁻⁸ | 227  | 0.2368  | 7.6142 × 10⁻⁸ |
| 16  |    | 219   | 0.3432  | 7.7560 × 10⁻⁸ | 203  | 0.3175  | 9.3996 × 10⁻⁸ |
| 16  |    | 32    | 1.674   | 5.2264 × 10⁻⁸ | 1,551| 5.0532  | 8.8966 × 10⁻⁸ |
| 17  | 8  | −−    | −−      | −−          |      |         |            |
| 16  |    | 29,202| 717.8035| 9.7427 × 10⁻⁸ | 33,609| 757.6161| 9.6732 × 10⁻⁸ |
| 18  | 8  | 20    | 0.0250  | 5.4243 × 10⁻⁸ | 20   | 0.0276  | 5.4243 × 10⁻⁸ |
| 16  |    | 21    | 0.0286  | 2.0159 × 10⁻⁸ | 21   | 0.0316  | 2.0159 × 10⁻⁸ |
| 19  | 8  | 25    | 0.0987  | 1.6317 × 10⁻⁸ | 25   | 0.0511  | 1.6317 × 10⁻⁸ |
| 16  |    | 1,156 | 2.9014  | 8.9851 × 10⁻⁸ | 21   | 0.0405  | 8.9851 × 10⁻⁸ |
| 20  | 8  | 1,136 | 2.8012  | 8.9865 × 10⁻⁸ | 1,169| 3.4974  | 8.8665 × 10⁻⁸ |
| 16  |    | 617   | 1.6847  | 9.9117 × 10⁻⁸ | 602  | 1.6106  | 9.5743 × 10⁻⁸ |

21. \( g_{jn+i} = g_{jn+i} + \beta_{jn+i} g_{jn+i} \)
22. End
23. Else
24. \( \beta_{jn+i} = -f_1^H r_{jn+i}/c_{jn} \)
25. \( g_{jn+i} = r_{jn+i} + \beta_{jn+i} g_{jn} \)
26. For \( s = 1, \ldots, i-1 \)
27. \( \beta_{jn+s} = -f_{s+1}^H g_{jn+s}/c_{jn+s} \)
28. \( g_{jn+i} = g_{jn+i} + \beta_{jn+i} g_{jn+i} \)
29. End
30. End
31. $\hat{g}_{jn+i} = M^{-1}g_{jn+i}; w_{jn+i} = A\hat{g}_{jn+i};$
32. $e_{jn+i} = q_{i+1}^H w_{jn+i};$
33. End
34. $\alpha_{jn+n} = e_{jn+n-1}/e_{jn+n-1};$
35. $x_{jn+n} = x_{jn+n-1} + \alpha_{jn+n}g_{jn+n-1};$
36. $u_{jn+n} = r_{jn+n-1} - \alpha_{jn+n}w_{jn+n-1};$
37. $\hat{u}_{jn+n} = M^{-1}u_{jn+n};$
38. $\omega_{j+1} = (A\hat{u}_{jn+n})^H u_{jn+n}/\|A\hat{u}_{jn+n}\|_2^2;$
39. $x_{jn+n} = x_{jn+n} + \omega_{j+1}u_{jn+n};$
40. $r_{jn+n} = -\omega_{j+1}A\hat{u}_{jn+n} + u_{jn+n};$
41. $e_{jn+n} = q_i^H r_{jn+n};$
42. $\beta_{(j-1)n+n} = -e_{jn+n}/e_{(j-1)n+n}; \% \beta_{(j-1)n+n} = -\omega_{j+1}\beta_{(j-1)n+n}$
43. $z_w = r_{jn+n} + \beta_{(j-1)n+n}w_{(j-1)n+n};$
44. $g_{jn+n} = \beta_{(j-1)n+n}g_{(j-1)n+n};$
45. For $s = 1, \ldots, n - 1$
46. $\bar{\beta}_{jn+s} = -e_{jn+s}/e_{(j-1)n+s}; \% \bar{\beta}_{jn+s} = -\omega_{j+1}\bar{\beta}_{s+jn}$
47. $z_w = z_w + \bar{\beta}_{jn+s} w_{jn+s};$
48. $g_{jn+n} = g_{jn+n} + \bar{\beta}_{jn+s} g_{jn+s};$
49. End
50. $g_{jn+n} = z_w - \frac{1}{\omega_{j+1}}g_{jn+n}; \hat{g}_{jn+n} = M^{-1}g_{jn+n};$
51. $w_{jn+n} = A\hat{g}_{jn+n}; c_{jn+n} = q_i^H w_{jn+n};$
52. End

