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

MAN-CHUNG YEUNG*

Abstract. The 1980 IDR method[28] plays an important role in the history of Krylov subspace methods. It started the research of transpose-free Krylov subspace methods. The ML(\(n\))BiCGStab method[29] is one of such methods. In this paper, we present a new ML(\(n\))BiCGStab variant that involves A-transpose in its implementation. Comparison of this new algorithm with the existing ML(\(n\))BiCGStab algorithms and some other Krylov subspace 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

\[ \mathbf{A}\mathbf{x} = \mathbf{b} \tag{1.1} \]

where \(\mathbf{A} \in \mathbb{C}^{N \times N}\) and \(\mathbf{b} \in \mathbb{C}^{N}\). It was introduced by Yeung and Chan[33] in 1999 and its algorithms were recently reformulated by Yeung[29]. ML(\(n\))BiCGStab is a natural generalization of BiCGStab[27], 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[23] and van der Vorst[27] to construct 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[19]. In fact, it is a BiCGStab when \(n = 1\) and becomes a GMRES/FOM when \(n = N\) (see [29, 30]). 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 [29, 33].

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 [33] is. Numerical experiments, however, showed that this new ML(\(n\))BiCG was unstable and weaker than the standard BiCG. Moreover, in [32], Yeung and Boley derived a SVLP from an one-sided multiple starting band Lanczos procedure (MSLP) with \(n\) left-starting and \(m\) right-starting vectors. From their experiments about the multi-input multi-output time-invariant linear dynamical systems, they observed that SVLP is more stable than the both-sided MSLP when \(m \neq n\). These two comparing examples hint that, when \(m \neq n\), a stable multiple starting procedure with A-transpose may come from a modification of a SVLP. In this paper, we make a first attempt in this direction by introducing A-transpose into ML(\(n\))BiCGStab. We call the resulting algorithm ML(\(n\))BiCGStabt[30, 31], standing for ML(\(n\))BiCGStab with transpose. We remark that \(\mathbf{A}^H\) has been used in [35] to

*Dept. 3036, 1000 East University Avenue, Laramie, WY 82071. E-mail: myeung@uwyo.edu.
improve the parallelism of GPBiCG($m,l$)[4]. Here we want to use $A^H$ to enhance the numerical stability of ML($n$)BiCGStab.

There exist two ML($n$)BiCGStab algorithms, labeled as Algorithms 4.1 and 5.1 respectively in [29], 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($n$)BiCGStab 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 BiCGStab2[7], BiCGStab($l$)[20], GPBi-CG[34], IDR($s$)[25, 5], IDRstab[21], GPBiCG($m,l$)[4], and GBi-CGSTAB($s,l$)[26]. Related articles include [1, 10, 11, 24].

The outline of the paper is as follows. In §2, index functions in [32] are introduced. They are helpful in the construction of a ML($n$)BiCGStab algorithm. In §3, we present the ML($n$)BiCG algorithm from [33]. The derivation of every ML($n$)BiCGStab algorithm is based on it. In §4, we introduce the ML($n$)BiCGStab 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

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

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

\[
k = jn + i
\]

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

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

3. ML($n$)BiCG. Analogously to the derivation of BiCGStab from BiCG, the ML($n$)BiCGStab algorithms[29] were derived from a BiCG-like algorithm named ML($n$)BiCG, which was built upon an one-sided 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 [33].

Consider the solution of (1.1). Throughout the paper we do not assume the coefficient matrix $A$ is nonsingular. In [29], we proved that ML($n$)BiCG/ML($n$)BiCGStab can solve a singular system almost surely provided that the underlying 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 [33].

**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 $r_0 = b - Ax_0$ and set $p_1 = q_1$, $x_0 = r_0$.
3. For $k = 1, 2, \ldots$, until convergence:
   4. $\alpha_k = p^H_k r_{k-1}/p^H_k A\tilde{g}_{k-1}$;
   5. $\tilde{x}_k = \tilde{x}_{k-1} + \alpha_k \tilde{g}_{k-1}$;
   6. $r_k = r_{k-1} - \alpha_k A\tilde{g}_{k-1}$;
   7. For $s = \max(k - n, 0), \ldots, k - 1$
      8. $\beta_s^{(k)} = -p^H_{s+1} A (r_k + \sum_{i=\max(k-n, 0)}^{s-1} \beta_i^{(k)} \tilde{g}_i) / p^H_{s+1} A\tilde{g}_s$;
   9. End
   10. $\tilde{g}_k = r_k + \sum_{s=\max(k-n, 0)}^{k-1} \beta_s^{(k)} \tilde{g}_s$;
   11. Compute $p_{k+1}$ according to (3.1)
   12. End

This ML($n$)BiCGStab 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

$$B_k = \text{span} \{ p_1, p_2, \ldots, p_k \} = \text{span} \{ q_1, q_2, \ldots, q_n \}$$

where $Q \equiv\{ q_1, q_2, \ldots, q_n \}$, $K_0(M, v) = \{ 0 \}$ and $K_k(M, v) \equiv \text{span} \{ v, Mv, \ldots, M^{k-1}v \}$

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 an one-sided Lanczos algorithm (see [9, 18]).

It can be shown that the quantities of ML($n$)BiCG satisfy the properties (see [29])

(a) $\tilde{x}_k \in \tilde{x}_0 + K_k(A, \tilde{r}_0)$, $\tilde{r}_k \in \tilde{r}_0 + A\tilde{x}_k(A, \tilde{r}_0)$.
(b) $\tilde{x}_k \perp \text{span} \{ p_1, p_2, \ldots, p_k \}$ and $\tilde{r}_k \perp p_{k+1}$.
(c) $A\tilde{g}_k \perp \text{span} \{ p_1, p_2, \ldots, p_k \}$ and $A\tilde{g}_k \perp p_{k+1}$.

