Electronic Transactions on Numerical Analysis.
Volume 1, pp. 11-32, September 1993.
Copyright © 1993, Kent State University.
ISSN 1068-9613.

ETNA
Kent State University
etna@mcs.kent.edu

BICGSTAB(L) FOR LINEAR EQUATIONS INVOLVING UNSYMMETRIC MATRICES WITH COMPLEX SPECTRUM*

GERARD L.G. SLEIJPEN† AND DIEDERIK R. FOKKEMA†

Abstract. For a number of linear systems of equations arising from realistic problems, using the Bi-CGSTAB algorithm of van der Vorst [17] to solve these equations is very attractive. Unfortunately, for a large class of equations, where, for instance, Bi-CG performs well, the convergence of Bi-CGSTAB stagnates. This was observed specifically in case of discretized advection dominated PDE's. The stagnation is due to the fact that for this type of equations the matrix has almost pure imaginary eigenvalues. With his BiCGStab2 algorithm Gutknecht [5] attempted to avoid this stagnation. Here, we generalize the Bi-CGSTAB algorithm further, and overcome some shortcomings of BiCGStab2. In some sense, the new algorithm combines GMRES(l) and Bi-CG and profits from both.

Key words. Bi-conjugate gradients, non-symmetric linear systems, CGS, Bi-CGSTAB, iterative solvers, GMRES, Krylov subspace.

AMS subject classification. 65F10.

1. Introduction.

The bi-conjugate gradient method (Bi-CG) [2, 8] solves iteratively equations

$$Ax = b \tag{1.1}$$

in which A is some given non-singular unsymmetric $n \times n$ matrix and b some given n-vector. Typically n is large and A is sparse. We will assume A and b to be real, but our methods are easily generalized to the complex case. In each iteration step, the approximation x_k is corrected by some search correction that depends on the true residual r_k ($r_k = b - Ax_k$) and some "shadow residual" $\tilde{r}_k$. The residuals r_k are "forced to converge" by making r_k orthogonal to the shadow residuals $\tilde{r}_j$ for $j < k$. Any iteration step requires a multiplication by A to produce the next true residual and a multiplication by A^T (the real transpose of A) to produce the next shadow residual. This strategy involves short recursions and hence an iteration step is cheap with respect to the computational cost (except for the matrix multiplications) and memory requirement. In addition to the MVs (i.e. matrix-vector multiplications), a few DOTs (inner products) and AXPYs (vector updates) are required, and apart from the x_k, four other vectors have to be stored.

Bi-CG seems like an ideal algorithm but in practice it has a few disadvantages.

- The transpose (either complex or real) of A is often not (easy) available.
- Although the computational cost is low in terms of AXPYs and DOTs, each step requires two matrix multiplications, which is double the cost of CG.
- Bi-CG may suffer from breakdown. This can be repaired by look-ahead strategies [1, 4]. We will not consider the breakdown situation for Bi-CG in this paper.
- Bi-CG often converges irregularly. In finite precision arithmetic, this irregular behavior may slow down the speed of convergence.

In [15] Sonneveld observed that the computational effort to produce the shadow residuals could as well be used to obtain an additional reduction of the Bi-CG residuals

* This research was supported in part by a NCF/Cray Research University Grant CRG 92.03. Received March 24, 1993. Accepted for publication July 16, 1993. Communicated by M. Gutknecht.

† Mathematical Institute, Utrecht University, P.O.Box 80.010, NL-3508 TA Utrecht, The Netherlands.

ETNA
Kent State University
etna@mcs.kent.edu

r_k. His CGS algorithm computes approximations $\mathbf{x}_k$ with a residual of the form $\mathbf{r}_k = q_k(A)r_k$, where q_k is some appropriate polynomial of degree k. The $\mathbf{r}_k$ are computed explicitly, while the polynomials q_k and the Bi-CG residuals r_k play only a theoretical role. One step of the CGS algorithm requires two multiplications by A and no multiplication at all by the transpose of A. The computational complexity and the amount of memory is comparable to that of Bi-CG. In case $q_k(A)$ gives an additional reduction, CGS is an attractive method [15]. Unfortunately, in many situations, the CGS choice for q_k leads to amplifications of r_k instead of reduction. This causes irregular convergence or even divergence and makes the method more sensitive to evaluation errors [17, 16].