### Table 5.4

Experimental results run on the data in Table 5.7 “-” means no convergence within 10N k-iterations.

| No. | n   | iter | Time (s) | True error | No. | n   | iter | Time (s) | True error |
|-----|-----|------|----------|------------|-----|-----|------|----------|------------|
| 1   | 8   | 182  | 0.2490   | 7.0962 x 10^-8 | 11  | 8   | 6     | 6.495    | 2.9844 x 10^-8 |
| 2   | 16  | 194  | 0.4072   | 8.6571 x 10^-8 | 64  | 6   | 12.292 | 48.8011  | 7.3525 x 10^-8 |
| 3   | 8   | 4,110| 5.1773   | 9.1479 x 10^-8 | 12  | 8   | 6     | 6.495    | 2.9844 x 10^-8 |
| 4   | 16  | 2,887| 5.9607   | 8.6763 x 10^-8 | 10  | 5   | 3.753  | 9.2844 x 10^-8 |
| 5   | 8   | 2,984| 5.7263   | 6.2621 x 10^-8 | 13  | 8   | 12.292 | 48.8011  | 7.3525 x 10^-8 |
| 6   | 16  | 3,464| 7.0040   | 6.7311 x 10^-8 | 16  | 8   | 4.681  | 30.0199  | 7.3730 x 10^-8 |
| 7   | 8   | 34,329| 93.4919  | 8.9878 x 10^-8 | 14  | 8   | 13    | 0.0291   | 4.7338 x 10^-8 |
| 8   | 16  | 16,244| 54.7066  | 8.4285 x 10^-8 | 16  | 14  | 0.0308 | 6.6326 x 10^-8 |
| 9   | 8   | 33,470| 30.1092  | 7.9861 x 10^-8 | 15  | 8   | 260   | 0.9341   | 6.1463 x 10^-8 |
| 10  | 8   | 11,147| 27.1564  | 9.8394 x 10^-8 | 16  | 225  | 0.5702 | 7.4062 x 10^-8 |
| 11  | 16  | 2,915 | 5.9639   | 9.7100 x 10^-8 | 16  | 8   | -     | -        | -          |
| 12  | 16  | 3,203 | 8.4778   | 8.2899 x 10^-8 | 16  | 8   | -     | -        | -          |
| 13  | 8   | 2,278 | 2.2367   | 5.8257 x 10^-8 | 17  | 8   | -     | -        | -          |
| 14  | 16  | 3,307 | 3.5007   | 7.3368 x 10^-8 | 16  | 8   | -     | -        | -          |
| 15  | 8   | 326   | 1.5374   | 4.7310 x 10^-8 | 20  | 8   | 0.0268 | 1.7960 x 10^-8 |
| 16  | 16  | 331   | 1.6738   | 9.1445 x 10^-8 | 16  | 19  | 0.0297 | 8.8714 x 10^-8 |
| 17  | 8   | 12,066| 22.2520  | 9.5102 x 10^-8 | 19  | 8   | 0.0399 | 9.1399 x 10^-8 |
| 18  | 16  | 1,614 | 3.7977   | 9.7401 x 10^-8 | 16  | 24  | 0.0458 | 6.4031 x 10^-8 |
| 19  | 8   | 7,490 | 17.8209  | 1.2933 x 10^-7 | 20  | 8   | 1.128 | 2.8335  | 9.6613 x 10^-8 |
| 20  | 16  | 5,100 | 16.3960  | 1.3307 x 10^-7 | 16  | 661 | 1.9800 | 9.2218 x 10^-8 |
Matlab code of Algorithm \texttt{BiCGStab}.

