Journal of Computational and Applied Mathematics 272 (2014) 57–69

Contents lists available at ScienceDirect

Journal of Computational and Applied Mathematics

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

ML(n)BiCGStabt: A ML(n)BiCGStab variant with **A**-transpose

Man-Chung Yeung

Department 3036, 1000 East University Avenue, Laramie, WY 82071, United States

ARTICLE INFO

Article history:
Received 18 January 2013
Received in revised form 27 January 2014

MSC:
primary 65F10
65F15
secondary 65F25
65F30

Keywords:
IDR
CGS
BiCGStab
ML(n)BiCGStab
Multiple starting Lanczos
Krylov subspace

ABSTRACT

The 1980 IDR method (Wesseling and Sonneveld, 1980 [12]) 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 (Yeung, 2012) 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.

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 [1] in 1999 and its algorithms were recently reformulated by Yeung [2]. ML(n)BiCGStab is a natural generalization of BiCGStab [3], 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 [4] and van der Vorst [3] to construct CGS and BiCGStab from BiCG [5]. In theory, ML(n)BiCGStab is a method that lies between the Lanczos-based BiCGStab and the Arnoldi-based GMRES/FOM [6]. In fact, it is a BiCGStab when $n = 1$ and becomes a GMRES/FOM when $n = N$ (see [2,7]). 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 [2,1].

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 [1] is. Numerical experiments, however, showed that this new ML(n)BiCG was unstable and weaker than the standard BiCG. Moreover, in [8], 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. From their experiments about the multi-input

E-mail address: myeung@uwyo.edu.

http://dx.doi.org/10.1016/j.cam.2014.04.011
0377-0427/

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 [7,9], standing for ML(n)BiCGStab with transpose. We remark that $\mathbf{A}^H$ has been used in [10] to improve the parallelism of GPBiCG(m, l) [11]. Here we want to use $\mathbf{A}^H$ 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 [2], derived from different definitions of the residual vectors $\mathbf{r}_k$. 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 [12], CGS and BiCGStab exist. Among them are BiCGStab2 [13], BiCGStab(l) [14], GPBi-CG [15], IDR(s) [16,17], IDRstab [18], GPBiCG(m, l) [11], and GBi-CGSTAB(s, l) [19]. Related articles include [20–23].

The outline of the paper is as follows. In Section 2, index functions in [8] are introduced. They are helpful in the construction of a ML(n)BiCGStab algorithm. In Section 3, we present the ML(n)BiCG algorithm from [1]. The derivation of every ML(n)BiCGStab algorithm is based on it. In Section 4, we introduce the ML(n)BiCGStabt algorithm and its properties. In Section 5, numerical experiments are presented, and in Section 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

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

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

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

3. ML(n)BiCG

Analogously to the derivation of BiCGStab from BiCG, the ML(n)BiCGStab algorithms [2] were derived from a BiCG-like algorithm named ML(n)BiCG, which was built upon a 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 [1].

Consider the solution of (1.1). Throughout the paper we do not assume the coefficient matrix **A** is nonsingular. In [2], 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 $\mathbf{q}_1, \ldots, \mathbf{q}_n \in \mathbb{C}^N$, which we call *left starting vectors* or *shadow vectors*. Define

$$\mathbf{p}_k = \left(\mathbf{A}^H\right)^{g_n(k)} \mathbf{q}_{r_n(k)}, \quad k \in \mathbb{N}. \tag{3.1}$$

The following algorithm for the solution of (1.1) is from [1].

Algorithm 3.1. ML(n)BiCG

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

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

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

where $\mathbf{Q} \equiv [\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n]$, $\mathcal{K}_0(\mathbf{M}, \mathbf{v}) = \{\mathbf{0}\}$ and

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

for $\mathbf{M} \in \mathbb{C}^{N\times N}$, $\mathbf{v} \in \mathbb{C}^N$ and $t \in \mathbb{N}$. Moreover, in this ML(n)BiCG, the basis used for $\mathcal{B}_k$ is not chosen to be bi-orthogonal, but simply the set $\{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_k\}$. Therefore, it can be viewed as a generalization of a one-sided Lanczos algorithm (see [24,25]).

It can be shown that the quantities of ML(n)BiCG satisfy the properties (see [2])

(a) $\widehat{\mathbf{x}}_k \in \widehat{\mathbf{x}}_0 + \mathcal{K}_k(\mathbf{A}, \widehat{\mathbf{r}}_0)$, $\widehat{\mathbf{r}}_k \in \widehat{\mathbf{r}}_0 + \mathbf{A}\mathcal{K}_k(\mathbf{A}, \widehat{\mathbf{r}}_0)$.
(b) $\widehat{\mathbf{r}}_k \perp span\{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_k\}$ and $\widehat{\mathbf{r}}_k \not\perp \mathbf{p}_{k+1}$.
(c) $\mathbf{A}\widehat{\mathbf{g}}_k \perp span\{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_k\}$ and $\mathbf{A}\widehat{\mathbf{g}}_k \not\perp \mathbf{p}_{k+1}$.

4. ML(n)BiCGStabt

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 $\mathbf{p}^H\widehat{\mathbf{r}}$ and $\mathbf{p}^H\mathbf{A}\widehat{\mathbf{g}}$ in BiCG can be replaced by inner products of the forms $\mathbf{q}^H\psi(\mathbf{A})\widehat{\mathbf{r}}$ and $\mathbf{q}^H\mathbf{A}\psi(\mathbf{A})\widehat{\mathbf{g}}$ respectively, where ψ 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 Section 3.

4.1. Algorithm

In [2], Yeung presented two ML(n)BiCGStab algorithms, labeled as Algorithms 4.1 and 5.1 respectively. Let ϕ_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 ω_k is a free parameter. Then the quantities in Algorithm 4.1 are defined by

$$\begin{aligned}&\mathbf{r}_k = \phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{r}}_k, \qquad &&\mathbf{u}_k = \phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{r}}_k,\\ &\mathbf{g}_k = \phi_{g_n(k)+1}(\mathbf{A})\widehat{\mathbf{g}}_k, \qquad &&\mathbf{d}_k = -\omega_{g_n(k)+1}\mathbf{A}\phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{g}}_k,\\ &\mathbf{w}_k = \mathbf{A}\mathbf{g}_k\end{aligned} \tag{4.1}$$

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

$$\begin{aligned}&\mathbf{r}_k = \phi_{g_n(k+1)}(\mathbf{A})\widehat{\mathbf{r}}_k, \qquad &&\mathbf{g}_k = \phi_{g_n(k+1)}(\mathbf{A})\widehat{\mathbf{g}}_k,\\ &\mathbf{u}_k = \phi_{g_n(k)}(\mathbf{A})\widehat{\mathbf{r}}_k, \qquad &&\mathbf{w}_k = \mathbf{A}\mathbf{g}_k\end{aligned} \tag{4.2}$$

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

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

Here $\mathbf{r}_k$ is the residual of the kth approximate solution $\mathbf{x}_k$. Numerical experiments in [2] 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

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k\mathbf{g}_{k-1}, \qquad \mathbf{r}_k = \mathbf{r}_{k-1} - \alpha_k\mathbf{w}_{k-1}$$

in most k-iterations, where α_k is a scalar. The true residual of the computed $\mathbf{x}_k$ is therefore

$$\mathbf{b} - \mathbf{A}\mathbf{x}_k = \mathbf{b} - \mathbf{A}(\mathbf{x}_{k-1} + \alpha_k\mathbf{g}_{k-1}) = (\mathbf{b} - \mathbf{A}\mathbf{x}_{k-1}) - \alpha_k\mathbf{A}\mathbf{g}_{k-1}. \tag{4.3}$$

In Algorithm 4.1, $\mathbf{w}_k$ is updated by $\mathbf{w}_k = \mathbf{A}\mathbf{g}_k$ (as it is defined in (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

[1] There is a similar comment on BiCGStab(l) [14, p. 27] when compared to BiCGStab2 [13].