Van der Vorst [17] proposes to take for q_k a product of appropriate 1-step MR-polynomials (Minimal Residual polynomials), i.e. degree one polynomials of the form $1 - \omega_k t$ for some optimal ω_k. To a large extend, this choice fulfills the promises: for many problems, his Bi-CGSTAB algorithm converges rather smoothly and also often faster than Bi-CG and CGS. In such cases $q_k(A)$ reduces the residual significantly, while the Bi-CGSTAB iteration steps are only slightly more expensive than the CGS steps.

However, ω_k may be close to zero, and this may cause stagnation or even breakdown. As numerical experiments confirm, this is likely to happen if A is real and has nonreal eigenvalues with an imaginary part that is large relative to the real part. One may expect that second degree MR-polynomials can better handle this situation. In [5] Gutknecht introduces a BiCGStab2 algorithm that employs such second degree polynomials. Although this algorithm is certainly an improvement in many cases, it may still suffer from problems in cases where Bi-CGSTAB stagnates or breaks down. At every second step, Gutknecht corrects the first degree MR-polynomial from the previous step to a second degree MR-polynomial. However, in the odd steps, the problem of a nearly degenerate MR-polynomial of degree one may already have occurred (this is comparable to the situation where GCR breaks down while GMRES (or Orthodir) proceeds nicely (cf. [12]). In BiCGStab2 (as well as in the other methods CGS, Bi-CGSTAB and the more general method BiCGstab(l), to be introduced below), the Bi-CG iteration coefficients play a crucial role in the computation. If, in an odd step, the MR polynomial almost degenerates, the next second degree polynomial as well as the Bi-CG iteration coefficients may be polluted by large errors and this may affect the process severely.

In this paper, we introduce the BiCGstab(l) algorithm. For $l = 1$, this algorithm coincides with Bi-CGSTAB. In BiCGstab(l), the polynomial q_k is chosen as the product of l-step MR-polynomials: for $k = ml + l$ we take

$$(1.2)\qquad \begin{array}{l} q_k = q_{ml+l} = p_m p_{m-1} \dots p_0, \text{ where the } p_i\text{'s are of degree } l,\ p_i(0) = 1 \\ \text{and } p_m \text{ minimizes } \|p_m(A) q_{k-l}(A) r_k\|_2. \end{array}$$

We form an l-degree MR-polynomial p_m after each l-th step. In the intermediate steps $k = ml + i$, $i = 1, \dots, l-1$, we employ simple factors t^i and the p_m are reconstructed from these powers. In this way, we can avoid certain near-breakdowns in these steps. Near-breakdown may still occur in our approach if the leading coefficient of p_m is almost 0. However, second degree or more general even degree polynomials seem to be well suited for complex eigenpairs and near-breakdown is hardly a problem in practice (although it may occur if, for instance, A is a cyclic matrix: $Ae_i = e_{i-1}$ for $i = 2, 3, \dots$). On the other hand, BiCGstab(l) still incorporates the breakdown dangers of Bi-CG.

```
$k = -1$,
    choose $x_0$ and $\tilde{r}_0$,
    compute $r_0 = b - Ax_0$,  take $u_{-1} = \tilde{u}_{-1} = 0$,  $\rho_{-1} = 1$,
repeat until $\|r_{k+1}\|$ is small enough:
$k := k + 1$,
    $\rho_k = (r_k, \tilde{r}_k)$,  $\beta_k = -\rho_k/\rho_{k-1}$,
    $u_k = r_k - \beta_k u_{k-1}$,  $c_k = Au_k$,
    $\tilde{u}_k = \tilde{r}_k - \beta_k \tilde{u}_{k-1}$,
    $\gamma_k = (c_k, \tilde{r}_k)$,  $\alpha_k = \rho_k/\gamma_k$,
    $x_{k+1} = x_k + \alpha_k u_k$,  $r_{k+1} = r_k - \alpha_k c_k$,  $\tilde{r}_{k+1} = \tilde{r}_k - \alpha_k A^T \tilde{u}_k$
```