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 $p^H r$ and $p^H A\tilde{g}$ in BiCG can be replaced by inner products of the forms $q^H \psi(A) r$ 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 [29], 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

$$\begin{align*}
r_k &= \phi_{g_n(k)+1}(A) \tilde{r}_k, \\
u_k &= \phi_{g_n(k)}(A) \tilde{r}_k, \\
g_k &= \phi_{g_n(k)+1}(A) \tilde{g}_k, \\
d_k &= -\omega_{g_n(k)+1} A \phi_{g_n(k)}(A) \tilde{g}_k, \\
w_k &= A\tilde{g}_k
\end{align*}$$

(4.1)
for $k > 0$, and those in Algorithm 5.1 defined as

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

for $k > 0$. When $k = 0$, both algorithms set

$$
\mathbf{r}_0 = \mathbf{r}_0^0 \text{ and } \mathbf{g}_0 = \mathbf{g}_0^0.
$$

Here $\mathbf{r}_k$ is the residual of the $k$th approximate solution $\mathbf{x}_k$. Numerical experiments in [29] 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

$$
\mathbf{b} - \mathbf{A}\mathbf{x}_k = \mathbf{b} - \mathbf{A}(\mathbf{x}_{k-1} + \alpha_k \mathbf{g}_{k-1}) = (\mathbf{b} - \mathbf{A}\mathbf{x}_{k-1}) - \alpha_k \mathbf{A}\mathbf{g}_{k-1}.
$$

In Algorithm 4.1, $\mathbf{w}_k$ is updated by $\mathbf{w}_k = \mathbf{A}\mathbf{g}_k$ (as it is defined in (4.1)) 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 is closer to (4.3)\(^1\). As a result, the residual $\mathbf{r}_k$ computed by Algorithm 4.1 is generally closer to the true residual (4.3) than the $\mathbf{r}_k$ computed by Algorithm 5.1. 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 $\text{ML}(n)\text{BiCGStab}$.

The derivation of Algorithm 5.1 in [29] was divided into several stages, starting from $\text{ML}(n)\text{BiCG}$. The following is a copy of its Derivation Stage #8 (DS #8) which is a list of equations that the quantities in (4.2) satisfy.

**Derivation Stage #8 in [29].**

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

\(^1\)There is a similar comment on BiCGStab(l) [20, p.27] when compared to BiCGStab2[7].
Adding (4.5) and $r$ is available. Recalling that $A/n$ vector multiplications as low as $1 + 1$

imize the 2-norm of $\omega$.

Compute $x_1$. Choose an initial guess $\alpha$.

For $s = g_n(k)n, \ldots, k - 1$

16. $\beta^{(k)} = q_{r_n(s+1)}^H (r_k - \omega g_n(k+1) \sum_{l=g_n(k)n}^{s-1} \beta^{(k)} w_t) / \omega g_n(k+1) q_{r_n(s+1)}^H w_s$

17. End

18. $g_s = r_k - \omega g_n(k+1) \sum_{s=g_n(k)n}^{k-1} \beta^{(k)} w_s + \sum_{s=g_n(k)n}^{k-1} \beta^{(k)} g_s$

19. End

20. End

According to (4.2), the equation in Line 9 can be rewritten as

$$
\beta^{(k)} = -q_{r_n(s+1)}^H A \left( r_k + \sum_{t=max(k-n,0)}^{s-1} \beta^{(k)} g_t \right) / q_{r_n(s+1)}^H w_s
$$

$$
= -q_{r_n(s+1)}^H A \left( r_k + \sum_{t=max(k-n,0)}^{s-1} \beta^{(k)} g_t \right) / q_{r_n(s+1)}^H w_s
$$

it is because of the $A$ in front of the parentheses, we can not update $w_k$ by $w_k = Ag_k$
in every $k$-iteration in Algorithm 5.1 while keeping the average number of matrix-vector
multiplications as low as $1 + 1/n$ per $k$-iteration. If, however, the vector $f_{r_n(s+1)} = A^H q_{r_n(s+1)}$ is available, then Line 9 will become

$$
\beta^{(k)} = -q_{r_n(s+1)}^H A \left( r_k + \sum_{t=max(k-n,0)}^{s-1} \beta^{(k)} g_t \right) / q_{r_n(s+1)}^H w_s
$$

(4.4)

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

Replace Line 9 in DS #8 with (4.4) and suppose

$$
F \equiv A^H Q = [A^H q_1, A^H q_2, \ldots, A^H q_n]
$$
is available.Recalling that $r_k$ is the residual of $x_k$, to be consistent with Lines 4, 13 and 14, we update the approximate solution $x_k$ as

$$
x_k = \begin{cases} 
  x_{k-1} + \alpha g_{k-1}, & \text{if } r_n(k) < n \\
  \omega g_n(k+1) u_k + x_{k-1} + \alpha g_{k-1}, & \text{if } r_n(k) = n.
\end{cases}
$$

(4.5)

Adding (4.5) and $w_k = Ag_k$ to DS #8, simplifying its operations appropriately, we then arrive at the following algorithm. The free parameter $\omega g_n(k+1)$ is chosen to minimize the 2-norm of $r_k$.