1. function \([x, \text{err, iter, flag}] = \text{mlbicgstabt}(A, x, b, Q, M, \text{max\_it}, \text{tol}, \kappa)\)
2. \%
3. \%
4. \%
5. \%
6. \%
7. \%
8. \%
9. \%
10. \%
11. \%
12. \%
13. \%
14. \%
15. \%
16. \%
17. \%
18. \%
19. \%
20. \%
21. \%
22. \%
23. \%
24. \%
25. \%
26. \%
27. \%
28. \%
29. \%
30. \%
31. \%
32. \%
33. \%
34. \%
35. \%
36. \%
37. \%
38. \%
39. \%
40. \%
41. \%
42. \%
43. \%
44. \%
45. \%
46. \%
47. \%
48. end
\[ G(:, i + 1) = W(:, i + 1) - G(:, i + 1)/\omega; \]

for \( s = 0 : i - 1 \)
\[ \beta = -F(:, s + 1)'*G(:, i + 1)/c(s + 1); \]
\[ G(:, i + 1) = G(:, i + 1) + \beta*G(:, s + 1); \]
\end{flushleft}

\[ G(:, i + 1) = W(:, i + 1) - G(:, i + 1)/\omega; \]

for \( s = 1 : i - 1 \)
\[ \beta = -F(:, s + 1)'*G(:, i + 1)/c(s + 1); \]
\[ G(:, i + 1) = G(:, i + 1) + \beta*G(:, s + 1); \]
\end{flushleft}

\[ g_{\mathcal{H}} = M\setminus G(:, i + 1); \quad W(:, i + 1) = A* g_{\mathcal{H}}; \]
\[ c(i + 1) = Q(:, i + 1)'*W(:, i + 1); \]
if \( c(i + 1) == 0, \ flag = -1; \ return,\ end \]
\[ alpha = c/c(n); \quad x = x + alpha* g_{\mathcal{H}}; \quad r = r - alpha*W(:, n); \]
\[ err = \text{norm}(r)/\text{bnrm2}; \]
if \( err < \text{tol}, \ flag = 0; \ iter = iter + 1; \ return,\ end \]
\[ g_{\mathcal{H}} = M\setminus r; \quad z = A* g_{\mathcal{H}}; \quad \omega = \alpha* r; \]
if \(\omega == 0, \ flag = -1; \ return, \ end \]
\[ rho = \alpha* r; \quad \omega = \rho/\omega; \]
if \(\kappa > 0 \)
\[ rho = \rho/\text{norm}(z)*\text{norm}(r); \quad \omega = \rho/\omega; \]
if \(\text{abs}\omega < \kappa \) & \(\text{abs}\omega == 0 \)
\[ \omega = \omega * \kappa/\text{abs}\omega; \]
\end{flushleft}

\[ e = Q(:, 1)'*r; \quad \beta = -e/c(1); \]
\[ W(:, 1) = r + \beta*W(:, 1); \quad G(:, 1) = \beta*G(:, 1); \]
for \( s = 1 : n - 1 \)
\[ \beta = -Q(:, s + 1)'*W(:, 1)/c(s + 1); \]
\[ W(:, 1) = W(:, 1) + \beta*W(:, s + 1); \]
\[ G(:, 1) = G(:, 1) + \beta*G(:, s + 1); \]
\end{flushleft}

\[ G(:, 1) = W(:, 1) - G(:, 1)/\omega; \quad g_{\mathcal{H}} = M\setminus G(:, 1); \]
\[ W(:, 1) = A* g_{\mathcal{H}}; \quad c(1) = Q(:, 1)'*W(:, 1); \]
if \( c(1) == 0, \ flag = -1; \ return, \ end \]
\end{flushleft}

\section*{REFERENCES}

\bibitem{Du1990}
L. Du, T. Sogabe, B. Yu, Y. Yamamoto, S.-L. Zhang, \textit{A block IDR(s) method for nonsym-}
metric linear systems with multiple right-hand sides, J. Comput. Appl. Math. 235(2011), no. 14, 4095-4106.