ALGORITHM 2.1. *The Bi-CG algorithm.*

— In exact arithmetic, if Gutknecht's version BiCGStab2 does not break down, it produces the same result as our BiCGstab(2). In actual computation the results can be quite different. Our version proceeds nicely as should be expected from BiCGstab(2) also in cases where BiCGStab2 stagnates due to the MR-choice in the odd steps. In cases where Gutknecht version does well, our version seems to converge slightly faster. In some cases in finite precision arithmetic, the approximations $\mathbf{x}_k$ and the residuals $\mathbf{r}_k$ drift apart (i.e. $b - A\mathbf{x}_k \not\approx \mathbf{r}_k$), due to irregular convergence behavior of the underlying Bi-CG process. Gutknecht's algorithm seems to be significantly more sensitive to this effect than ours.
— In addition the steps of our version are cheaper with respect to both computational cost as well as memory requirement: except for the number of MVs, which is the same for both versions, our version is about 33% less expensive and it needs about 10% less memory space.
— Gutknecht's approach can also be used to construct a BiCGstab(l) version. However, if l increases, the formulas and the resulting algorithm will become increasingly more complicated, while we have virtually the same algorithm for every l. We can easily increase l if stagnation threatens.
— In some situations it may be profitable to take $l > 2$. Although the steps of BiCGstab(l) are more expensive for larger l, numerical experiments indicate that, in certain situations, due to a faster convergence, for instance, BiCGstab(4) performs better than BiCGstab(2). Our BiCGstab(l) algorithm combines the advantages of both Bi-CG and GMRES(l) and seems to converge faster than any of those.

In the next section, we give theoretical details on the above observations. Section 3 contains a detailed description of the BiCGstab(l) algorithm and its derivation. In addition, it contains comments on the implementation, the computational costs and the memory requirement. We conclude section 3 with a number of possible variants for BiCGstab(l). In section 4 we give some remarks on preconditioning. In the last section, we present some numerical experiments.

2. Theoretical justification of BiCGstab(l).

The Bi-CG algorithm [2, 8] in ALGORITHM 2.1 solves iteratively the linear equation (1.1). One has to select some initial approximation x_0 for x and some "shadow" residual $\tilde{r}_0$. Then the Bi-CG algorithm produces iteratively sequences of approximations x_k, residuals r_k and search directions u_k by

$$u_k = r_k - \beta_k u_{k-1}, \quad x_{k+1} = x_k + \alpha_k u_k, \quad r_{k+1} = r_k - \alpha_k A u_k \tag{2.1}$$

(where $u_{-1} = 0$, and r_0 is computed by $r_0 = b - Ax_0$). The scalars α_k and β_k are computed such that both Au_k and r_k are orthogonal to the Krylov subspace $\mathcal{K}_k(A^T; \tilde{r}_0)$ of order k, spanned by the vectors $\tilde{r}_0$, $A^T\tilde{r}_0$,$\ldots$,$(A^T)^{k-1}\tilde{r}_0$.

By induction it follows that both u_k and r_k belong to the Krylov subspace $\mathcal{K}_{k+1}(A; r_0)$. Moreover, $r_k = \phi_k(A)r_0$, for some ϕ_k in the space $\mathcal{P}_k^1$ of all polynomials p of degree k for which $p(0) = 1$. Since, in the generic case, by increasing k the "shadow" Krylov subspaces $\mathcal{K}_k(A^T; \tilde{r}_0)$ "fill" the complete space, the sequence of $\|r_k\|$ may be expected to decrease. The vector r_k is the unique element of the form $p(A)r_0$, with $p \in \mathcal{P}_k^1$ that is orthogonal to $\mathcal{K}_k(A^T; \tilde{r}_0)$: in some weak sense, p is the best polynomial in $\mathcal{P}_k^1$.

Consider some sequence of polynomials q_k of exact degree k. The vectors Au_k and r_k are orthogonal to $\mathcal{K}_k(A^T; \tilde{r}_0)$ if and only if these vectors are orthogonal to $\tilde{s}_j = q_j(A^T)\tilde{r}_0$ for all $j = 0, \ldots, k-1$. Now, as we will see in 3.1,

$$\beta_k = \theta_k \frac{(r_k, \tilde{s}_k)}{(r_{k-1}, \tilde{s}_{k-1})} \quad \text{and} \quad \alpha_k = \frac{(r_k, \tilde{s}_k)}{(Au_k, \tilde{s}_k)} \tag{2.2}$$

in which θ_k is a scalar that depends on the leading coefficients of the polynomials ϕ_j and q_j for $j = k, k-1$. The Bi-CG algorithm takes $q_k = \phi_k$ (and $\theta_k = 1$), so that the $\tilde{s}_j$ are computed by the same recursions as for the r_j (see ALGORITHM 2.1, where for the choice $q_k = \phi_k$, we use the notation $\tilde{r}_k$ instead of $\tilde{s}_k$). However, in exact arithmetic, for any choice of q_k the same approximations x_k, residuals r_k and search directions u_k can be constructed.