**Algorithm 4.1.** **ML($n$)BiCGStab without preconditioning**

1. Choose an initial guess $x_0$ and $n$ vectors $q_1, q_2, \ldots, q_n$.
2. Compute $[I_1, \ldots, I_{n-1}] = A^H q_1, \ldots, q_{n-1}$.
3. Compute $r_0 = b - Ax_0$ and $g_0 = r_0$, $w_0 = Ag_0$, $c_0 = q_1^H w_0$, $\omega_0 = 1$.
4. For $k = 1, 2, \ldots$, until convergence:
5. $\alpha_k = q_{r_n(k)}^H r_{k-1} / \epsilon_{k-1}$
6. If $r_n(k) < n$
7. $x_k = x_{k-1} + \alpha_k g_{k-1}; \quad r_k = r_{k-1} - \alpha_k w_{k-1};$
8. $z_w = r_k; \quad g_k = 0;$
9. For $s = \max(k-n, 0), \ldots, g_n(k)n - 1$
10. $\beta^{(k)} = -q_n^{(s+1)} z_w / c_s; \quad \% \beta^{(k)} = -\omega_n(k+1) \beta^{(k)}$
11. $z_w = z_w + \beta^{(k)} w_s;$
12. $g_k = g_k + \beta^{(k)} g_s;$
13. End
14. $g_k = z_w - \frac{1}{\omega_n(k+1)} g_k;$
15. For $s = g_n(k)n, \ldots, k - 1$
16. $\beta^{(k)} = -r_n^{(s+1)} g_k / c_s; \quad g_k = g_k + \beta^{(k)} g_s;$
17. End
18. Else
19. $x_k = x_{k-1} + \alpha_k g_{k-1};$
20. $u_k = r_k - \alpha_k w_{k-1};$
21. $\omega_n(k+1) = (A u_k)^H u_k / \| A u_k \|^2_2;$
22. $x_k = x_k + \omega_n(k+1) u_k; \quad r_k = -\omega_n(k+1) A u_k + u_k;$
23. $z_w = r_k; \quad g_k = 0;$
24. For $s = g_n(k)n, \ldots, k - 1$
25. $\beta^{(k)} = -q_n^{(s+1)} z_w / c_s; \quad \% \beta^{(k)} = -\omega_n(k+1) \beta^{(k)}$
26. $z_w = z_w + \beta^{(k)} w_s;$
27. $g_k = g_k + \beta^{(k)} g_s;$
28. End
29. $g_k = z_w - \frac{1}{\omega_n(k+1)} g_k;$
30. End
31. $w_k = A g_k; \quad c_k = q_n^{(s+1)} w_k;$
32. 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 DS #8 on improving the stability of Algorithm 5.1 in [29]. The stability of updates of the type (4.6) has been studied in detail by Neumaier[16] and Sleijpen and van der Vorst[22].

We remark that (i) the above 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 7.1) 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 [29], the computational cost of ML($n$)BiCGStabt is slightly cheaper.

Theoretically, it can be guaranteed that an exact breakdown in the above Algorithm 4.1 is almost impossible (see [29] for a detailed analysis). The algorithm, however, can encounter a near breakdown in its implementation. The divisors in the algorithm are $c_k, \| A u_k \|_2$ and $\omega_n(k+1)$. If $\| A u_k \|_2 \approx 0$, then $u_k \approx 0$ and the $x_k$ in Line 20 is an approximate solution. When $\omega_n(k+1) \approx 0$, we can add some
Since the quantities of ML($n$)BiCGStab are defined exactly the same as those of Algorithm 5.1 in [29], ML($n$)BiCGStab 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 $S_l$ and $W_l$ the $l \times l$ leading principal submatrices of $S_\nu$ and $W_\nu$ respectively (Joubert[12, 13] called these the moment matrices). Then some facts about ML($n$)BiCGStab (the above Algorithm 4.1) are summarized as follows.

**Proposition 4.2.** [29, Prop. 5.1] In infinite precision arithmetic, if $\prod_{l=1}^{\nu} \det(S_l) \det(W_l) \neq 0$, $\omega_{\nu,(k+1)} \neq 0$ and $1/\omega_{\nu,(k+1)} \not\in \sigma(A)$ for $1 \leq k \leq \nu-1$, where $\sigma(A)$ is the spectrum of $A$, then ML($n$)BiCGStab 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 (1.1). Moreover, the computed quantities satisfy

(a) $x_k \in x_0 + \text{span}\{r_0, A r_0, \ldots, A^{\nu-1} r_0\}$ and $r_k = b - A x_k \in r_0 + \text{span}\{A_0, A^2 r_0, \ldots, A^{\nu-1} r_0\}$ for $1 \leq k \leq \nu - 1$.

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

(c) $r_k \perp \text{span}\{q_1, q_2, \ldots, q_{r_0(k)}\}$ and $r_k \not\perp q_{r_0(k)+1}$ for $1 \leq k \leq \nu - 1$ with $r_0(k) < n$; $r_k \not\perp q_k$ for $1 \leq k \leq \nu - 1$ with $r_0(k) = n$.

(d) $u_k \perp \text{span}\{q_1, q_2, \ldots, q_n\}$ for $1 \leq k \leq \nu$ with $r_0(k) < n$; $u_k \not\perp q_k$ for $1 \leq k \leq \nu - 1$ with $r_0(k) = n$.

(e) $A g_k \perp \text{span}\{q_1, q_2, \ldots, q_n\}$ and $A g_k \not\perp q_{r_0(k)+1}$ for $1 \leq k \leq \nu - 1$ with $r_0(k) < n$; $A g_k \not\perp q_k$ for $1 \leq k \leq \nu - 1$ with $r_0(k) = n$.

In §6.2 of [29], relations of Algorithm 5.1 in [29] to some existing methods were presented. The same arguments applied to ML($n$)BiCGStab 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)BiCGStab 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 an $i$-loop and an $j$-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 now compare ML(n)BiCGStabt with BiCG, BiCGStab, GMRES-DR[15] and two algorithms of ML(n)BiCGStab: Algorithms 4.1 and 5.1 in [29]. The experimental results are shown in the following. All the test data were downloaded from The University of Florida Sparse Matrix Collection², and computations were 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 $\|r_k\|_2/\|b\|_2 < 10^{-7}$ where $r_k$ was the computed residual, and the Sleijpen-van der Vorst minimization control parameter (see [29]) $\kappa = 0$. As for the shadow vectors, we chose $Q = [r_0, \text{sign}(\text{randn}(N, n - 1))]$. When a data did not provide a right-hand side, we set $b = Ae$ where $e$ is the vector of ones.

We also compared ML(n)BiCGStabt with IDR(s)[5] and found that IDR(s) has the same level of stability with the ML(n)BiCGStab Algorithm 5.1 in [29], but is less stable than ML(n)BiCGStabt. In general, IDR(s) takes about the same number of matrix-vector multiplications(MVs) to converge as the ML(n)BiCGStab algorithms do, but in terms of time, IDR(s) is about 20%−50% faster than the ML(n)BiCGStab algorithms on average. Perhaps, this is due to the fact that about half number of the inner products in IDR(s) are computed simultaneously by solving a triangular system.