$\mathbf{r}_k$ 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 ML(n)BiCGStabt.

The derivation of Algorithm 5.1 in [2] was divided into several stages, starting from ML(n)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 [2].

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

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

$$\begin{aligned} \beta_s^{(k)} &= -\mathbf{q}_{r_n(s+1)}^H \mathbf{A} \left(\mathbf{r}_k + \sum_{t=\max(k-n,0)}^{g_n(k)n-1} \beta_t^{(k)} (\mathbf{I} - \omega_{g_n(k+1)} \mathbf{A}) \mathbf{g}_t + \sum_{t=g_n(k)n}^{s-1} \beta_t^{(k)} \mathbf{g}_t \right) \Big/ \mathbf{q}_{r_n(s+1)}^H \mathbf{w}_s \\ &= -\mathbf{q}_{r_n(s+1)}^H \mathbf{A} \left(\mathbf{r}_k + \sum_{t=\max(k-n,0)}^{g_n(k)n-1} \beta_t^{(k)} (\mathbf{g}_t - \omega_{g_n(k+1)} \mathbf{w}_t) + \sum_{t=g_n(k)n}^{s-1} \beta_t^{(k)} \mathbf{g}_t \right) \Big/ \mathbf{q}_{r_n(s+1)}^H \mathbf{w}_s. \end{aligned}$$

It is because of the $\mathbf{A}$ in front of the parentheses, we cannot update $\mathbf{w}_k$ by $\mathbf{w}_k = \mathbf{A}\mathbf{g}_k$ 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 $\mathbf{f}_{r_n(s+1)} \equiv \mathbf{A}^H \mathbf{q}_{r_n(s+1)}$ is available, then Line 9 will become

$$\beta_s^{(k)} = -\mathbf{f}_{r_n(s+1)}^H \left(\mathbf{r}_k + \sum_{t=\max(k-n,0)}^{g_n(k)n-1} \beta_t^{(k)} (\mathbf{g}_t - \omega_{g_n(k+1)} \mathbf{w}_t) + \sum_{t=g_n(k)n}^{s-1} \beta_t^{(k)} \mathbf{g}_t \right) \Big/ \mathbf{q}_{r_n(s+1)}^H \mathbf{w}_s \tag{4.4}$$

and the troubling $\mathbf{A}$ is gone. It is the observation that leads to the ML(n)BiCGStabt algorithm.

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

$$\mathbf{F} \equiv \mathbf{A}^H \mathbf{Q} = [\mathbf{A}^H \mathbf{q}_1, \mathbf{A}^H \mathbf{q}_2, \ldots, \mathbf{A}^H \mathbf{q}_n]$$

is available. Recalling that $\mathbf{r}_k$ is the residual of $\mathbf{x}_k$, to be consistent with Lines 4, 13 and 14, we update the approximate solution $\mathbf{x}_k$ as

$$\mathbf{x}_k = \begin{cases} \mathbf{x}_{k-1} + \alpha_k \mathbf{g}_{k-1}, & \text{if } r_n(k) < n \\ \omega_{g_n(k+1)} \mathbf{u}_k + \mathbf{x}_{k-1} + \alpha_k \mathbf{g}_{k-1}, & \text{if } r_n(k) = n. \end{cases} \tag{4.5}$$

Adding (4.5) and $\mathbf{w}_k = \mathbf{A}\mathbf{g}_k$ 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 $\mathbf{r}_k$.

Table 4.1
Average cost per k-iteration of the preconditioned ML(n)BiCGStabt Algorithm A.1 and its storage. This table does not count the cost in Lines 1–2 of the algorithm.

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

Algorithm 4.1. ML(n)BiCGStabt without preconditioning

1. Choose an initial guess $\mathbf{x}_0$ and n vectors $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n$.
2. Compute $[\mathbf{f}_1, \ldots, \mathbf{f}_{n-1}] = \mathbf{A}^H[\mathbf{q}_1, \ldots, \mathbf{q}_{n-1}]$.
3. Compute $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$ and $\mathbf{g}_0 = \mathbf{r}_0$, $\mathbf{w}_0 = \mathbf{A}\mathbf{g}_0$, $c_0 = \mathbf{q}_1^H\mathbf{w}_0, \omega_0 = 1$.
4. For $k = 1, 2, \ldots,$ until convergence:
5. $\quad \alpha_k = \mathbf{q}_{r_n(k)}^H\mathbf{r}_{k-1}/c_{k-1}$;
6. $\quad$ If $r_n(k) < n$
7. $\quad\quad \mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k\mathbf{g}_{k-1}$; $\mathbf{r}_k = \mathbf{r}_{k-1} - \alpha_k\mathbf{w}_{k-1}$;
8. $\quad\quad \mathbf{z}_w = \mathbf{r}_k$; $\mathbf{g}_k = \mathbf{0}$;
9. $\quad\quad$ For $s = \max(k-n, 0), \ldots, g_n(k)n - 1$
10. $\quad\quad\quad \tilde{\beta}_s^{(k)} = -\mathbf{q}_{r_n(s+1)}^H\mathbf{z}_w/c_s$; $\quad \% \ \tilde{\beta}_s^{(k)} = -\omega_{g_n(k+1)}\beta_s^{(k)}$
11. $\quad\quad\quad \mathbf{z}_w = \mathbf{z}_w + \tilde{\beta}_s^{(k)}\mathbf{w}_s$;
12. $\quad\quad\quad \mathbf{g}_k = \mathbf{g}_k + \tilde{\beta}_s^{(k)}\mathbf{g}_s$;
13. $\quad\quad$ End
14. $\quad\quad \mathbf{g}_k = \mathbf{z}_w - \frac{1}{\omega_{g_n(k+1)}}\mathbf{g}_k$;
15. $\quad\quad$ For $s = g_n(k)n, \ldots, k-1$
16. $\quad\quad\quad \beta_s^{(k)} = -\mathbf{f}_{r_n(s+1)}^H\mathbf{g}_k/c_s$;
17. $\quad\quad\quad \mathbf{g}_k = \mathbf{g}_k + \beta_s^{(k)}\mathbf{g}_s$;
18. $\quad\quad$ End
19. $\quad$ Else
20. $\quad\quad \mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k\mathbf{g}_{k-1}$;
21. $\quad\quad \mathbf{u}_k = \mathbf{r}_{k-1} - \alpha_k\mathbf{w}_{k-1}$;
22. $\quad\quad \omega_{g_n(k+1)} = (\mathbf{A}\mathbf{u}_k)^H\mathbf{u}_k/\|\mathbf{A}\mathbf{u}_k\|_2^2$;
23. $\quad\quad \mathbf{x}_k = \mathbf{x}_k + \omega_{g_n(k+1)}\mathbf{u}_k$; $\mathbf{r}_k = -\omega_{g_n(k+1)}\mathbf{A}\mathbf{u}_k + \mathbf{u}_k$;
24. $\quad\quad \mathbf{z}_w = \mathbf{r}_k$; $\mathbf{g}_k = \mathbf{0}$;
25. $\quad\quad$ For $s = g_n(k)n, \ldots, k-1$
26. $\quad\quad\quad \tilde{\beta}_s^{(k)} = -\mathbf{q}_{r_n(s+1)}^H\mathbf{z}_w/c_s$; $\quad \% \ \tilde{\beta}_s^{(k)} = -\omega_{g_n(k+1)}\beta_s^{(k)}$
27. $\quad\quad\quad \mathbf{z}_w = \mathbf{z}_w + \tilde{\beta}_s^{(k)}\mathbf{w}_s$;
28. $\quad\quad\quad \mathbf{g}_k = \mathbf{g}_k + \tilde{\beta}_s^{(k)}\mathbf{g}_s$;
29. $\quad\quad$ End
30. $\quad\quad \mathbf{g}_k = \mathbf{z}_w - \frac{1}{\omega_{g_n(k+1)}}\mathbf{g}_k$;
31. $\quad$ End
32. $\quad \mathbf{w}_k = \mathbf{A}\mathbf{g}_k$; $c_k = \mathbf{q}_{r_n(k+1)}^H\mathbf{w}_k$;
33. End