Algorithms as CGS, Bi-CGSTAB and BiCGstab(l) are based on the observation that

$$(r_k, \tilde{s}_k) = (r_k, q_k(A^T)\tilde{r}_0) = (q_k(A)r_k, \tilde{r}_0) \quad \text{and} \quad (Au_k, \tilde{s}_k) = (Aq_k(A)u_k, \tilde{r}_0). \tag{2.3}$$

In the ideal case the operator $\phi_k(A)$ reduces r_0. One may try to select q_k such that $Q_k = q_k(A)$ additionally reduces r_k as much as possible. In such a case it would be an advantage to avoid the computation of r_k and u_k, and one might try to compute immediately $\mathbf{r}_k = Q_k r_k$, $\mathbf{u}_k = Q_k u_k$, and the associated approximation $\mathbf{x}_k$ by appropriate recursions. As (2.2) and (2.3) show, one can use these vectors to compute the Bi-CG iteration coefficients β_k and α_k (for more details, see 3.1). If the polynomials q_k are related by simple recursions, these recursions can be used to compute efficiently the iterates $\mathbf{r}_k$, $\mathbf{u}_k$ and $\mathbf{x}_k$ without computing r_k and u_k explicitly. Therefore, the computational effort of the Bi-CG algorithm to build the shadow Krylov subspace can also be used to obtain an additional reduction (by Q_k) for the Bi-CG residual r_k. Since r_{2k} would have been constructed from the "weakly best" polynomial in $\mathcal{P}_{2k}^1$, we may not expect that $\|\mathbf{r}_k\| \ll \|r_{2k}\|$: which says that a method based on Q_k can only converge twice as fast (in terms of costs). Since a Bi-CG step involves two MVs it only makes sense to compute $\mathbf{r}_k$ instead of r_k if we can obtain $\mathbf{r}_k$ from $\mathbf{r}_{k-1}$ by 4 MVs at most and a few vector updates and if we can update simultaneously the corresponding approximation $\mathbf{x}_k$, where $\mathbf{r}_k = b - A\mathbf{x}_k$. This has been realized in CGS and Bi-CGSTAB (these algorithms require 2 MVs per step):
— The choice

$$q_k = \phi_k \tag{2.4}$$

leads to the CGS (Conjugate Gradient-squared) algorithm of Sonneveld [15]. Since $\phi_k(A)$ is constructed to reduce r_0 as much as possible, one may not expect that $\phi_k(A)$

reduces r_k as well. Actually, for a large class of problems $\phi_k(A)$ often transforms the r_k to a sequence of residuals $\mathbf{r}_k$ that converges very irregularly or even diverges [16]. — In [17], van der Vorst attempted to repair this irregular convergence behavior of CGS by choosing

$$(2.5)\quad \begin{array}{l} q_k(t) = (1-\omega_k t)q_{k-1}(t) \quad \text{with } \omega_k \quad \text{such that} \\ \|(I-\omega A)q_{k-1}(A)r_k\|_2 \text{ is minimal with respect to the scalar } \omega \text{ for } \omega = \omega_k. \end{array}$$

In fact this is a special case of our algorithm, namely $l = 1$.

Unfortunately, for matrix-vector equations with real coefficients, ω_k is real as well. This may lead to a poor reduction of $\widehat{r} = q_{k-1}(A)r_k$, i.e. $\|\mathbf{r}_k\| \approx \|\widehat{r}\|$, where $\mathbf{r}_k = (I-\omega_k A)\widehat{r}$. The convergence of Bi-CGSTAB may even stagnate (and actually does, cf. experiments in [9]). The Bi-CG iteration coefficients α_k and β_k can not be computed from $\mathbf{r}_k$ (see (2.3)) if the polynomial q_k is not of exact degree k. This happens if $\omega_k = 0$. Likewise, $\omega_k \approx 0$ may be expected to lead to inaccurate Bi-CG iteration coefficients. Consequently, stagnation in the Minimal Residual stage ($\omega_k \approx 0$) may cause breakdown or poor convergence of Bi-CGSTAB. One may come across an almost zero ω_k if the matrix has non-real eigenvalues λ with relatively large imaginary parts. If the components of $\widehat{r}$ in the direction of the associated eigenvectors are relatively large then the best reduction by $I - \omega_k A$ is obtained for $\omega_k \approx 0$. This fatal behavior of Bi-CGSTAB may be "cured" as follows.