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 5.2-5.5. The “True error” column in each table contains the true relative errors $\|b - Ax\|_2/\|b\|_2$ where $x$ is the computed solution output by an algorithm when it converges. Recall that GMRES-DR has two parameters $m$ and $k$ needed to be indicated with $m$ being the size of the underlying Krylov subspace and $k$ the number of approximate eigenvectors saved at the restart. It requires the storage of about the same number of vectors as GMRES(m) to implement. In Table 5.3, we choose $m$ and $k$ so that the storage of GMRES-DR is comparable to that of ML(n)BiCGStab and ML(n)BiCGStabt.

In this experiment, we observe that ML(n)BiCGStabt generally outperforms BiCG, BiCGStab and GMRES-DR in stability, computational time and the number of MVs. Data #11 and #16, however, present to be difficult for ML(n)BiCGStabt and ML(n)BiCGStab, but relatively much easier to GMRES-DR. On the other hand,

²http://www.cise.ufl.edu/research/sparse/matrices/
as an improved version of Algorithm 5.1 in [29], ML(n)BiCGStabt has the same stability as Algorithm 4.1 in [29] and is more stable than Algorithm 5.1.

Example 2. Our experience with the Florida collection has shown that Algorithm 5.1 in [29] is overall a stable algorithm. But still, one can find one or two matrices where it is unstable. For example, consider

1. e40r0100, 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 $E(n) \equiv \|b - Ax\|_2/\|b\|_2$ where $x$ is the computed solution output by an algorithm when it converges. The graphs of $E(n)$ are plotted in Figure 5.1. It can be seen that the computed relative errors $\|r_k\|_2/\|b\|_2$ by Algorithm 5.1 significantly drift away from their exact counterparts. By contrast, however, the computed $\|r_k\|_2/\|b\|_2$ by ML(n)BiCGStabt and Algorithm 4.1 in [29] well approximate their corresponding true relative errors. In this experiment, ML(n)BiCGStabt improves the stability of Algorithm 5.1 in [29] significantly.

![Graphs of E(n) against n. (a) e40r0100. Since all the three algorithms do not converge when 1 ≤ n ≤ 16, we only plot the graphs over the range 17 ≤ n ≤ 100; (b) utm5940.](image)

6. Concluding Remarks. The original motivation of developing ML(n)BiCGStabt was to improve the stability of Algorithm 5.1 in [29]. From our experiments, the improvement can sometimes be significant. Since, however, the two algorithms are 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)BiCGStabt(l) will naturally be carried out. They are clearly different from ML(n)BiCGStabt in structure and thereby we expect different properties that they will have.

7. Appendix. In this section, we present a preconditioned ML(n)BiCGStabt algorithm together with its Matlab code.
Table 5.1

A group of data selected from the Florida collection. Data #13 contains multiple right-hand sides, and we selected the 45th in our experiments.

<table>
<thead>
<tr>
<th>No.</th>
<th>Matrix name</th>
<th>Group name</th>
<th>Size</th>
<th>Nonzeros</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>rb50000</td>
<td>Bai</td>
<td>5,000</td>
<td>29,600</td>
</tr>
<tr>
<td>2</td>
<td>sherman3</td>
<td>HB</td>
<td>5,905</td>
<td>20,033</td>
</tr>
<tr>
<td>3</td>
<td>olm5000</td>
<td>Bai</td>
<td>5,000</td>
<td>19,996</td>
</tr>
<tr>
<td>4</td>
<td>cavity19</td>
<td>Drivcav</td>
<td>4,562</td>
<td>131,735</td>
</tr>
<tr>
<td>5</td>
<td>tols4000</td>
<td>Bai</td>
<td>4,000</td>
<td>8,784</td>
</tr>
<tr>
<td>6</td>
<td>ex31</td>
<td>Fidap</td>
<td>3,909</td>
<td>91,223</td>
</tr>
<tr>
<td>7</td>
<td>sherman5</td>
<td>HB</td>
<td>3,312</td>
<td>20,793</td>
</tr>
<tr>
<td>8</td>
<td>raelsky2</td>
<td>Simon</td>
<td>3,242</td>
<td>293,551</td>
</tr>
<tr>
<td>9</td>
<td>garon1</td>
<td>Garon</td>
<td>3,175</td>
<td>84,723</td>
</tr>
<tr>
<td>10</td>
<td>utm5940</td>
<td>Tokamak</td>
<td>5,940</td>
<td>88,842</td>
</tr>
<tr>
<td>11</td>
<td>Chebyshev3</td>
<td>Muite</td>
<td>4,101</td>
<td>36,879</td>
</tr>
<tr>
<td>12</td>
<td>pores2</td>
<td>HB</td>
<td>1,224</td>
<td>9,613</td>
</tr>
<tr>
<td>13</td>
<td>tsopf_rs_b162_c1</td>
<td>Tsopf</td>
<td>5,374</td>
<td>205,399</td>
</tr>
<tr>
<td>14</td>
<td>rw5151</td>
<td>Bai</td>
<td>5,151</td>
<td>20,199</td>
</tr>
<tr>
<td>15</td>
<td>circuit2</td>
<td>Bomhof</td>
<td>4,510</td>
<td>21,199</td>
</tr>
<tr>
<td>16</td>
<td>viscoplastic1</td>
<td>Quaglino</td>
<td>4,326</td>
<td>61,166</td>
</tr>
<tr>
<td>17</td>
<td>heart1</td>
<td>Norris</td>
<td>4,557</td>
<td>1,385,717</td>
</tr>
<tr>
<td>18</td>
<td>cage9</td>
<td>vanHenkelum</td>
<td>3,534</td>
<td>41,594</td>
</tr>
<tr>
<td>19</td>
<td>thermal</td>
<td>Brunetiere</td>
<td>3,456</td>
<td>66,528</td>
</tr>
<tr>
<td>20</td>
<td>raelsky6</td>
<td>Simon</td>
<td>3,402</td>
<td>130,371</td>
</tr>
</tbody>
</table>