Line 32 indicates that $\mathbf{w}_k$ is computed by $\mathbf{w}_k = \mathbf{A}\mathbf{g}_k$ for all k-iterations. Therefore the updates for $\mathbf{x}_k$ and $\mathbf{r}_k$ in Algorithm 4.1 are

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k\mathbf{g}_{k-1}, \qquad \mathbf{r}_k = \mathbf{r}_{k-1} - \alpha_k\mathbf{A}\mathbf{g}_{k-1} \tag{4.6}$$

which meets the goal that we set right before DS #8 on improving the stability of Algorithm 5.1 in [2]. The stability of updates of the type (4.6) has been studied in detail by Neumaier [26] and Sleijpen and van der Vorst [27].

We remark that (i) Algorithm 4.1 does not compute $\mathbf{u}_k$ when $r_n(k) < n$. In fact, $\mathbf{u}_k = \mathbf{r}_k$ when $r_n(k) < n$ from (4.2); (ii) if the $\mathbf{u}_k$ in Line 21 happens to be zero, then the $\mathbf{x}_k$ 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 A.1) is presented in Table 4.1. Note that we do not need to store both $\mathbf{A}$ and $\mathbf{A}^H$ since $\mathbf{A}^H$ is only used in Line 2. Compared with Algorithm 5.1 in [2], 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 [2] for a detailed analysis). The algorithm, however, can encounter a near breakdown in its implementation. The divisors in the algorithm are c_k, $\|\mathbf{A}\mathbf{u}_k\|_2$ and $\omega_{g_n(k+1)}$. If $\|\mathbf{A}\mathbf{u}_k\|_2 \approx 0$, then $\mathbf{u}_k \approx \mathbf{0}$ and the $\mathbf{x}_k$ in Line 20 is an approximate solution. When $\omega_{g_n(k+1)} \approx 0$, we can add some small perturbation to it or adopt the Sleijpen–van der Vorst minimization control technique (see [2])

so that it is relatively far from 0. About c_k, it can be shown that it is a quantity that relates to $\omega_{g_n(k+1)}$ and the ML(n)BiCG divisor $\mathbf{p}_{k+1}^H\mathbf{A}\widehat{\mathbf{g}}_k$. The ML($n$)BiCG divisor $\mathbf{p}_{k+1}^H\mathbf{A}\widehat{\mathbf{g}}_k$ 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 [13], in most cases such breakdowns can be overcome by a look-ahead step, see [28–31] and further references cited there. Moreover, for how to avoid a breakdown in a nonsymmetric block Lanczos algorithm, one can consult [32].

4.2. Properties

Since the quantities of ML(n)BiCGStabt are defined exactly the same as those of Algorithm 5.1 in [2], ML(n)BiCGStabt shares the same properties with Algorithm 5.1.

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

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

and

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

Denote by $\mathbf{S}_l$ and $\mathbf{W}_l$ the $l \times l$ leading principal submatrices of $\mathbf{S}_\nu$ and $\mathbf{W}_\nu$ respectively (Joubert [33,34] called these matrices the moment matrices). Then some facts about ML(n)BiCGStabt (Algorithm 4.1) are summarized as follows.

Proposition 4.2 (*[2, Prop. 5.1]*). *In infinite precision arithmetic, if* $\prod_{l=1}^{\nu}\det(\mathbf{S}_l)\det(\mathbf{W}_l) \neq 0$, $\omega_{g_n(k+1)} \neq 0$ *and* $1/\omega_{g_n(k+1)} \notin \sigma(\mathbf{A})$ *for* $1 \leq k \leq \nu-1$, *where* $\sigma(\mathbf{A})$ *is the spectrum of* $\mathbf{A}$, *then ML(n)BiCGStabt does not break down by zero division for* $k = 1, 2, \ldots, \nu$, *and the approximate solution* $\mathbf{x}_\nu$ *at step* $k = \nu$ *is exact to the system (1.1). Moreover, the computed quantities satisfy*

(a) $\mathbf{x}_k \in \mathbf{x}_0 + span\{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \ldots, \mathbf{A}^{g_n(k+1)+k-1}\mathbf{r}_0\}$ *and* $\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k \in \mathbf{r}_0 + span\{\mathbf{A}\mathbf{r}_0, \mathbf{A}^2\mathbf{r}_0, \ldots, \mathbf{A}^{g_n(k+1)+k}\mathbf{r}_0\}$ *for* $1 \leq k \leq \nu-1$.
(b) $\mathbf{r}_k \neq \mathbf{0}$ *for* $1 \leq k \leq \nu-1$; $\mathbf{r}_\nu = \mathbf{0}$.
(c) $\mathbf{r}_k \perp span\{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_{r_n(k)}\}$ *and* $\mathbf{r}_k \not\perp \mathbf{q}_{r_n(k)+1}$ *for* $1 \leq k \leq \nu-1$ *with* $r_n(k) < n$; $\mathbf{r}_k \not\perp \mathbf{q}_1$ *for* $1 \leq k \leq \nu-1$ *with* $r_n(k) = n$.
(d) $\mathbf{u}_k \perp span\{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n\}$ *for* $1 \leq k \leq \nu$ *with* $r_n(k) = n$.
(e) $\mathbf{A}\mathbf{g}_k \perp span\{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_{r_n(k)}\}$ *and* $\mathbf{A}\mathbf{g}_k \not\perp \mathbf{q}_{r_n(k)+1}$ *for* $1 \leq k \leq \nu-1$ *with* $r_n(k) < n$; $\mathbf{A}\mathbf{g}_k \not\perp \mathbf{q}_1$ *for* $1 \leq k \leq \nu-1$ *with* $r_n(k) = n$.