Select some $l \geq 2$. For $k = ml + l$, take

$$(2.6)\quad \begin{array}{l} q_k = p_m q_{k-l}, \text{ where } p_m \text{ is a polynomial of degree } l,\ p_m(0) = 1 \\ \text{such that } \|p(A)q_{k-l}(A)r_k\|_2 \text{ is minimal with respect to } p \in \mathcal{P}_l^1 \text{ for } p = p_m \end{array}$$

and where q_{k-l} is the product of MR-polynomials in $\mathcal{P}_l^1$ constructed in previous steps. In the intermediate steps, $k = ml + i$, $i = 1, \ldots, l-1$, we take $q_k = q_{ml}$ to compute the residual $\mathbf{r}_k = \mathbf{r}_{ml+i} = q_{ml}(A)r_k$ and search direction $\mathbf{u}_k = q_{ml}(A)u_k$. We use $t^i q_{ml}$ to compute the Bi-CG iteration coefficients through $(A^i\mathbf{r}_k, \tilde{r}_0)$ and $(A^{i+1}\mathbf{u}_k, \tilde{r}_0)$ (cf. (2.3) and (2.2)). This choice leads to the **BiCGstab(l) algorithm** in 3.2. A pseudo code for the algorithm is given in ALGORITHM 3.1.

So, only for $k = ml$, do the polynomials q_k that we use belong to $\mathcal{P}_k^1$. In the intermediate steps we employ two types of polynomials, polynomials of exact degree k (the $t^i q_{ml}$) and polynomials that are 1 in 0 (the q_{ml}).

The vectors $\mathbf{r}_k$, $\mathbf{u}_k$, $A^i\mathbf{r}_k$ and $A^{i+1}\mathbf{u}_k$ can be computed efficiently. However, we are interested in approximations $\mathbf{x}_k$ and not primarily in the residual $\mathbf{r}_k$. These approximations can easily be computed as a side-product: if $\mathbf{r}_{k+1} = \mathbf{r}_k - Aw$ then $\mathbf{x}_{k+1} = \mathbf{x}_k + w$. Whenever we update $\mathbf{r}_k$ by some vecto u we have its original under $A^{-1}u$ as well. The polynomials used in the algorithm ensure that this is possible.

The residuals $\mathbf{r}_k$ in the intermediate steps $k = ml + i$ will not be optimal in $\mathcal{K}_{k+k}(A; r_0)$. They cannot, because they belong to $\mathcal{K}_{k+ml}(A; r_0)$. Although the BiCGstab($l$) algorithm produces approximations and residuals also in the intermediate steps, the approximations and residuals of interest are only computed every l-th step.

In the BiCGstab(l) algorithm we have that $\mathbf{r}_k = q_k(A)\phi_k(A)r_0$. For $k = ml$ the reduction operator $q_k(A)\phi_k(A)$ that acts on r_0 is the product of the Bi-CG reduction operator and a GMRES(l)-like reduction operator: $q_k(A)$ is the product of a sequence of GMRES reduction operators of degree l (or, equivalently, of l-step Minimal Residual operators; see [12]). Note that $q_k(A)$ is not the operator that would be obtained by applying k steps of GMRES(l) to the residual $\phi_k(A)r_0$ of k steps Bi-CG. After each l

ETNA
Kent State University
etna@mcs.kent.edu

steps of Bi-CG we apply an l-step Minimal Residual step and accumulate the effect. Nevertheless BiCGstab(l) seems to combine the nice properties of both methods. If GMRES stagnates in the first l steps then typically GMRES(l) does not make any progress later. By restarting, the process builds approximately the same Krylov subspace as before the restart, thus encountering the same point of stagnation. This is avoided in the BiCGstab(l) process where the convergence may keep on going by the incorporated Bi-CG process. The residual $\widehat{r} = q_{k-l}(A)r_k$ may differ significantly from the residual $q_{k-l}(A)r_{k-l}$. Therefore, each "Minimal Residual phase" in BiCGstab(l) in general has a complete "new" starting residual, while, in case of stagnation, at each new start, GMRES(l) employs about the same starting residual, keeping stagnating for a long while.

3. The BiCGstab(l) algorithm.

Let u_k, c_k, r_k be the vectors as produced by Bi-CG and let α_k, β_k be the Bi-CG iteration coefficients, $c_k = Au_k$ (see ALGORITHM 2.1).

3.1. The computation of the Bi-CG iteration coefficients.

Consider the Bi-CG method in ALGORITHM 2.1.

We are not really interested in the shadow residuals $\tilde{r}_k$ nor in the shadow search directions $\tilde{u}_k$. Actually, as observed in section 2, we only need $\tilde{r}_k$ to compute β_k and α_k through the scalars $(r_k, \tilde{r}_k)$ and $(c_k, \tilde{r}_k)$. We can compute these scalars by means of any vector $\tilde{s}_k$ of the form $\tilde{s}_k = q_k(A^T)\tilde{r}_0$ where q_k is some polynomial in $\mathcal{P}_k$ of which the leading coefficient is non-trivial and known.