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 $[f_1, \ldots, f_{n-1}] = M^{-H} A^H [q_1, \ldots, q_{n-1}]$, $r_0 = b - A x_0$ and $g_0 = r_0$.
   Compute $\hat{g}_0 = M^{-H} r_0$, $w_0 = A \hat{g}_0$, $e_0 = q_1^H w_0$, $e_0 = q_1^H r_0$.
3. For $j = 0, 1, 2, \ldots$
4. For $i = 1, 2, \ldots, n - 1$
5. \[ \alpha_{jn+i} = e_{jn+i-1}/c_{jn+i-1}; \]
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. \[ e_{jn+i} = q_{jn+i}^H r_{jn+i}; \]
9. If $j \geq 1$
   10. \[ \beta_{(j-1)n+i}^{(jn+i)} = -e_{jn+i}/e_{(j-1)n+i}; \]
    \[ \% \quad \beta_{(j-1)n+i}^{(jn+i)} = -\omega_j \beta_{(j-1)n+i}^{(jn+i)}; \]
11. \[ z_{jn+i} = r_{jn+i} + \beta_{(j-1)n+i}^{(jn+i)} w_{(j-1)n+i}; \]
12. \[ g_{jn+i} = \beta_{(j-1)n+i}^{(jn+i)} g_{(j-1)n+i}; \]
13. For $s = i + 1, \ldots, n - 1$
   14. \[ \hat{\beta}_{(j-1)n+s}^{(jn+i)} = -q_{jn+s}^H z_{w}/e_{(j-1)n+s}; \]
    \[ \% \quad \hat{\beta}_{(j-1)n+s}^{(jn+i)} = -\omega_j \hat{\beta}_{(j-1)n+s}^{(jn+i)}; \]
   15. \[ z_{w} = z_{w} + \beta_{(j-1)n+s}^{(jn+i)} w_{(j-1)n+s}; \]
16. \[ g_{jn+i} = g_{jn+i} + \beta_{(j-1)n+s}^{(jn+i)} g_{(j-1)n+s}; \]
17. End
18. \[ g_{jn+i} = z_{w} - 1 \omega_j \beta_{jn+i}; \]
19. For $s = 0, \ldots, i - 1$
20. \[ \beta_{jn+s}^{(jn+i)} = -f_{jn+s}^H g_{jn+i}/c_{jn+s}; \]
21. \[ g_{jn+i} = g_{jn+i} + \beta_{jn+s}^{(jn+i)} g_{jn+s}; \]
\[ \beta_{jn}^{(j+n+i)} = -t_i^H r_{jn+i}/c_{jn}; \]
\[ g_{jn+i} = r_{jn+i} + \beta_{jn+i}^* g_{jn}; \]
\[ \text{For } s = 1, \ldots, i - 1 \]
\[ \beta_{jn+s} = -t_s^H g_{jn+1}/c_{jn+s}; \]
\[ g_{jn+i} = g_{jn+i} + \beta_{jn+s}^* g_{jn+s}; \]
\[ g_{jn+i} = M^{-1} g_{jn+i}; \]
\[ w_{jn+i} = A g_{jn+i}; \]
\[ c_{jn+i} = q_i^H w_{jn+i}; \]
\[ \alpha_{jn+n} = c_{jn+n-1}/c_{jn+n-1}; \]
\[ x_{jn+n} = x_{jn+n-1} + c_{jn+n} g_{jn+n-1}; \]
\[ u_{jn+n} = r_{jn+n-1} - \alpha_{jn+n} w_{jn+n-1}; \]
\[ u_{jn+n} = M^{-1} u_{jn+n}; \]
\[ \omega_{j+1} = (A u_{jn+n})^H u_{jn+n}/\|A u_{jn+n}\|_2^2; \]
\[ x_{jn+n} = x_{jn+n} + \omega_{j+1} u_{jn+n}; \]
\[ r_{jn+n} = -\omega_{j+1} A u_{jn+n} + u_{jn+n}; \]
\[ c_{jn+n} = q_i^H r_{jn+n}; \]
\[ \beta_{jn+n} = -c_{jn+n}/c_{(j-1)n+n}; \]
\[ x_{jn+n} = x_{jn+n} + \omega_{j+1} g_{jn+n}; \]
\[ u_{jn+n} = r_{jn+n} + \beta_{jn+n} g_{jn+n}; \]
\[ z_u = r_{jn+n} + \beta_{jn+n} g_{jn+n}; \]
\[ g_{jn+n} = \beta_{(j-1)n+n} g_{(j-1)n+n}; \]
\[ g_{jn+n} = \beta_{(j-1)n+n} g_{(j-1)n+n}; \]
\[ \text{For } s = 1, \ldots, n - 1 \]
Table 5.3

Experimental results run on the data in Table 5.1. “-” means no convergence within 10N matrix-vector multiplications with A.