In Section 6.2 of [2], relations of Algorithm 5.1 in [2] 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 $\mathbf{A}^H$ in its implementation, if we set $n \geq \nu$ and $\mathbf{q}_k = \mathbf{r}_{k-1}$.
(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 $\mathbf{A}^H$ in its implementation.

5. Numerical experiments

A preconditioned ML(n)BiCGStabt algorithm can be obtained by applying Algorithm 4.1 to the system

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

where $\mathbf{M}$ is nonsingular, then recovering $\mathbf{x}$ through $\mathbf{x} = \mathbf{M}^{-1}\mathbf{y}$. The resulting algorithm, Algorithm A.1, together with its Matlab code are presented in the Appendix. 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 [35] and two algorithms of ML(n)BiCGStab: Algorithms 4.1 and 5.1 in [2]. The experimental results are shown in the following. All the test data were downloaded from The University of Florida Sparse Matrix Collection,[2] 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 $\mathbf{x}_0 = \mathbf{0}$, the stopping criterion $\|\mathbf{r}_k\|_2/\|\mathbf{b}\|_2 < 10^{-7}$ where $\mathbf{r}_k$ was the computed residual, and the Sleijpen–van der Vorst minimization control parameter (see [2]) $\kappa = 0$. As for the shadow vectors, we chose $\mathbf{Q} = [\mathbf{r}_0, sign(randn(N, n-1))]$. When a data did not provide a right-hand side, we set $\mathbf{b} = \mathbf{A}\mathbf{e}$ where $\mathbf{e}$ is the vector of ones.

We also compared ML(n)BiCGStabt with IDR(s) [17] and found that IDR(s) has the same level of stability with the ML(n)BiCGStab Algorithm 5.1 in [2], 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

[2] http://www.cise.ufl.edu/research/sparse/matrices/.

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.

No.	Matrix name	Group name	Size	Nonzeros
1	rdb5000	Bai	5,000	29,600
2	sherman3	HB	5,005	20,033
3	olm5000	Bai	5,000	19,996
4	cavity19	Drivcav	4,562	131,735
5	tols4000	Bai	4,000	8,784
6	ex31	Fidap	3,909	91,223
7	sherman5	HB	3,312	20,793
8	raefsky2	Simon	3,242	293,551
9	garon1	Garon	3,175	84,723
10	utm5940	Tokamak	5,940	83,842
11	Chebyshev3	Muite	4,101	36,879
12	pores_2	HB	1,224	9,613
13	tsopf_rs_b162_c1	Tsopf	5,374	205,399
14	rw5151	Bai	5,151	20,199
15	circuit_2	Bomhof	4,510	21,199
16	viscoplastic1	Quaglino	4,326	61,166
17	heart1	Norris	3,557	1,385,317
18	cage9	vanHeukelum	3,534	41,594
19	thermal	Brunetiere	3,456	66,528
20	raefsky6	Simon	3,402	130,371

Table 5.2
Experimental results run on the data in Table 5.1. "–" means no convergence within $10N$ matrix–vector multiplications with $\mathbf{A}$ and $\mathbf{A}^H$.

	BiCG			BiCGStab		
No.	MVs	Time (s)	True error	MVs	Time (s)	True error
1	380	0.92	8.69×10^{-8}	486	0.65	2.48×10^{-8}
2	–	–	–	12,112	12.63	8.70×10^{-8}
3	11,876	23.19	7.73×10^{-8}	–	–	–
4	–	–	–	–	–	–
5	–	–	–	–	–	–
6	6,702	42.34	5.28×10^{-8}	8,060	23.35	6.79×10^{-8}
7	3,570	6.30	8.19×10^{-8}	6,296	5.70	7.78×10^{-8}
8	732	13.47	2.63×10^{-8}	–	–	–
9	6,262	35.23	9.99×10^{-8}	–	–	–
10	21,800	132.54	9.10×10^{-8}	–	–	–
11	–	–	–	–	–	–
12	–	–	–	–	–	–
13	–	–	–	–	–	–
14	36	0.07	7.86×10^{-8}	–	–	–
15	–	–	–	668	0.68	3.56×10^{-8}
16	750	3.19	9.91×10^{-8}	–	–	–
17	–	–	–	–	–	–
18	46	0.14	1.92×10^{-8}	26	0.04	4.40×10^{-8}
19	46	0.21	5.73×10^{-8}	28	0.06	3.87×10^{-8}
20	1,832	15.48	7.83×10^{-8}	10,688	41.69	3.22×10^{-8}

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 $\|\mathbf{b} - \mathbf{A}\mathbf{x}\|_2/\|\mathbf{b}\|_2$ where $\mathbf{x}$ 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, as an improved version of Algorithm 5.1 in [2], ML(n)BiCGStabt has the same stability as Algorithm 4.1 in [2] and is more stable than Algorithm 5.1.

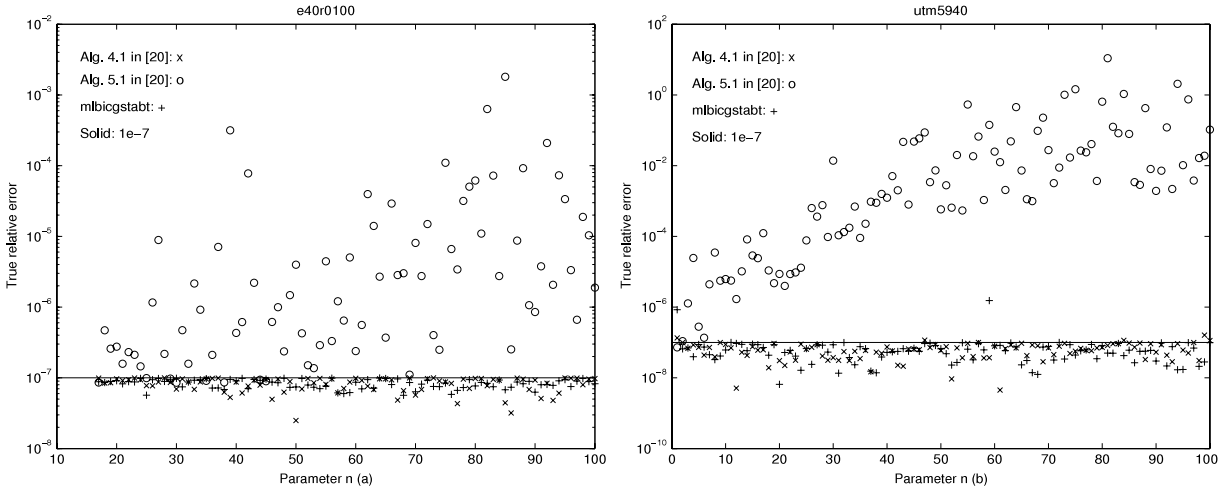

Fig. 5.1. Graphs of $E(n)$ against n. (a) e40r0100. Since all the three algorithms do not converge when $1 \leq n \leq 16$, we only plot the graphs over the range $17 \leq n \leq 100$; (b) utm5940.

