An introduction to ML(n)BiCGStab

M. C. Yeung
Department of Mathematics, University of Wyoming, USA

Abstract

ML(n)BiCGStab is a Krylov subspace method for the solution of large, sparse and non-symmetric linear systems. In theory, it is a method that lies between the well-known BiCGStab and GMRES/FOM. In fact, when $n = 1$, ML(1)BiCGStab is BiCGStab and when $n = N$, ML(N)BiCGStab is GMRES/FOM where N is the size of the linear system. Therefore, ML(n)BiCGStab is a bridge that connects the Lanczos-based BiCGStab and the Arnoldi-based GMRES/FOM. In computation, ML(n)BiCGStab can be much more stable and converge much faster than BiCGStab when a problem with ill-condition is solved. We have tested ML(n)BiCGStab on the standard oil reservoir simulation test data called SPE9 and found that ML(n)BiCGStab reduced the total computational time by more than 60% when compared to BiCGStab. Tests made on the data from Matrix Market also support the superiority of ML(n)BiCGStab over BiCGStab. Because of the $O(N^2)$ storage requirement in the full GMRES, one has to adopt a restart strategy to get the storage under control when GMRES is implemented. In comparison, ML(n)BiCGStab is a method with only $O(nN)$ storage requirement and therefore it does not need a restart strategy. In this paper, we introduce ML(n)BiCGStab (in particular, a new algorithm involving $\mathbf{A}$-transpose), its relations to some existing methods and its implementations.
Keywords: CGS, BiCGStab, ML(n)BiCGStab, multiple starting Lanczos, Krylov subspace, iterative methods, linear systems.

1 Introduction

ML(n)BiCGStab is a Krylov subspace method for the solution of the linear system

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{1}$$

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


WIT Transactions on Modelling and Simulation, Vol 53, © 2012 WIT Press
www.witpress.com, ISSN 1743-3533 (on-line)
doi:10.2495/BE120291

a natural generalization of BiCGStab by van der Vorst [3], built on the multiple starting Lanczos process rather than on the single starting Lanczos process. Its derivation relies on the techniques introduced by Sonneveld [4] and van der Vorst [3] in the construction of CGS and BiCGStab. There have been three algorithms associated with the ML(n)BiCGStab method so far, depending on how the residual vector $\mathbf{r}_k$ is defined and whether or not the Hermitian transpose $\mathbf{A}^H$ is used. In this paper, we shall simply introduce the algorithms and address some implementation issues. For more detailed, one is referred to [2].

Other extensions of BiCGStab exist. Among them are BiCGStab2 by Gutknecht [5], BiCGStab(l) by Sleijpen and Fokkema [6] and CPBi-CG by Zhang [7].

The outline of the paper is as follows. In Section 2, we introduce index functions which are helpful in presenting the ML(n)BiCGStab algorithms. In Section 3, we present the ML(n)BiCG algorithm from [1], from which ML(n)BiCGStab algorithms were derived. In Section 4, we introduce the ML(n)BiCGStab algorithms and their relationships with some existing methods. In Section 5, implementation issues are addressed and conclusions are made in Section 6.

2 Index functions

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

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

where $\lfloor \cdot \rfloor$ rounds its argument to the nearest integer towards minus infinity. We call g_n and r_n index functions; they are defined on $\mathbb{Z}$, the set of all integers, with ranges $\mathbb{Z}$ and $\{1, 2, \cdots, n\}$, respectively. Table 1 illustrates the behavior of g_n and r_n with $n = 3$.

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

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

3 A ML(n)BiCG algorithm

Parallel to the derivation of BiCGStab from BiCG by Fletcher [8], ML(n)BiCGStab was derived from a BiCG-like method named ML(n)BiCG, which was constructed based on the multiple starting Lanczos process with n left starting vectors and a single right starting vector.

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*. Set

$$\mathbf{p}_k = \left(\mathbf{A}^H\right)^{g_n(k)} \mathbf{q}_{r_n(k)}, \qquad k = 1, 2, 3, \cdots . \tag{2}$$

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

Algorithm 3.1 ML(n)BiCG

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

Even though the algorithm has not been tested, it is believed to be numerically instable because of Line 11 in which the shadow vectors are repeatedly multiplied by $\mathbf{A}^H$, a type of operation which is highly sensitive to round-off errors. The algorithm has been introduced only for the purpose of developing ML(n)BiCGStab algorithms.