<table>
<thead>
<tr>
<th>No.</th>
<th>m</th>
<th>k</th>
<th>MVs</th>
<th>Time (s)</th>
<th>True error</th>
<th>No.</th>
<th>m</th>
<th>k</th>
<th>MVs</th>
<th>Time (s)</th>
<th>True error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
<td>10</td>
<td>191</td>
<td>0.54</td>
<td>7.62 × 10⁻⁴</td>
<td>11</td>
<td>30</td>
<td>10</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>10</td>
<td>180</td>
<td>0.77</td>
<td>6.64 × 10⁻⁴</td>
<td>450</td>
<td>150</td>
<td>-</td>
<td>-</td>
<td>1,050</td>
<td>60.34  × 10⁻⁶</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>10</td>
<td>36,770</td>
<td>61.06</td>
<td>1.00 × 10⁻¹</td>
<td>12</td>
<td>30</td>
<td>10</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td>10</td>
<td>6,340</td>
<td>17.00</td>
<td>3.24 × 10⁻⁴</td>
<td>60</td>
<td>20</td>
<td>7,652</td>
<td>8.23</td>
<td>8.37 × 10⁻⁶</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>30</td>
<td>10</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>30</td>
<td>10</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>120</td>
<td>40</td>
<td>44,070</td>
<td>212.85</td>
<td>5.33 × 10⁻⁴</td>
<td>450</td>
<td>150</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>40</td>
<td>10</td>
<td>14,744</td>
<td>40.63</td>
<td>9.89 × 10⁻⁸</td>
<td>16</td>
<td>30</td>
<td>10</td>
<td>462</td>
<td>1.05</td>
<td>6.59 × 10⁻⁸</td>
</tr>
<tr>
<td>8</td>
<td>60</td>
<td>20</td>
<td>5,466</td>
<td>19.55</td>
<td>9.46 × 10⁻⁸</td>
<td>60</td>
<td>20</td>
<td>423</td>
<td>1.54</td>
<td>6.10 × 10⁻⁸</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>30</td>
<td>10</td>
<td>4,772</td>
<td>5.95</td>
<td>9.54 × 10⁻⁸</td>
<td>17</td>
<td>30</td>
<td>10</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>60</td>
<td>20</td>
<td>2,142</td>
<td>4.10</td>
<td>4.74 × 10⁻⁸</td>
<td>450</td>
<td>150</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>11</td>
<td>30</td>
<td>10</td>
<td>579</td>
<td>3.96</td>
<td>7.06 × 10⁻⁸</td>
<td>18</td>
<td>30</td>
<td>10</td>
<td>30</td>
<td>0.21</td>
<td>4.68 × 10⁻¹²</td>
</tr>
<tr>
<td>12</td>
<td>60</td>
<td>20</td>
<td>420</td>
<td>3.11</td>
<td>1.47 × 10⁻⁷</td>
<td>60</td>
<td>20</td>
<td>60</td>
<td>0.32</td>
<td>7.61 × 10⁻¹⁶</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>30</td>
<td>10</td>
<td>1,780</td>
<td>5.55</td>
<td>9.77 × 10⁻⁸</td>
<td>19</td>
<td>30</td>
<td>10</td>
<td>30</td>
<td>0.24</td>
<td>4.02 × 10⁻¹⁰</td>
</tr>
<tr>
<td>14</td>
<td>60</td>
<td>20</td>
<td>1,500</td>
<td>4.96</td>
<td>9.70 × 10⁻⁸</td>
<td>60</td>
<td>20</td>
<td>60</td>
<td>0.37</td>
<td>1.32 × 10⁻¹⁴</td>
<td></td>
</tr>
</tbody>
</table>

46. \[ \tilde{\beta}^{(jn+n)}_{jn+s} = -q_{jn+n}^H z_w / c_{jn+s}; \quad \% \tilde{\beta}^{(jn+n)}_{k+jn} = -\omega_{j+1} \beta^{(jn+n)}_{k+jn} \]

47. \[ z_w = z_s + \tilde{\beta}^{(jn+n)}_{jn+s} w_{jn+n}; \]

48. \[ g_{jn+n} = g_{jn+n} + \tilde{\beta}^{(jn+n)}_{jn+s} g_{jn+n}; \]

49. \[ \text{End} \]

50. \[ g_{jn+n} = z_w - \frac{1}{\omega_{j+1}} g_{jn+n}; \quad \tilde{g}_{jn+n} = M^{-1} g_{jn+n}; \]

51. \[ w_{jn+n} = A g_{jn+n}; \quad c_{jn+n} = q_{jn+n}^H w_{jn+n}; \]

52. \[ \text{End} \]

Matlab code of Algorithm 7.1

1. function \([x, err, iter, flag] = mlbicgstabt(A, x, b, Q, M, max_it, tol, kappa)\)
2. \[ \text{function} \]
3. \[ \% input: A: N-by-N matrix; M: N-by-N preconditioner matrix. \]
4. \[ \% Q: N-by-n shadow matrix \([q_1, \ldots, q_n]\); x: initial guess. \]
5. \[ \% b: right hand side vector. max_it: maximum number of iterations. \]
6. \[ \% tol: error tolerance. \]
7. \[ \% kappa: \quad (real number) minimization step controller: \]
8. \[ \% kappa = 0, standard minimization \]
9. \[ \% kappa > 0, Sleijpen-van der Vorst minimization \]
10. \[ \% output: x: solution computed. err: error norm.  \]
11. \[ \text{flag: = 0, solution found to tolerance} \]
12. \[ \% = 1, no convergence given max_it iterations \]
13. \[ \% = -1, breakdown. \]
14. \[ \% storage: F: N \times (n - 1) matrix. G, Q, W: N \times n matrices. A, M: N \times N matrices. \]
15. \[ x, r, q_{jn}, z, b: N \times 1 matrices. c: 1 \times n matrix. \]
## Table 5.4

Experimental results run on the data in Table 5.1. ‘-’ means no convergence within 10N matrix-vector multiplications with A and $A^H$.