Table 5.3
Experimental results run on the data in Table 5.1. "–" means no convergence within 10N matrix–vector multiplications with **A**.

	GMRES-DR						GMRES-DR				
No.	m	k	MVs	Time (s)	True error	No.	m	k	MVs	Time (s)	True error
1	30	10	191	0.54	7.62×10^{-8}	11	30	10	–	–	–
	60	20	180	0.77	6.64×10^{-8}		450	150	1,050	60.34	8.16×10^{-8}
2	30	10	36,770	61.06	1.00×10^{-7}	12	30	10	–	–	–
	60	20	6,340	17.00	3.24×10^{-7}		60	20	7,652	8.23	8.37×10^{-6}
3	30	10	–	–	–	13	30	10	–	–	–
	120	40	44,070	212.85	5.33×10^{-4}		450	150	–	–	–
4	30	10	–	–	–	14	30	10	30	0.21	1.58×10^{-9}
	450	150	8,561	470.71	8.82×10^{-5}		60	20	60	0.38	1.53×10^{-9}
5	30	10	–	–	–	15	30	10	710	1.11	7.92×10^{-8}
	120	40	22,149	91.93	7.60×10^{-7}		60	20	220	0.71	7.18×10^{-8}
6	30	10	15,474	40.63	9.89×10^{-8}	16	30	10	462	1.05	6.59×10^{-8}
	60	20	5,465	19.55	9.46×10^{-8}		60	20	423	1.54	6.10×10^{-8}
7	30	10	4,772	5.95	9.54×10^{-8}	17	30	10	–	–	–
	60	20	2,142	4.10	4.74×10^{-8}		450	150	–	–	–
8	30	10	579	3.96	7.06×10^{-8}	18	30	10	30	0.21	4.68×10^{-12}
	60	20	426	3.11	1.47×10^{-9}		60	20	60	0.32	7.61×10^{-16}
9	30	10	–	–	–	19	30	10	30	0.24	4.02×10^{-10}
	60	20	1,780	5.55	9.77×10^{-8}		60	20	60	0.37	1.32×10^{-15}
10	30	10	–	–	–	20	30	10	1,500	4.96	9.70×10^{-8}
	450	150	–	–	–		60	20	1,225	4.89	7.63×10^{-8}

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

1. *e*40*r*0100, 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. *utm*5940, 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 $E(n) \equiv \|\mathbf{b} - \mathbf{A}\mathbf{x}\|_2/\|\mathbf{b}\|_2$ where $\mathbf{x}$ is the computed solution output by an algorithm when it converges. The graphs of $E(n)$ are plotted in Fig. 5.1. It can be seen that the computed relative errors $\|\mathbf{r}_k\|_2/\|\mathbf{b}\|_2$ by Algorithm 5.1 significantly drift away from their exact counterparts. By contrast, however, the computed $\|\mathbf{r}_k\|_2/\|\mathbf{b}\|_2$ by ML(n)BiCGStabt and Algorithm 4.1 in [2] well approximate their corresponding true relative errors. In this experiment, ML(n)BiCGStabt improves the stability of Algorithm 5.1 in [2] significantly.

Table 5.4
Experimental results run on the data in Table 5.1. "–" means no convergence within $10N$ matrix–vector multiplications with $\mathbf{A}$ and $\mathbf{A}^H$.

		ML(n)BiCGStab (Alg. 5.1 in [2])			ML(n)BiCGStabt		
No.	n	MVs	Time (s)	True error	MVs	Time (s)	True error
1	8	202	0.79	6.26×10^{-8}	207	0.61	9.07×10^{-8}
	16	201	1.34	9.96×10^{-8}	218	0.99	4.44×10^{-8}
2	8	3,951	17.18	1.11×10^{-7}	3,757	12.01	9.34×10^{-8}
	16	2,645	20.60	1.11×10^{-4}	2,618	14.80	8.16×10^{-8}
3	8	5,337	23.30	1.12×10^{-7}	5,517	18.45	8.62×10^{-8}
	16	4,261	37.34	8.81×10^{-8}	3,570	22.57	8.21×10^{-8}
4	8	–	–	–	–	–	–
	16	14,197	93.06	1.40×10^{-7}	17,984	119.06	9.35×10^{-8}
5	8	24,532	40.27	8.16×10^{-8}	26,517	47.88	9.12×10^{-8}
	16	16,510	45.58	9.82×10^{-8}	15,194	51.97	9.88×10^{-8}
6	8	3,301	13.26	8.99×10^{-8}	3,529	16.17	8.24×10^{-8}
	16	3,205	15.99	8.53×10^{-8}	2,995	14.37	7.06×10^{-8}
7	8	2,607	8.25	9.37×10^{-8}	2,434	6.61	9.36×10^{-8}
	16	2,720	15.20	4.41×10^{-7}	2,599	11.35	8.73×10^{-8}
8	8	368	3.72	2.55×10^{-8}	374	3.98	1.29×10^{-8}
	16	346	3.57	3.10×10^{-8}	363	3.69	3.90×10^{-8}
9	8	8,973	38.87	9.83×10^{-8}	10,461	42.02	9.07×10^{-8}
	16	2,298	16.45	9.94×10^{-8}	1,804	9.73	8.98×10^{-8}
10	8	8,975	59.77	6.05×10^{-5}	8,410	49.77	1.01×10^{-5}
	16	5,825	60.79	1.85×10^{-4}	5,702	45.62	2.71×10^{-7}
11	8	–	–	–	–	–	–
	128	–	–	–	9,299	296.24	1.32×10^{-7}
12	8	5,667	4.87	9.70×10^{-8}	6,151	5.21	7.90×10^{-8}
	16	4,398	5.62	7.19×10^{-8}	5,370	6.63	9.45×10^{-8}
13	8	13,349	119.31	9.46×10^{-8}	12,759	105.37	9.82×10^{-8}
	16	4,572	59.89	7.86×10^{-8}	4,569	50.55	9.08×10^{-8}
14	8	14	0.05	6.48×10^{-8}	21	0.03	6.48×10^{-8}
	16	11	0.03	6.29×10^{-8}	26	0.03	6.29×10^{-8}
15	8	282	0.89	9.42×10^{-8}	265	0.55	6.36×10^{-8}
	16	215	1.20	9.73×10^{-8}	237	0.73	5.40×10^{-8}
16	8	–	–	–	–	–	–
	32	1,313	18.18	9.82×10^{-8}	2,588	23.43	9.67×10^{-8}
17	8	–	–	–	–	–	–
	128	9,924	822.34	8.39×10^{-8}	9,340	667.71	7.96×10^{-8}
18	8	24	0.06	2.90×10^{-8}	31	0.06	2.90×10^{-8}
	16	23	0.07	7.57×10^{-8}	38	0.08	7.57×10^{-8}
19	8	25	0.08	6.98×10^{-8}	32	0.09	6.98×10^{-8}
	16	28	0.10	2.85×10^{-8}	43	0.11	2.85×10^{-8}
20	8	1,199	5.89	7.67×10^{-8}	1,548	7.49	9.90×10^{-8}
	16	717	4.17	1.43×10^{-7}	632	3.86	6.44×10^{-8}

6. Concluding remarks

The original motivation of developing ML(n)BiCGStabt was to improve the stability of Algorithm 5.1 in [2]. 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.