To prove this, suppose $q_k \in \mathcal{P}_k$ has non-trivial leading coefficient σ_k, that is $q_k(t) - \sigma_k t^k$ is a polynomial in $\mathcal{P}_{k-1}$. Note that $\tilde{r}_k = \phi_k(A^T)\tilde{r}_0$ for the Bi-CG polynomial $\phi_k \in \mathcal{P}_k^1$ and that $\phi_k(t) - \tau_k t^k$ belongs to $\mathcal{P}_{k-1}$ for $\tau_k = (-\alpha_{k-1})(-\alpha_{k-2})\ldots(-\alpha_0)$. Hence, both the vectors $\tilde{r}_k - \tau_k(A^T)^k\tilde{r}_0$ and $\tilde{s}_k - \sigma_k(A^T)^k\tilde{r}_0$ belong to $\mathcal{K}_k(A^T;\tilde{r}_0)$. Since both the vectors r_k and c_k are orthogonal to this space (see [2]), we have that

$$(r_k, \tilde{r}_k) = \frac{\tau_k}{\sigma_k}(r_k, \tilde{s}_k) \quad \text{and} \quad (c_k, \tilde{r}_k) = \frac{\tau_k}{\sigma_k}(c_k, \tilde{s}_k). \tag{3.1}$$

Hence,

$$\beta_k = -\frac{(r_k, \tilde{r}_k)}{(r_{k-1}, \tilde{r}_{k-1})} = -\frac{\tau_k \sigma_{k-1}}{\sigma_k \tau_{k-1}} \frac{(r_k, \tilde{s}_k)}{(r_{k-1}, \tilde{s}_{k-1})} = \alpha_{k-1}\frac{\sigma_{k-1}}{\sigma_k}\frac{(r_k, \tilde{s}_k)}{(r_{k-1}, \tilde{s}_{k-1})} \tag{3.2}$$

and

$$\alpha_k = \frac{(r_k, \tilde{r}_k)}{(c_k, \tilde{r}_k)} = \frac{(r_k, \tilde{s}_k)}{(c_k, \tilde{s}_k)}. \tag{3.3}$$

Using (2.3) it even follows that we do not need $\tilde{s}_k$. With $\mathbf{r}_k = q_k(A)r_k$ and $\mathbf{c}_k = q_k(A)c_k$, we have that

$$(r_k, \tilde{s}_k) = (\mathbf{r}_k, \tilde{r}_0) \quad \text{and} \quad (c_k, \tilde{s}_k) = (\mathbf{c}_k, \tilde{r}_0). \tag{3.4}$$

Therefore, we can compute the Bi-CG iteration coefficients α_k and β_k by means of $\mathbf{r}_k$ and $\mathbf{c}_k$. We do not need the $\tilde{r}_k$, $\tilde{u}_k$, nor $\tilde{s}_k$.

3.2. The construction of the BiCGstab(l) algorithm.

The Bi-CG vectors u_k, c_k, r_k are only computed implicitly — they only play a role in the derivation of the algorithm — while the Bi-CG iteration coefficients α_k, β_k are computed explicitly

— they are explicitly needed in the computation.[1] Instead of the Bi-CG vectors, for certain indices $k = lm$, we compute explicitly vectors $\mathbf{u}_{k-1}$, $\mathbf{r}_k$ and $\mathbf{x}_k$: $\mathbf{x}_k$ is the approximate solution with residual $\mathbf{r}_k$ and $\mathbf{u}_{k-1}$ is a search direction.

The BiCGstab(l) algorithm (ALGORITHM 3.1) iteratively computes $\mathbf{u}_{k-1}$, $\mathbf{x}_k$ and $\mathbf{r}_k$ for $k = l, 2l, 3l, \ldots$. These steps are called outer iteration steps. One single outer step, in which we proceed from $k = ml$ to $k = ml + l$, consists of an inner iteration process. In the first half of this inner iteration process (the "Bi-CG part") we implicitly compute new Bi-CG vectors. In the second half (the "MR part") we construct by a Minimal Residual approach a locally minimal residual.

We describe the BiCGstab(l) algorithm by specifying one inner loop.