Relations to some other methods:

1. *Relation with FOM by Saad and Schultz [9].* Consider the case where $n \geq N$. If we choose $\mathbf{q}_k = \widehat{\mathbf{r}}_{k-1}$ in Algorithm 3.1 (it is possible since $\widehat{\mathbf{r}}_{k-1}$ is computed before $\mathbf{q}_k$ is used in Line 11), then Algorithm 3.1 is a FOM algorithm.
2. *Relation with GMRES by Saad and Schultz [9].* Consider the case where $n \geq N$. If we choose $\mathbf{q}_k = \mathbf{A}\widehat{\mathbf{r}}_{k-1}$ in Algorithm 3.1, then Algorithm 3.1 is a GMRES algorithm.
3. *Relation with BiCG.* When $n = 1$, Algorithm 3.1 is a BiCG algorithm.

4 ML(n)BiCGStab algorithms

There are three algorithms for the ML(n)BiCGStab method. All were derived from Algorithm 3.1. The first two algorithms do not involve $\mathbf{A}^H$ in their implementation and can be found in [2]. The third one, however, needs $\mathbf{A}^H$ and is new. Therefore, we spend more space here on the third algorithm.

4.1 First algorithm

Let $\Omega_k(\lambda)$ be the polynomial of degree k defined by

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

If we define the ML(n)BiCGStab residual $\mathbf{r}_k$ by

$$\mathbf{r}_k = \begin{cases} \Omega_{g_n(k)+1}(\mathbf{A})\,\widehat{\mathbf{r}}_k, & \text{if } k \geq 1, \\ \widehat{\mathbf{r}}_0, & \text{if } k = 0, \end{cases}$$

then Algorithm 3.1 will lead to the first ML(n)BiCGStab algorithm (Algorithm 4.1 in [2]). Computational and storage cost based on its preconditioned version (Algorithm 9.1 in [2]) is presented in Table 2.

Relations to some other methods: this first algorithm is a BiCGStab algorithm when $n = 1$.

Table 2: Average cost per iteration of the first ML(n)BiCGStab algorithm and its storage (besides the storage of $\mathbf{A}$ and $\mathbf{M}$).

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

4.2 Second algorithm

If we define the ML(n)BiCGStab residual $\mathbf{r}_k$ by

$$\mathbf{r}_k = \begin{cases} \Omega_{g_n(k+1)}(\mathbf{A})\,\widehat{\mathbf{r}}_k, & \text{if } k \geq 1, \\ \widehat{\mathbf{r}}_0, & \text{if } k = 0, \end{cases} \tag{3}$$

then Algorithm 3.1 will lead to the second ML(n)BiCGStab algorithm (Algorithm 5.1 in [2]). Computational and storage cost based on its preconditioned version (Algorithm 9.2 in [2]) is presented in Table 3.

Relations to some other methods:

1. *Relation with FOM*. Consider the case where $n \geq N$. If we choose $\mathbf{q}_k = \mathbf{r}_{k-1}$, then this algorithm is a FOM algorithm.
2. *Relation with GMRES*. Consider the case where $n \geq N$. If we choose $\mathbf{q}_k = \mathbf{A}\mathbf{r}_{k-1}$, then this algorithm is a GMRES algorithm.

Table 3: Average cost per iteration of the second ML(n)BiCGStab algorithm and its storage (besides the storage of $\mathbf{A}$ and $\mathbf{M}$).

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

3. *Relation with BiCGStab*. When $n = 1$, this algorithm is a BiCGStab algorithm.
4. *Relation with IDR(s) by Sonneveld and van Gijzen [10, 11]*. This algorithm is a IDR(n) algorithm.