Appendix

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

Table 5.5
Experimental results run on the data in Table 5.1. "–" means no convergence within 10N matrix–vector multiplications with $\mathbf{A}$.

	ML(n)BiCGStab (Alg. 4.1 in [2])				ML(n)BiCGStab (Alg. 4.1 in [2])				
No.	n	MVs	Time (s)	True error	No.	n	MVs	Time (s)	True error
1	8	201	0.72	7.30×10^{-8}	11	8	–	–	–
	16	201	1.44	8.81×10^{-8}		128	–	–	–
2	8	4,374	17.42	9.85×10^{-8}	12	8	6,706	6.21	9.81×10^{-8}
	16	3,060	24.28	7.44×10^{-8}		16	5,942	9.00	9.10×10^{-8}
3	8	4,905	19.87	9.84×10^{-8}	13	8	13,775	124.83	8.89×10^{-8}
	16	3,532	27.68	9.09×10^{-8}		16	4,784	59.82	9.27×10^{-8}
4	8	–	–	–	14	8	19	0.05	5.74×10^{-8}
	16	17,388	137.93	9.39×10^{-8}		16	21	0.08	7.68×10^{-8}
5	8	38,420	106.12	8.67×10^{-8}	15	8	266	0.78	9.68×10^{-8}
	16	15,652	105.82	7.70×10^{-8}		16	242	1.54	9.70×10^{-8}
6	8	3,173	17.50	9.78×10^{-8}	16	8	–	–	–
	16	3,260	18.66	9.21×10^{-8}		32	1,170	15.60	9.42×10^{-8}
7	8	2,490	9.51	9.25×10^{-8}	17	8	–	–	–
	16	2,226	16.17	8.25×10^{-8}		128	12,726	1,127.75	9.60×10^{-8}
8	8	368	3.78	9.01×10^{-8}	18	8	24	0.07	6.91×10^{-8}
	16	347	4.16	4.50×10^{-8}		16	22	0.08	8.83×10^{-8}
9	8	8,844	39.84	9.53×10^{-8}	19	8	26	0.09	5.01×10^{-8}
	16	1,590	12.03	8.24×10^{-8}		16	25	0.13	9.16×10^{-8}
10	8	8,686	55.19	1.19×10^{-6}	20	8	1,563	8.49	7.09×10^{-8}
	16	5,428	58.54	2.16×10^{-7}		16	685	5.75	6.75×10^{-8}

Algorithm A.1. ML(n)BiCGStabt with preconditioning

1. Choose an initial guess $\mathbf{x}_0$ and n vectors $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n$.
2. Compute $[\mathbf{f}_1, \ldots, \mathbf{f}_{n-1}] = \mathbf{M}^{-H}\mathbf{A}^H[\mathbf{q}_1, \ldots, \mathbf{q}_{n-1}]$, $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$ and $\mathbf{g}_0 = \mathbf{r}_0$.
   Compute $\hat{\mathbf{g}}_0 = \mathbf{M}^{-1}\mathbf{r}_0$, $\mathbf{w}_0 = \mathbf{A}\hat{\mathbf{g}}_0$, $c_0 = \mathbf{q}_1^H\mathbf{w}_0$, $e_0 = \mathbf{q}_1^H\mathbf{r}_0$.
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. $\mathbf{x}_{jn+i} = \mathbf{x}_{jn+i-1} + \alpha_{jn+i}\hat{\mathbf{g}}_{jn+i-1}$;
7. $\mathbf{r}_{jn+i} = \mathbf{r}_{jn+i-1} - \alpha_{jn+i}\mathbf{w}_{jn+i-1}$;
8. $e_{jn+i} = \mathbf{q}_{i+1}^H\mathbf{r}_{jn+i}$;
9. If $j \geq 1$
10. $\tilde{\beta}_{(j-1)n+i}^{(jn+i)} = -e_{jn+i}/c_{(j-1)n+i}$; % $\tilde{\beta}_{(j-1)n+i}^{(jn+i)} = -\omega_j\beta_{(j-1)n+i}^{(jn+i)}$
11. $\mathbf{z}_w = \mathbf{r}_{jn+i} + \tilde{\beta}_{(j-1)n+i}^{(jn+i)}\mathbf{w}_{(j-1)n+i}$;
12. $\mathbf{g}_{jn+i} = \tilde{\beta}_{(j-1)n+i}^{(jn+i)}\mathbf{g}_{(j-1)n+i}$;
13. For $s = i+1, \ldots, n-1$
14. $\tilde{\beta}_{(j-1)n+s}^{(jn+i)} = -\mathbf{q}_{s+1}^H\mathbf{z}_w/c_{(j-1)n+s}$; % $\tilde{\beta}_{(j-1)n+s}^{(jn+i)} = -\omega_j\beta_{(j-1)n+s}^{(jn+i)}$
15. $\mathbf{z}_w = \mathbf{z}_w + \tilde{\beta}_{(j-1)n+s}^{(jn+i)}\mathbf{w}_{(j-1)n+s}$;
16. $\mathbf{g}_{jn+i} = \mathbf{g}_{jn+i} + \tilde{\beta}_{(j-1)n+s}^{(jn+i)}\mathbf{g}_{(j-1)n+s}$;
17. End
18. $\mathbf{g}_{jn+i} = \mathbf{z}_w - \dfrac{1}{\omega_j}\mathbf{g}_{jn+i}$;
19. For $s = 0, \ldots, i-1$
20. $\beta_{jn+s}^{(jn+i)} = -\mathbf{f}_{s+1}^H\mathbf{g}_{jn+i}/c_{jn+s}$;
21. $\mathbf{g}_{jn+i} = \mathbf{g}_{jn+i} + \beta_{jn+s}^{(jn+i)}\mathbf{g}_{jn+s}$;
22. End
23. Else
24. $\beta_{jn}^{(jn+i)} = -\mathbf{f}_1^H\mathbf{r}_{jn+i}/c_{jn}$;
25. $\mathbf{g}_{jn+i} = \mathbf{r}_{jn+i} + \beta_{jn}^{(jn+i)}\mathbf{g}_{jn}$;
26. For $s = 1, \ldots, i-1$