[2] R. Fletcher, Conjugate gradient methods for indefinite systems, volume 506 of Lecture Notes Math., pages 73-89. Springer-Verlag, Berlin-Heidelberg-New York, 1976.

[3] R. Freund, M. Gutknecht and N. Nachtigal, An implementation of the look-ahead Lanczos algorithm for non-Hermitian matrices, SIAM J. Sci. Comput., 14(1993), pp. 137-158.

[4] M. Gijzen and P. Sonneveld, Algorithm 913: an elegant IDR(s) variant that efficiently exploits bi-orthogonality properties, ACM Trans. Math. Software, 38(2011), pp. 5:1–5:19.

[5] M. H. Gutknecht, A completed theory of the unsymmetric Lanczos process and related algorithms. Part I., SIAM J. Matrix Anal. Appl., 13(1992), pp.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] ——, IDR-type solvers for nonsymmetric linear systems of equations, Acta Numerica, 6 (1997), pp. 271-397.

[9] ——, IDR Explained, ETNA 36 (2010), 126–148.

[10] Martin H. Gutknecht and Jens-Peter M. Zemke, Eigenvalue computations based on IDR, Bericht 145, TUHH, Institute of Numerical Simulation, May 2010.

[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] D. Loher, Reliable nonsymmetric block Lanczos algorithms, Ph.D. thesis, Swiss Federal Institute of Technology, Zurich, 2006.

[14] A. Neumaier, Oral presentation at the Oberwolfach meeting “Numerical Linear Algebra”, Oberwolfach, April 1994.

[15] B. N. Parlett, D. R. Taylor, and Z. A. Liu, A look-ahead Lanczos algorithm for unsymmetric matrices, Math. Comp., 44(1985), pp.105-124.

[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] 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.

[18] G. L. Sleijpen and D. R. Fokkema, BiCGSTAB(l) for linear equations involving nonsymmetric matrices with complex spectrum, ETNA, 1:11-32, 1993.

[19] G.L.G. Sleijpen and M. B. van Gijzen, Exploiting BiCGStab(l) strategies to induce dimension reduction, SIAM J. Sci. Comput. 32(2010), no. 5, 2687-2709.

[20] G. L. G. Sleijpen and H. A. van der Vorst, Reliable updated residuals in hybrid Bi-CG methods, Computing, 56 (1996), pp. 141–163.

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

[22] P. Sonneveld, On the convergence behavior of IDR(s) and related methods, SIAM J. Sci. Comput., 34(5), A2576-A2598.

[23] P. Sonneveld and M. van Gijzen, IDR(s): a family of simple and fast algorithms for solving large nonsymmetric systems of linear equations, SIAM J. Sci. Comput. 31(2008), no. 2, pp. 1035-1062.

[24] M. Tanio and M. Sugihara, GBi-CGStab(s,1): IDR(s) with higher-order stabilization polynomials, J. Comput. Appl. Math. 235(2010), no. 3, 765-784.

[25] 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.

[26] P. Wesseling and P. Sonneveld, Numerical experiments with a multiple grid and a preconditioned Lanczos type method, Lecture Notes in Mathematics, vol. 771, pp.543-562, Springer Verlag, Berlin, Heidelberg, New York, 1980.

[27] M. Yeung, ML(n)BiCGStab: Reformulation, Analysis and Implementation, Numer. Math. Theor. Meth. Appl. 5 (2012), pp. 447-492.

[28] M. Yeung, An introduction to ML(n)BiCGStab, available at http://arxiv.org/abs/1106.3678

[29] M. Yeung and D. Boley, Transpose-free multiple Lanczos and its application in Padé appl-
proximation, Journal of Computational and Applied Mathematics, Vol 177/1 pp. 101-127, 2005.

[30] 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.

[31] 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.