4.3 Third algorithm

If we define the ML(n)BiCGStab residual $\mathbf{r}_k$ by eqn (3) and get $\mathbf{A}^H$ involved in its implementation, then through the derivation stages #5 - #8 in [2], Algorithm 3.1 will lead to the following ML(n)BiCGStab algorithm which we name ML(n)BiCGStabt, standing for ML(n)BiCGStab with $\mathbf{A}$-transpose.

Algorithm 4.1 ML(n)BiCGStabt without preconditioning

Choose an initial guess $\mathbf{x}_0$ *and* n *vectors* $\mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_n$.
Compute $[\mathbf{f}_1, \cdots, \mathbf{f}_{n-1}] = \mathbf{A}^H[\mathbf{q}_1, \cdots, \mathbf{q}_{n-1}]$.
Compute $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$ *and* $\mathbf{g}_0 = \mathbf{r}_0,\ \mathbf{w}_0 = \mathbf{A}\mathbf{g}_0,\ c_0 = \mathbf{q}_1^H\mathbf{w}_0$.
For $k = 1, 2, \cdots$, *until convergence:*
  $\alpha_k = \mathbf{q}_{r_n(k)}^H\mathbf{r}_{k-1}/c_{k-1}$;
  If $r_n(k) < n$
    $\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k\mathbf{g}_{k-1};\ \ \mathbf{r}_k = \mathbf{r}_{k-1} - \alpha_k\mathbf{w}_{k-1}$;
    $\mathbf{z}_w = \mathbf{r}_k;\ \mathbf{g}_k = \mathbf{0}$;
    For $s = \max(k-n, 0), \cdots, g_n(k)n - 1$
      $\tilde{\beta}_s^{(k)} = -\mathbf{q}_{r_n(s+1)}^H\mathbf{z}_w/c_s$; % $\tilde{\beta}_s^{(k)} = -\omega_{g_n(k+1)}\beta_s^{(k)}$
      $\mathbf{z}_w = \mathbf{z}_w + \tilde{\beta}_s^{(k)}\mathbf{w}_s$;
      $\mathbf{g}_k = \mathbf{g}_k + \tilde{\beta}_s^{(k)}\mathbf{g}_s$;
    End
    $\mathbf{g}_k = \mathbf{z}_w - \frac{1}{\omega_{g_n(k+1)}}\mathbf{g}_k$;
    For $s = g_n(k)n, \cdots, k-1$
      $\beta_s^{(k)} = -\mathbf{f}_{r_n(s+1)}^H\mathbf{g}_k/c_s$;
      $\mathbf{g}_k = \mathbf{g}_k + \beta_s^{(k)}\mathbf{g}_s$;
    End
  Else

20. $\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k \mathbf{g}_{k-1}$;
21. $\mathbf{u}_k = \mathbf{r}_{k-1} - \alpha_k \mathbf{w}_{k-1}$;
22. $\omega_{g_n(k+1)} = (\mathbf{A}\mathbf{u}_k)^H \mathbf{u}_k / \|\mathbf{A}\mathbf{u}_k\|_2^2$;
23. $\mathbf{x}_k = \mathbf{x}_k + \omega_{g_n(k+1)} \mathbf{u}_k$; $\mathbf{r}_k = -\omega_{g_n(k+1)} \mathbf{A}\mathbf{u}_k + \mathbf{u}_k$;
24. $\mathbf{z}_w = \mathbf{r}_k$; $\mathbf{g}_k = \mathbf{0}$;
25. *For* $s = g_n(k)n, \cdots, k-1$
26. $\tilde{\beta}_s^{(k)} = -\mathbf{q}_{r_n(s+1)}^H \mathbf{z}_w / c_s$; $\quad \%\ \tilde{\beta}_s^{(k)} = -\omega_{g_n(k+1)} \beta_s^{(k)}$
27. $\mathbf{z}_w = \mathbf{z}_w + \tilde{\beta}_s^{(k)} \mathbf{w}_s$;
28. $\mathbf{g}_k = \mathbf{g}_k + \tilde{\beta}_s^{(k)} \mathbf{g}_s$;
29. *End*
30. $\mathbf{g}_k = \mathbf{z}_w - \frac{1}{\omega_{g_n(k+1)}} \mathbf{g}_k$;
31. *End*
32. $\mathbf{w}_k = \mathbf{A}\mathbf{g}_k$; $c_k = \mathbf{q}_{r_n(k+1)}^H \mathbf{w}_k$;
33. *End*

A preconditioned version of Algorithm 4.1 can be obtained by applying it to $\mathbf{A}\mathbf{M}^{-1}\mathbf{y} = \mathbf{b}$, then recovering $\mathbf{x}$ through $\mathbf{x} = \mathbf{M}^{-1}\mathbf{y}$. The resulting preconditioned algorithm can be found in [12]. In Section 7, we attach its Matlab code. Computational and storage cost based on it is presented in Table 4.