$\beta_{jn+s}^{(jn+i)} = -\mathbf{f}_{s+1}^H \mathbf{g}_{jn+i} / c_{jn+s}$;
$\mathbf{g}_{jn+i} = \mathbf{g}_{jn+i} + \beta_{jn+s}^{(jn+i)} \mathbf{g}_{jn+s}$;
End
End
$\hat{\mathbf{g}}_{jn+i} = \mathbf{M}^{-1}\mathbf{g}_{jn+i}$; $\mathbf{w}_{jn+i} = \mathbf{A}\hat{\mathbf{g}}_{jn+i}$;
$c_{jn+i} = \mathbf{q}_{i+1}^H \mathbf{w}_{jn+i}$;
End
$\alpha_{jn+n} = e_{jn+n-1}/c_{jn+n-1}$;
$\mathbf{x}_{jn+n} = \mathbf{x}_{jn+n-1} + \alpha_{jn+n}\hat{\mathbf{g}}_{jn+n-1}$;
$\mathbf{u}_{jn+n} = \mathbf{r}_{jn+n-1} - \alpha_{jn+n}\mathbf{w}_{jn+n-1}$;
$\hat{\mathbf{u}}_{jn+n} = \mathbf{M}^{-1}\mathbf{u}_{jn+n}$;
$\omega_{j+1} = (\mathbf{A}\hat{\mathbf{u}}_{jn+n})^H \mathbf{u}_{jn+n} / \|\mathbf{A}\hat{\mathbf{u}}_{jn+n}\|_2^2$;
$\mathbf{x}_{jn+n} = \mathbf{x}_{jn+n} + \omega_{j+1}\hat{\mathbf{u}}_{jn+n}$;
$\mathbf{r}_{jn+n} = -\omega_{j+1}\mathbf{A}\hat{\mathbf{u}}_{jn+n} + \mathbf{u}_{jn+n}$;
$e_{jn+n} = \mathbf{q}_1^H \mathbf{r}_{jn+n}$;
$\tilde{\beta}_{(j-1)n+n}^{(jn+n)} = -e_{jn+n}/c_{(j-1)n+n}$; % $\tilde{\beta}_{(j-1)n+n}^{(jn+n)} = -\omega_{j+1}\beta_{(j-1)n+n}^{(jn+n)}$
$\mathbf{z}_w = \mathbf{r}_{jn+n} + \tilde{\beta}_{(j-1)n+n}^{(jn+n)} \mathbf{w}_{(j-1)n+n}$;
$\mathbf{g}_{jn+n} = \tilde{\beta}_{(j-1)n+n}^{(jn+n)} \mathbf{g}_{(j-1)n+n}$;
For $s = 1, \ldots, n-1$
$\tilde{\beta}_{jn+s}^{(jn+n)} = -\mathbf{q}_{s+1}^H \mathbf{z}_w / c_{jn+s}$; % $\tilde{\beta}_{s+jn}^{(jn+n)} = -\omega_{j+1}\beta_{s+jn}^{(jn+n)}$
$\mathbf{z}_w = \mathbf{z}_w + \tilde{\beta}_{jn+s}^{(jn+n)} \mathbf{w}_{jn+s}$;
$\mathbf{g}_{jn+n} = \mathbf{g}_{jn+n} + \tilde{\beta}_{jn+s}^{(jn+n)} \mathbf{g}_{jn+s}$;
End
$\mathbf{g}_{jn+n} = \mathbf{z}_w - \dfrac{1}{\omega_{j+1}}\mathbf{g}_{jn+n}$; $\hat{\mathbf{g}}_{jn+n} = \mathbf{M}^{-1}\mathbf{g}_{jn+n}$;
$\mathbf{w}_{jn+n} = \mathbf{A}\hat{\mathbf{g}}_{jn+n}$; $c_{jn+n} = \mathbf{q}_1^H \mathbf{w}_{jn+n}$;
End

Matlab code of Algorithm A.1

```
function [x, err, iter, flag] = mlbicgstabt(A, x, b, Q, M, max_it, tol, kappa)

% input:  A: N-by-N matrix. M: N-by-N preconditioner matrix.
%         Q: N-by-n shadow matrix [q_1, ..., q_n]. x: initial guess.
%         b: right hand side vector. max_it: maximum number of iterations.
%         tol: error tolerance.
%         kappa:  (real number) minimization step controller:
%                 kappa = 0, standard minimization
%                 kappa > 0, Sleijpen–van der Vorst minimization
% output: x: solution computed. err: error norm. iter: number of iterations performed.
%         flag:   = 0, solution found to tolerance
%                 = 1, no convergence given max_it iterations
%                 = -1, breakdown.
% storage: F: N × (n − 1) matrix. G, Q, W: N × n matrices. A, M: N × N matrices.
%         x, r, g_h, z, b: N × 1 matrices. c: 1 × n matrix.

    N = size(A, 2);  n = size(Q, 2);
    G = zeros(N, n);  W = zeros(N, n);   % initialize work spaces
    if n > 1, F = zeros(N, n − 1); end
    c = zeros(1, n);        % end initialization

    iter = 0;  flag = 1;  bnrm2 = norm(b);
    if bnrm2 == 0.0, bnrm2 = 1.0; end
    r = b − A ∗ x;  err = norm(r)/bnrm2;
    if err < tol, flag = 0; return, end

    if n > 1, F = M′\(A′ ∗ Q(:, 1 : n − 1)); end
    G(:, 1) = r;  g_h = M\r;  W(:, 1) = A ∗ g_h;  c(1) = Q(:, 1)′ ∗ W(:, 1);
    if c(1) == 0, flag = −1; return, end
```

```
e = Q(:, 1)' * r;

for j = 0 : max_it
    for i = 1 : n − 1
        alpha = e/c(i);  x = x + alpha * g_h;  r = r − alpha * W(:, i);
        err = norm(r)/bnrm2;  iter = iter + 1;
        if err < tol, flag = 0;  return, end
        if iter >= max_it, return, end

        e = Q(:, i + 1)' * r;
        if j >= 1
            beta = −e/c(i + 1);
            W(:, i + 1) = r + beta * W(:, i + 1);
            G(:, i + 1) = beta * G(:, i + 1);
            for s = i + 1 : n − 1
                beta = −Q(:, s + 1)' * W(:, i + 1)/c(s + 1);
                W(:, i + 1) = W(:, i + 1) + beta * W(:, s + 1);
                G(:, i + 1) = G(:, i + 1) + beta * G(:, s + 1);
            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
        else
            beta = −F(:, 1)' * r/c(1);  G(:, i + 1) = r + beta * G(:, 1);
            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
        end
        g_h = M\G(:, i + 1);  W(:, i + 1) = A * g_h;
        c(i + 1) = Q(:, i + 1)' * W(:, i + 1);
        if c(i + 1) == 0, flag = −1;  return, end
    end
    alpha = e/c(n);  x = x + alpha * g_h;  r = r − alpha * W(:, n);
    err = norm(r)/bnrm2;
    if err < tol, flag = 0;  iter = iter + 1;  return, end
    g_h = M\r;  z = A * g_h;  omega = z' * z;
    if omega == 0, flag = −1;  return, end
    rho = z' * r;  omega = rho/omega;
    if kappa > 0
        rho = rho/(norm(z) * norm(r));  abs_om = abs(rho);
        if (abs_om < kappa) & (abs_om ~= 0)
            omega = omega * kappa/abs_om;
        end
    end
    if omega == 0, flag = −1;  return, end
    x = x + omega * g_h;  r = r − omega * z;
    err = norm(r)/bnrm2;  iter = iter + 1;
    if err < tol, flag = 0;  return, end
    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);
    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
    G(:, 1) = W(:, 1) − G(:, 1)./omega;  g_h = M\G(:, 1);
```