Suppose, for $k = ml$ and for some polynomial $q_k \in \mathcal{P}_k$ with $q_k(0) = 1$ we have computed $\mathbf{u}_{k-1}$, $\mathbf{r}_k$ and $\mathbf{x}_k$ such that

$$\mathbf{u}_{k-1} = Q_k u_{k-1}, \;\; \mathbf{r}_k = Q_k r_k \;\; \text{and} \;\; \mathbf{x}_k \;\; \text{where} \;\; Q_k = q_k(A). \tag{3.5}$$

The steps of the inner loop may be represented by a triangular scheme (SCHEME 3.1) where the steps for the computation of residuals and search directions are indicated for $l = 2$ (and $k = m2$). The computation proceeds from row to row, replacing vectors from the previous row by vectors on the next row. In SCHEME 3.1 vector updates derived from the Bi-CG relations (2.1) are indicated by arrows. For instance, in the transition from the first row to the second one, we use $Q_k u_k = Q_k(r_k - \beta_k u_{k-1}) = Q_k r_k - \beta_k Q_k u_{k-1}$ and to get from the second row to the third, we use $Q_k r_{k+1} = Q_k(r_k - \alpha_k A u_k) = Q_k r_k - \alpha_k A Q_k u_k$. The computation involves other vector updates as well (in the MR part); these are not represented. Vectors that are obtained by multiplication by the matrix A are framed. The column at the left edge represents iteration coefficients computed according to (3.2)–(3.4) before replacing the old row by the new one. Recall that $\mathbf{c}_k = A Q_k u_k$. The scheme does not show how the approximations for x are updated from row to row. However, their computation is analogous to the update of the residuals $Q_k r_{k+j}$: when a residual is updated by adding a vector of the form $-Aw$, the approximation for x is updated by adding the vector w.

In the Bi-CG part, we construct the next row from the previous rows by means of the Bi-CG recursions (2.1) and matrix multiplication. The fifth row, for instance, is computed as follows: since $r_{k+2} = r_{k+1} - \alpha_{k+1} A u_{k+1}$ we have that

$$Q_k r_{k+2} = Q_k r_{k+1} - \alpha_{k+1} A Q_k u_{k+1} \quad \text{and} \quad A Q_k r_{k+2} = A Q_k r_{k+1} - \alpha_{k+1} A^2 Q_k u_{k+1}.$$

By multiplying $A Q_k r_{k+2}$ by A we compute the vector $A^2 Q_k r_{k+2}$ on the diagonal of the scheme. After $2l$ rows we have the vectors $A^i Q_k r_{k+l}$ and $A^i Q_k u_{k+l-1}$ $(i = 0, \ldots, l)$.

In the MR part, we combine these vectors $A^i Q_k r_{k+l}$ to find the **minimal residual** $\mathbf{r}_{k+l}$. This vector is the residual in the best approximation of $Q_k r_{k+l}$ in the Krylov subspace $\mathcal{K}_{l-1}(A; A Q_k r_{k+l})$. The computation of the scalars γ_i needed for this linear combination is done by the modified Gram-Schmidt orthogonalization process. The γ_i and $A^i Q_k u_{k+l-1}$ lead to $\mathbf{u}_{k+l-1}$:

$$\mathbf{r}_{k+l} = Q_k r_{k+l} - \sum_{i=1}^{l} \gamma_i A^i Q_k r_{k+l} \quad \text{and} \quad \mathbf{u}_{k+l-1} = Q_k u_{k+l-1} - \sum_{i=1}^{l} \gamma_i A^i Q_k u_{k+l-1}.$$

[1] Moreover, even if they were not needed in the computation, it could be worthwhile to compute them: from these coefficients one can easily compute the representation of the matrix of A with respect to the basis of the Bi-CG vectors c_i. This matrix is tri-diagonal and enables us to compute cheaply approximations (Ritz values) of the eigenvalues of A.