Relations to some other methods: Algorithm 4.1 is a BiCGStab algorithm when $n = 1$.

Table 4: Average cost per iteration of preconditioned ML(n)BiCGStabt and its storage (besides the storage of $\mathbf{A}$ and $\mathbf{M}$). This table does not count the cost in Lines 1-3 of Algorithm 4.1.

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

5 Implementation issues

The following test data were downloaded from Matrix Market (http://math.nist.gov /MatrixMarket/data/). More experiments can be found in [1, 2].

1. *utm5940*, TOKAMAK Nuclear Physics (Plasmas). *utm5940* contains a 5940×5940 real unsymmetric matrix $\mathbf{A}$ with $83,842$ nonzero entries and a real right-hand side $\mathbf{b}$.
2. *qc2534*, H2PLUS Quantum Chemistry, NEP Collection. *qc2534* contains a 2534×2534 complex symmetric indefinite matrix with $463,360$ nonzero entries, but does not provide the right-hand side $\mathbf{b}$. We set $\mathbf{b} = \mathbf{A}\mathbf{1}$ with $\mathbf{1} = [1, , 1, \cdots, 1]^T$.

All computing in this section was done in Matlab Version 7.1 on a Windows XP machine with a Pentium 4 processor. $ILU(0)$ preconditioners [13, p. 294] were used, initial guess was $\mathbf{x}_0 = \mathbf{0}$ and the stopping criterion was $\|\mathbf{r}_k\|_2/\|\mathbf{b}\|_2 < 10^{-7}$ where $\mathbf{r}_k$ was the computed residual. Shadow vectors $\mathbf{Q} = [\mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_n]$ were $\mathbf{Q} = [\mathbf{r}_0, randn(N, n-1)]$ for *utm5940* and $\mathbf{Q} = [\mathbf{r}_0, randn(N, n-1) + sqrt(-1) * randn(N, n-1)]$ for *qc2534*.

For the convenience of our presentation, let us introduce the following functions:

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

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

5.1 Stability

The graphs of $E(n)$ are plotted in Figure 1. It can be seen that the computed relative errors $\|\mathbf{r}_k\|_2/\|\mathbf{b}\|_2$ by the second algorithm can significantly diverge from its exact counterpart $\|\mathbf{b} - \mathbf{A}\mathbf{x}_k\|/\|\mathbf{b}\|_2$. By contrast, the computed $\|\mathbf{r}_k\|_2/\|\mathbf{b}\|_2$ by the first and the third algorithms well approximate their corresponding true ones. Thus, from this point of view, we consider that the first and the third algorithms are numerically more stable than the second algorithm.

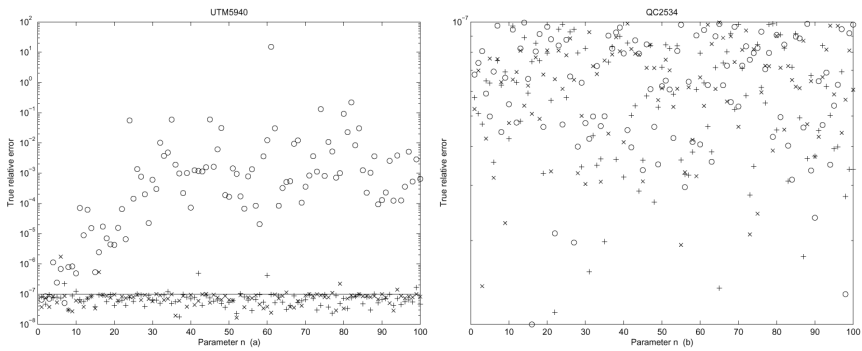

Figure 1: Graphs of $E(n)$ against n. First algorithm: $\times$-mark; Second algorithm: o-mark; Third algorithm: $+$-mark; 10^{-7}: solid line.