<table>
<thead>
<tr>
<th>No.</th>
<th>n</th>
<th>ML(n)BiCGStab (Alg. 5.1 in [29])</th>
<th>ML(n)BiCGStab</th>
<th>MVs</th>
<th>Time (s)</th>
<th>True error</th>
<th>MVs</th>
<th>Time (s)</th>
<th>True error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>0.79</td>
<td>207</td>
<td>9.98</td>
<td>1.26 × 10^-8</td>
<td>0.61</td>
<td>9.07</td>
<td>1.26 × 10^-8</td>
<td>0.61</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>1.73</td>
<td>218</td>
<td>9.90</td>
<td>1.11 × 10^-8</td>
<td>0.99</td>
<td>4.44</td>
<td>1.11 × 10^-8</td>
<td>0.99</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>17.41</td>
<td>3.75</td>
<td>12.01</td>
<td>1.11 × 10^-8</td>
<td>14.80</td>
<td>8.16</td>
<td>1.11 × 10^-8</td>
<td>14.80</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>20.60</td>
<td>2.61</td>
<td>12.01</td>
<td>1.11 × 10^-8</td>
<td>14.80</td>
<td>8.16</td>
<td>1.11 × 10^-8</td>
<td>14.80</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>23.30</td>
<td>5.37</td>
<td>18.45</td>
<td>1.12 × 10^-8</td>
<td>8.62</td>
<td>18.45</td>
<td>1.12 × 10^-8</td>
<td>8.62</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>37.34</td>
<td>3.50</td>
<td>22.57</td>
<td>8.21 × 10^-8</td>
<td>10.90</td>
<td>22.57</td>
<td>8.21 × 10^-8</td>
<td>10.90</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>93.06</td>
<td>1.40 × 10^-8</td>
<td>17.84</td>
<td>9.35 × 10^-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>40.27</td>
<td>26.57</td>
<td>47.88</td>
<td>9.12 × 10^-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>45.58</td>
<td>15.19</td>
<td>51.97</td>
<td>9.88 × 10^-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>13.26</td>
<td>3.29</td>
<td>16.17</td>
<td>8.24 × 10^-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>15.99</td>
<td>2.99</td>
<td>14.37</td>
<td>7.06 × 10^-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>8.25</td>
<td>2.44</td>
<td>6.61</td>
<td>9.36 × 10^-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>15.20</td>
<td>2.59</td>
<td>11.35</td>
<td>8.73 × 10^-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>3.72</td>
<td>374</td>
<td>3.98</td>
<td>1.29 × 10^-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>3.57</td>
<td>367</td>
<td>3.69</td>
<td>1.39 × 10^-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>2.87</td>
<td>10.46</td>
<td>42.02</td>
<td>9.07 × 10^-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>16.45</td>
<td>1.80</td>
<td>9.73</td>
<td>8.98 × 10^-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>56.57</td>
<td>6.05</td>
<td>49.77</td>
<td>1.01 × 10^-7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>60.79</td>
<td>5.70</td>
<td>45.62</td>
<td>2.71 × 10^-7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>9</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>9</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>8</td>
<td>5.06</td>
<td>6.15</td>
<td>5.21</td>
<td>7.90 × 10^-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>5.62</td>
<td>5.37</td>
<td>6.63</td>
<td>9.45 × 10^-10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>8</td>
<td>59.89</td>
<td>4.56</td>
<td>60.55</td>
<td>9.08 × 10^-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>14</td>
<td>21</td>
<td>0.03</td>
<td>6.48 × 10^-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>8</td>
<td>0.03</td>
<td>26</td>
<td>0.03</td>
<td>6.29 × 10^-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>1.20</td>
<td>265</td>
<td>0.55</td>
<td>6.46 × 10^-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>8</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>8</td>
<td>1.82</td>
<td>2.58</td>
<td>23.43</td>
<td>9.67 × 10^-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>8</td>
<td>2.11</td>
<td>4.14</td>
<td>46.36</td>
<td>4.44 × 10^-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>8</td>
<td>5.89</td>
<td>1.54</td>
<td>7.49</td>
<td>9.90 × 10^-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>8</td>
<td>4.17</td>
<td>1.43</td>
<td>3.86</td>
<td>4.44 × 10^-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

17. $N = \text{size}(A, 2); n = \text{size}(Q, 2)$;
18. $G = \text{zeros}(N, n); W = \text{zeros}(N, n)$; % initialize work spaces
19. if $n > 1$, $F = \text{zeros}(N, n - 1)$; end
20. $c = \text{zeros}(1, n)$; % end initialization
21.
22.   iter = 0; flag = 1; $\text{barm}2 = \text{norm}(b)$;
23.   if $\text{barm}2 = 0.0$, $\text{barm}2 = 1.0$; end
24.   $r = b - A \ast x$; $\text{err} = \text{norm}(r)/\text{barm}2$;
25.   if err < tol, flag = 0; return, end
26.  

Experimental results run on the data in Table 5.1. ‘-’ means no convergence within 100 matrix-vector multiplications with $A$. 

<table>
<thead>
<tr>
<th>No.</th>
<th>$n$</th>
<th>MVs</th>
<th>Time (s)</th>
<th>True error</th>
<th>No.</th>
<th>MVs</th>
<th>Time (s)</th>
<th>True error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>201</td>
<td>0.72</td>
<td>$7.30 \times 10^{-8}$</td>
<td>11</td>
<td>8</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>201</td>
<td>1.44</td>
<td>$8.81 \times 10^{-8}$</td>
<td>128</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>4,374</td>
<td>17.42</td>
<td>$9.85 \times 10^{-8}$</td>
<td>12</td>
<td>8</td>
<td>6,706</td>
<td>6.21</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>3,060</td>
<td>24.28</td>
<td>$7.44 \times 10^{-8}$</td>
<td>16</td>
<td>5,942</td>
<td>9.00</td>
<td>$9.10 \times 10^{-8}$</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>4,905</td>
<td>19.87</td>
<td>$9.84 \times 10^{-8}$</td>
<td>13</td>
<td>8</td>
<td>13,775</td>
<td>124.83</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>3,532</td>
<td>27.68</td>
<td>$9.69 \times 10^{-8}$</td>
<td>16</td>
<td>4,784</td>
<td>59.82</td>
<td>$9.27 \times 10^{-8}$</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>14</td>
<td>8</td>
<td>19</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>17,388</td>
<td>137.93</td>
<td>$9.39 \times 10^{-8}$</td>
<td>16</td>
<td>21</td>
<td>0.08</td>
<td>7.68 \times 10^{-8}</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>38,420</td>
<td>106.12</td>
<td>$8.68 \times 10^{-8}$</td>
<td>15</td>
<td>8</td>
<td>266</td>
<td>0.78</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>15,652</td>
<td>105.52</td>
<td>$7.70 \times 10^{-8}$</td>
<td>16</td>
<td>272</td>
<td>1.54</td>
<td>$9.70 \times 10^{-8}$</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>3,173</td>
<td>17.50</td>
<td>$9.78 \times 10^{-8}$</td>
<td>16</td>
<td>8</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>3,260</td>
<td>18.66</td>
<td>$9.21 \times 10^{-8}$</td>
<td>32</td>
<td>1,170</td>
<td>15.60</td>
<td>$9.42 \times 10^{-8}$</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>2,490</td>
<td>9.51</td>
<td>$9.28 \times 10^{-8}$</td>
<td>17</td>
<td>8</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>2,226</td>
<td>16.17</td>
<td>$8.25 \times 10^{-8}$</td>
<td>128</td>
<td>12,726</td>
<td>1,127.75</td>
<td>$9.60 \times 10^{-8}$</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>368</td>
<td>3.78</td>
<td>$9.01 \times 10^{-8}$</td>
<td>18</td>
<td>8</td>
<td>24</td>
<td>0.07</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>347</td>
<td>4.16</td>
<td>$4.50 \times 10^{-8}$</td>
<td>16</td>
<td>22</td>
<td>0.08</td>
<td>$8.84 \times 10^{-8}$</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>38,444</td>
<td>39.94</td>
<td>$9.58 \times 10^{-8}$</td>
<td>19</td>
<td>8</td>
<td>266</td>
<td>0.78</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>1,590</td>
<td>12.03</td>
<td>$8.24 \times 10^{-8}$</td>
<td>16</td>
<td>25</td>
<td>0.13</td>
<td>$9.16 \times 10^{-8}$</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>8,886</td>
<td>55.19</td>
<td>$1.19 \times 10^{-6}$</td>
<td>20</td>
<td>8</td>
<td>1,563</td>
<td>8.49</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>5,428</td>
<td>58.54</td>
<td>$2.16 \times 10^{-7}$</td>
<td>16</td>
<td>685</td>
<td>5.75</td>
<td>$6.75 \times 10^{-8}$</td>
</tr>
</tbody>
</table>