91. $W(:,1) = A * g\_h$; $c(1) = Q(:,1)' * W(:,1)$;
92. if $c(1) == 0$, $flag = -1$; return, end
93. end

References

[1] M. Yeung, T. Chan, ML(k)BiCGSTAB: a BiCGSTAB variant based on multiple Lanczos starting vectors, SIAM J. Sci. Comput. 21 (4) (1999) 1263–1290.
[2] M. Yeung, ML(n)BiCGStab: refomulation, analysis and implementation, Numer. Math. Theory Methods Appl. 5 (2012) 447–492.
[3] 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) 631–644.
[4] P. Sonneveld, CGS, a fast Lanczos-type solver for nonsymmetric linear systems, SIAM J. Sci. Statist. Comput. 10 (1989) 36–52.
[5] R. Fletcher, Conjugate gradient methods for indefinite systems, in: Lecture Notes Math., vol. 506, Springer-Verlag, Berlin, Heidelberg, New York, 1976, pp. 73–89.
[6] Y. Saad, M.H. Schultz, GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Statist. Comput. 7 (1986) 856–869.
[7] M. Yeung, An introduction to ML(n)BiCGStab, in: Brebbia, Popov (Eds.), Proceedings of Boundary Elements and Other Mesh Reduction Methods XXXIV, WIT Press, 2012, available at http://arxiv.org/abs/1106.3678.
[8] M. Yeung, D. Boley, Transpose-free multiple Lanczos and its application in Padé approximation, J. Comput. Appl. Math. 177/1 (2005) 101–127.
[9] M. Yeung, ML(n)BiCGStabt: a ML(n)BiCGStab variant with A-transpose, in: Proceedings of 2012 Fourth International Conference on Computational and Information Sciences, China, 2012.
[10] Sheng-Xin Zhu, Parallel GPBiCG(m, l) algorithm and preconditioning techniques, MSc. Thesis, Graduate School, China Academy of Engineering Physics, May, 2010 (in Chinese).
[11] S. Fujino, GPBiCG(m, l): a hybird of BiCGSTAB and GPBiCG methods with efficiency and robustness, Appl. Numer. Math. 41 (2002) 107–117.
[12] P. Wesseling, P. Sonneveld, Numerical experiments with a multiple grid and a preconditioned Lanczos type method, in: Lecture Notes in Mathematics, vol. 771, Springer Verlag, Berlin, Heidelberg, New York, 1980, pp. 543–562.
[13] M.H. Gutknecht, Variants of BICGStab for matrices with complex spectrum, SIAM J. Sci. Comput. 14 (1993) 1020–1033.
[14] G.L.G. Sleijpen, D.R. Fokkema, BiCGSTAB(l) for linear equations involving unsymmetric matrices with complex spectrum, ETNA 1 (1993) 11–32.
[15] Shao-Liang Zhang, GPBi-CG: Generalized product-type methods based on Bi-CG for solving nonsymmetric linear systems, SIAM J. Sci. Comput. 18 (1997) 537–551.
[16] P. Sonneveld, 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 (2) (2008) 1035–1062.
[17] M. Gijzen, P. Sonneveld, Algorithm 913: an elegant IDR(s) variant that efficiently exploits bi-orthogonality properties, ACM Trans. Math. Software 38 (2011) 5:1–5:19.
[18] G.L.G. Sleijpen, M.B. van Gijzen, Exploiting BiCGstab(l) strategies to induce dimension reduction, SIAM J. Sci. Comput. 32 (5) (2010) 2687–2709.
[19] M. Tanio, M. Sugihara, GBi-CGSTAB(s, l): IDR(s) with higher-order stabilization polynomials, J. Comput. Appl. Math. 235 (3) (2010) 765–784.
[20] L. Du, T. Sogabe, B. Yu, Y. Yamamoto, S.-L. Zhang, A block IDR(s) method for nonsymmetric linear systems with multiple right-hand sides, J. Comput. Appl. Math. 235 (14) (2011) 4095–4106.
[21] M.H. Gutknecht, IDR explained, ETNA 36 (2010) 126–148.
[22] Martin H. Gutknecht, Jens-Peter M. Zemke, Eigenvalue computations based on IDR, SIAM J. Matrix Anal. Appl. 34 (2) (2013) 283–311.
[23] P. Sonneveld, On the convergence behavior of IDR(s) and related methods, SIAM J. Sci. Comput. 34 (5) (2012) A2576–A2598.
[24] M.H. Gutknecht, Lanczos-type solvers for nonsymmetric linear systems of equations, Acta Numer. 6 (1997) 271–397.
[25] Y. Saad, The Lanczos biorthogonalization algorithm and other oblique projection methods for solving large unsymmetric systems, SIAM J. Numer. Anal. 19 (1982) 485–506.
[26] A. Neumaier, Oral Presentation at the Oberwolfach Meeting Numerical Linear Algebra, Oberwolfach, April 1994.
[27] G.L.G. Sleijpen, H.A. van der Vorst, Reliable updated residuals in hybrid Bi-CG methods, Computing 56 (1996) 141–163.
[28] R. Freund, M. Gutknecht, N. Nachtigal, An implementation of the look-ahead Lanczos algorithm for non-Hermitian matrices, SIAM J. Sci. Comput. 14 (1993) 137–158.
[29] M.H. Gutknecht, A completed theory of the unsymmetric Lanczos process and related algorithms. Part I, SIAM J. Matrix Anal. Appl. 13 (1992) 594–639.
[30] M.H. Gutknecht, A completed theory of the unsymmetric Lanczos process and related algorithms. Part II, SIAM J. Matrix Anal. Appl. 15 (1994) 15–58.
[31] B.N. Parlett, D.R. Taylor, Z.A. Liu, A look-ahead Lanczos algorithm for unsymmetric matrices, Math. Comp. 44 (1985) 105–124.
[32] D. Loher, Reliable nonsymmetric block Lanczos algorithms, Ph.D. Thesis, Swiss Federal Institute of Technology, Zurich, 2006.
[33] 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.
[34] W.D. Joubert, Lanczos methods for the solution of nonsymmetric systems of linear equations, SIAM J. Matrix Anal. Appl. 13 (1992) 926–943.
[35] R.B. Morgan, GMRES with deflated restarting, SIAM J. Sci. Comput. 24 (2002) 20–37.