5.2 Choice of n

From the experiments in [1, 2], we have observed that ML(n)BiCGStab behaves more and more robust as n is increased. So, for an ill-conditioned problem, we would tend to suggest a large n for ML(n)BiCGStab. On the other hand, ML(n)BiCGStab minimizes $\|\mathbf{r}_k\|_2$ once every n iterations. The convergence of a well-

conditioned problem is usually accelerated by the minimization steps. So, when a problem is well-conditioned, we would suggest a small n.

In [10, 11], it is suggested to fix s at 4 or 8 in the general use of IDR(s). This good idea also applies to ML(n)BiCGStab, namely, fixing $n = 4$ or 8 in its general use.

In the case where a sequence of linear systems is solved, the parameter n can be chosen dynamically based on the information obtained from the solution of previous systems. We once tested the first algorithm (see Algorithm 9.1 in [2]) with $n = 9$ and $\kappa = 0$ (see Section 5.3 for κ) on the standard oil reservoir simulation test data called SPE9 and found that ML(n)BiCGStab reduced the total computational time by over 70% when compared to BiCGStab. A later test on SPE9 with Code #4 in [2] showed that a 60% reduction in time can be reached.

Code #4 is a design of automatic selection of n. Let $walk = 1$ and -1 denote the search directions of increasing n and decreasing n respectively, and let $t1$ and $t2$ be the times of solving the previous and the current systems. Then the basic idea behind the code is: if $walk = 1$ and $t1 > t2$, increase n to $n + step$ for the next system. Similarly, if $walk = 1$ and $t1 < t2$, set $walk = -1$ and decrease n to $n - step$. Here $step$ is the search step size.

We also plot the graphs of $T_{conv}(n)$ in Figure 2 to provide more information on how n affects the performance of ML(n)BiCGStab.

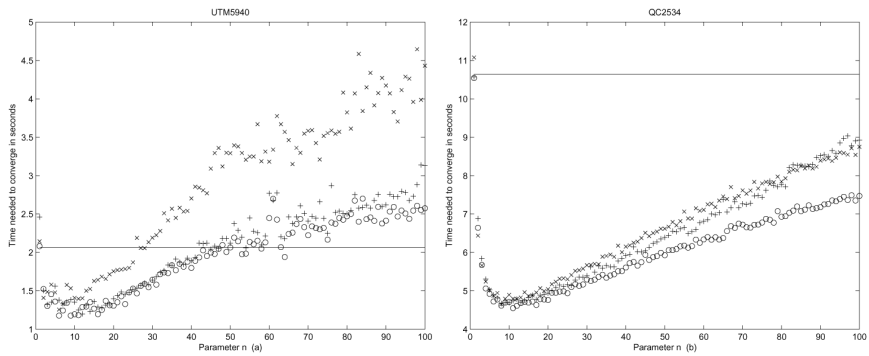

Figure 2: Graphs of $T_{conv}(n)$ against n. First algorithm: $\times$-mark; Second algorithm: o-mark; Third algorithm: $+$-mark; BiCGStab: Solid line. BiCGStab took 2.07 and 10.64 seconds to converge for $utm5940$ and $qc2534$ respectively.

5.3 Choice of ω

The standard choice for the $\omega_{g_n(k+1)}$ in Algorithm 4.1 (see Line 22) is $\omega_{g_n(k+1)} = (\mathbf{A}\mathbf{u}_k)^H \mathbf{u}_k / \|\mathbf{A}\mathbf{u}_k\|_2^2$. This choice of $\omega_{g_n(k+1)}$ minimizes the 2-norm of $\mathbf{r}_k =$