27. if $n > 1$, \( F = M' \backslash (A' \ast Q(:,1:n-1)) \); end
28. \( G(:,1) = r; g_jh = M \backslash r; W(:,1) = A \ast g_jh; c(1) = Q(:,1)' \ast W(:,1); \)
29. if \( c(1) == 0 \), flag = -1; return, end
30. \( e = Q(:,1)' \ast r; \)
31.
32. for \( j = 0 : \max \_it \)
33. for \( i = 1 : n \)
34. \( alpha = e/c(i); x = x + alpha \ast g_jh; r = r - alpha \ast W(:,i); \)
35. \( err = \text{norm}(r)/\text{norm2}; \ iter = iter + 1; \)
36. if \( err < \text{tol}, \) flag = 0; return, end
37. if \( \text{iter} >= \max \_it \), return, end
38.
39. \( e = Q(:,i+1)' \ast r; \)
40. if \( j >= 1 \)
41. \( beta = -e/c(i+1); \)
42. \( W(:,i+1) = r + beta \ast W(:,i+1); \)
43. \( G(:,i+1) = beta \ast G(:,i+1); \)
44. for \( s = i + 1 : n-1 \)
45. \( beta = -Q(:,s+1)' \ast W(:,i+1)/c(s+1); \)
46. \( W(:,i+1) = W(:,i+1) + beta \ast W(:,s+1); \)
47. \( G(:,i+1) = G(:,i+1) + beta \ast G(:,s+1); \)
48. end
49. \( G(:,i+1) = W(:,i+1) - G(:,i+1)/\omega_{\text{e}g}; \)
50. for \( s = 0 : i - 1 \)
51. \( beta = -F(:,s+1)' \ast G(:,i+1)/c(s+1); \)
52. \( G(:,i+1) = G(:,i+1) + beta \ast G(:,s+1); \)
53.  end
54. else
55.    beta = -F(:,1)' * r/c(1);  G(:,i + 1) = r + beta * G(:,1);
56.    for s = 1 : i - 1
57.      beta = -F(:,s + 1)' * G(:,i + 1)/c(s + 1);
58.      G(:,i + 1) = G(:,i + 1) + beta * G(:,s + 1);
59.    end
60.  end
61.  g_jh = M\G(:,i + 1);  W(:,i + 1) = A * g_jh;
62.  c(i + 1) = Q(:,i + 1)' * W(:,i + 1);
63.  if c(i + 1) == 0,  flag = -1; return, end
64.  end
65.  alpha = e/c(n);  x = x + alpha * g_jh;  r = r - alpha * W(:,n);
66.  err = norm(r)/bnrm2;
67.  if err < tol,  flag = 0; iter = iter + 1; return, end
68.  g_jh = M\r;  z = A * g_jh;  omega = z' * z;
69.  if omega == 0,  flag = -1; return, end
70.  rho = z' * r;  omega = rho/omega;
71.  if kappa > 0
72.    rho = rho/(norm(z) * norm(r));  abs_om = abs(rho);
73.    if (abs_om < kappa) & (abs_om ~= 0)
74.      omega = omega * kappa/abs_om;
75.    end
76.  end
77.  if omega == 0,  flag = -1; return, end
78.  x = x + omega * g_jh;  r = r - omega * z;
79.  err = norm(r)/bnrm2; iter = iter + 1;
80.  if err < tol,  flag = 0; return, end
81.  if iter >= max2it, return, end
82.  end
83.  c = Q(:,1)' * r;  beta = -e/c(1);
84.  W(:,1) = r + beta * W(:,1);  G(:,1) = beta * G(:,1);
85.  for s = 1 : n - 1
86.    beta = -Q(:,s + 1)' * W(:,1)/c(s + 1);
87.    W(:,1) = W(:,1) + beta * W(:,s + 1);
88.    G(:,1) = G(:,1) + beta * G(:,s + 1);
89.  end
90.  G(:,1) = W(:,1) - G(:,1)/omega;  g_jh = M\G(:,1);
91.  W(:,1) = A * g_jh;  c(1) = Q(:,1)' * W(:,1);
92.  if c(1) == 0,  flag = -1; return, end
93. end

REFERENCES


[31] M. Yeung, ML(n)BiCGStab: a ML(n)BiCGStab variant with A-transpose, Proceedings of 2012 Fourth International Conference on Computational and Information Sciences, China,
2012.