$-\omega_{g_n(k+1)}\mathbf{A}\mathbf{u}_k + \mathbf{u}_k$ (Line 23), but sometimes can cause instability due to that it can be very small during an execution. The following remedy to guard $\omega_{g_n(k+1)}$ away from zero has been proposed by Sleijpen and van der Vorst [14]:

$$\begin{aligned} &\omega_{g_n(k+1)} = (\mathbf{A}\mathbf{u}_k)^H\mathbf{u}_k/\|\mathbf{A}\mathbf{u}_k\|_2^2; \\ &\rho = (\mathbf{A}\mathbf{u}_k)^H\mathbf{u}_k/(\|\mathbf{A}\mathbf{u}_k\|_2\ \|\mathbf{u}_k\|_2); \\ &\text{if } |\rho| < \kappa,\ \ \omega_{g_n(k+1)} = \kappa\omega_{g_n(k+1)}/|\rho|;\ \ \text{end} \end{aligned} \tag{4}$$

where κ is a user-defined parameter. See the numerical experiments in [2, 10] for more information about eqns (4).

6 Conclusions

ML(n)BiCGStab is a powerful Krylov subspace method, especially in the solution of a sequence of linear systems with the parameter n dynamically chosen (see [2] for detail). This method has three algorithms. The first two can be found in [2] and the third is new and is presented here as Algorithm 4.1. The third algorithm involves $\mathbf{A}^H$ in its implementation and behaves as stable as the first algorithm, but converges faster than the first algorithm. Compared to the second algorithm, this third algorithm is more stable, but takes more time to converge.

7 Appendix

Attached below is the Matlab code of the preconditioned version of Algorithm 4.1.

```
1. function [x, err, iter, flag] = mlbicgstabt(A, x, b, Q, M, max_it, tol, kappa)
2.
3. % input:  A: N-by-N matrix. M: N-by-N preconditioner matrix.
4. %        Q: N-by-n shadow matrix [q_1, ..., q_n]. x: initial guess.
5. %        b: right hand side vector. max_it: maximum number of iterations.
6. %        tol: error tolerance.
7. %        kappa:  (real number) minimization step controller:
8. %                kappa = 0, standard minimization
9. %                kappa > 0, Sleijpen-van der Vorst minimization
10. % output: x: solution computed. err: error norm.
    % iter: number of iterations performed.
11. %       flag:   = 0, solution found to tolerance
12. %               = 1, no convergence given max_it iterations
13. %               = -1, breakdown.
14. % storage: F: N x (n-1) matrix. G, Q, W: N x n matrices.
    % A, M: N x N matrices.
15. %       x, r, g_h, z, b: N x 1 matrices. c: 1 x n matrix.
16.
17.     N = size(A,2); n = size(Q,2);
18.     G = zeros(N,n); W = zeros(N,n);   % initialize work spaces
19.     if n > 1, F = zeros(N,n-1); end
20.     c = zeros(1,n);        % end initialization
```

```
21.
22.   iter = 0;  flag = 1;  bnrm2 = norm(b);
23.   if bnrm2 == 0.0, bnrm2 = 1.0; end
24.   r = b − A ∗ x;  err = norm(r)/bnrm2;
25.   if err < tol,  flag = 0;  return, end
26.
27.   if n > 1, F = M′\(A′ ∗ Q(:, 1 : n − 1)); end
28.   G(:, 1) = r;  g_h = M\r;  W(:, 1) = A ∗ g_h;  c(1) = Q(:, 1)′ ∗ W(:, 1);
29.   if c(1) == 0,  flag = −1;  return, end
30.   e = Q(:, 1)′ ∗ r;
31.
32.   for j = 0 : max_it
33.       for i = 1 : n − 1
34.           alpha = e/c(i);  x = x + alpha ∗ g_h;  r = r − alpha ∗ W(:, i);
35.           err = norm(r)/bnrm2;  iter = iter + 1;
36.           if err < tol,  flag = 0;  return, end
37.           if iter >= max_it, return, end
38.
39.           e = Q(:, i + 1)′ ∗ r;
40.           if j >= 1
41.               beta = −e/c(i + 1);
42.               W(:, i + 1) = r + beta ∗ W(:, i + 1);
43.               G(:, i + 1) = beta ∗ G(:, i + 1);
44.               for s = i + 1 : n − 1
45.                   beta = −Q(:, s + 1)′ ∗ W(:, i + 1)/c(s + 1);
46.                   W(:, i + 1) = W(:, i + 1) + beta ∗ W(:, s + 1);
47.                   G(:, i + 1) = G(:, i + 1) + beta ∗ G(:, s + 1);
48.               end
49.               G(:, i + 1) = W(:, i + 1) − G(:, i + 1)./omega;
50.               for s = 0 : i − 1
51.                   beta = −F(:, s + 1)′ ∗ G(:, i + 1)/c(s + 1);
52.                   G(:, i + 1) = G(:, i + 1) + beta ∗ 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_h = M\G(:, i + 1);  W(:, i + 1) = A ∗ g_h;
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_h;  r = r − alpha ∗ W(:, n);
66.       err = norm(r)/bnrm2;
67.       if err < tol,  flag = 0;  iter = iter + 1;  return, end
68.       g_h = M\r;  z = A ∗ g_h;  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_h; r = r - omega * z;
79.       err = norm(r)/bnrm2; iter = iter + 1;
80.       if err < tol, flag = 0; return, end
81.       if iter >= max_it, return, end
82.
83.       e = 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_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
```

A sample execution of ML(n)BiCGstabt

1. $N = 100$; $A = randn(N)$; $M = randn(N)$; $b = randn(N, 1)$;
2. $n = 10$; $kappa = 0.7$; $tol = 10^{-7}$; $max\_it = 3 * N$;
3. $Q = randn(N, n)$; $x = zeros(N, 1)$; $Q(:, 1) = b - A * x$;
4. $[x, err, iter, flag] = mlbicgstabt(A, x, b, Q, M, max\_it, tol, kappa)$;

References

[1] Yeung, M. and Chan, T., *ML(k)BiCGSTAB: A BiCGSTAB variant based on multiple Lanczos starting vectors*, SIAM J. Sci. Comput., Vol. 21, No. 4, pp. 1263-1290, 1999.

[2] Yeung, M., *ML(n)BiCGStab: Reformulation, Analysis and Implementation*, submitted to Numerical Mathematics: Theory, Methods and Applications. Available at http://www.uwyo.edu/mathmyeung/r17.pdf.

[3] van der Vorst, H.A., *Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems,* SIAM J. Sci. Statist. Comput., 12 (1992), pp. 631–644.

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

[5] Gutknecht, M.H., *Variants of BiCGStab for matrices with complex spectrum*, SIAM J. Sci. Comput. 14, 1020-1033, 1993.

[6] Sleijpen, G.L.G. and Fokkema, D.R., *BiCGSTAB(l) for linear equations involving unsymmetric matrices with complex spectrum,* ETNA, 1:11-32, 1993.
[7] Zhang, S.L., *GPBi-CG: Generalized product-type methods based on Bi-CG for solving nonsymmetric linear systems*, SIAM J. Sci. Comput., 18:537-551, 1997.
[8] Fletcher, R., *Conjugate gradient methods for indefinite systems,* volume 506 of Lecture Notes Math., pages 73-89. Springer-Verlag, Berlin-Heidelberg-New York, 1976.
[9] Saad, Y. and Schultz, M.H., *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems,* SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869.
[10] Sonneveld P. and van Gijzen, M., *IDR(s): a family of simple and fast algorithms for solving large nonsymmetric linear systems,* SIAM J. Sci. Comput. Vol. 31, No. 2, pp. 1035-1062.
[11] van Gijzen, M. and Sonneveld, P., *An elegant IDR(s) variant that efficiently exploits bi-orthogonality properties*, Delft University of Technology, Reports of the Department of Applied Mathematical Analysis, Report 08-21.
[12] Yeung, M., *An introduction to ML(n)BiCGStab*, available at http://arxiv.org /abs/1106.3678.
[13] Saad, Y., *Iterative methods for sparse linear systems*, 2nd edition, SIAM, Philadelphia, PA, 2003.
[14] Sleijpen, G.L.G. and van der Vorst, H.A., *Maintaining convergence properties of BiCGSTAB methods in finite precision arithmetic,* Numer. Algorithms, 10 (1995), pp. 203–223.