ETNA
Kent State University
etna@mcs.kent.edu

$$\begin{tikzcd}
 & Q_k u_{k-1} \arrow[dd] & Q_k r_k \arrow[ddl] & & & & & \\
\beta_k & & & & & & & \\
 & Q_k u_k & Q_k r_k \arrow[dd] & \boxed{AQ_k u_k} \arrow[ddl] & & & & \mathrm{B} \\
\alpha_k & & & & & & & \mathrm{I} \\
 & Q_k u_k \arrow[dd] & Q_k r_{k+1} \arrow[ddl] & AQ_k u_k \arrow[dd] & \boxed{AQ_k r_{k+1}} \arrow[ddl] & & & | \\
\beta_{k+1} & & & & & & & \mathrm{C} \\
 & Q_k u_{k+1} & Q_k r_{k+1} \arrow[dd] & AQ_k u_{k+1} \arrow[ddl] & AQ_k r_{k+1} \arrow[dd] & \boxed{A^2 Q_k u_{k+1}} \arrow[ddl] & & \mathrm{G} \\
\alpha_{k+1} & & & & & & & \\
 & Q_k u_{k+1} & Q_k r_{k+2} & AQ_k u_{k+1} & AQ_k r_{k+2} & A^2 Q_k u_{k+1} & \boxed{A^2 Q_k r_{k+2}} & \\
 & Q_k u_{k+1} & Q_k r_{k+2} & AQ_k u_{k+1} & AQ_k r_{k+2} & A^2 Q_k u_{k+1} & A^2 Q_k r_{k+2} & \mathrm{M} \\
\gamma_1, \gamma_2 & & & & & & & \mathrm{R} \\
 & Q_{k+2} u_{k+1} & Q_{k+2} r_{k+2} & & & & &
\end{tikzcd}$$

SCHEME 3.1. *The computational schema for BiCGstab(2).*

In this part we determine from a theoretical point of view the polynomial $p_m(t) = 1 - \gamma_1 t - \ldots - \gamma_l t^l$. Implicitly we update q_k to find q_{k+l}. We emphasize that we do not use complicated polynomials until we arrive at the last row of the scheme where we form implicitly p_m.

The vectors in the second column of vectors, the $Q_k r_{k+j}$ ($\widehat{r}_0$ in the algorithm and in the detailed explanation below) are also residuals (corresponding to $\widehat{x}_0$ in the algorithm). The vectors $A^j Q_k u_{k+j-1}$ and $A^j Q_k r_{k+j}$ along the diagonal are used in the computation of the Bi-CG iteration coefficients α_{k+j} and β_{k+j+1}. The computation of all these "diagonal vectors" require also $2l$ multiplications by A (MVs). Note that l steps of the Bi-CG algorithm require also $2l$ multiplications by some matrix: l multiplications by A and another l by A^T. As indicated above, the other vectors $A^i Q_k r_{k+j}$ and $A^i Q_k u_{k+j-1}$ $(i = 0, \ldots, j-1)$ in the triangular schemes can cheaply be constructed from vector updates: we obtain the vectors $Q_k r_{k+l}, \ldots, A^{l-1} Q_k r_{k+l}$ as a by-product. Consequently, the last step in the inner loop can easily be executed. The vector $\mathbf{r}_{k+l}$ is the minimal residual of the form $p(A) Q_k r_{k+l}$, where $p \in \mathcal{P}_l^1$. In exact arithmetic, l steps of GMRES starting with the residual $\widehat{r}_0 = Q_k r_{k+l}$ would yield the same residual $\mathbf{r}_{k+l}$ (see [12]). However, GMRES would require l MVs to compute this projection. For stability reasons, GMRES avoids the explicit computation of vectors of the form $A^i \widehat{r}_0$. Since we keep the value for l low (less than 8), our approach does not seem to give additional stability problems besides the ones already encountered in the Bi-CG process. We trade a possible instability for efficiency (see also 3.6).

We now give details on the Bi-CG part, justify the computation of the Bi-CG iteration coefficients and discuss the MR part. In the MR part, we compute $\mathbf{u}_{k+l-1}$, $\mathbf{r}_{k+l}$ and $\mathbf{x}_{k+l}$.

The Bi-CG part. In this part, in l steps, we compute iteratively $A^i Q_k u_{k+l-1}$, $A^i Q_k r_{k+l}$ $(i = 0, \ldots, l)$, the approximation $\widehat{x}_0$ for which $b - A\widehat{x}_0 = Q_k r_{k+l}$, the Bi-CG iteration coefficients α_{k+j}, β_{k+j} $(j = 0, \ldots, l-1)$ and an additional scalar ρ_0. We start our computation with the vectors $\widehat{u}_0 = \mathbf{u}_{k-1} = Q_k u_{k-1}$, $\widehat{r}_0 = \mathbf{r}_k = Q_k r_k$ and $\widehat{x}_0 = \mathbf{x}_k$, the scalar α_{k-1} and some scalars ρ_0 and ω from the previous step; $-\omega$ is the leading coefficient of p_{m-1}.

Suppose after j-steps, we have

$$\begin{aligned}&\widehat{u}_i = A^i Q_k u_{k+j-1}, \quad \widehat{r}_i = A^i Q_k r_{k+j} \quad (i = 0, \ldots, j),\\ &\widehat{x}_0 \quad \text{such that} \quad \widehat{r}_0 = b - A\widehat{x}_0,\\ &\alpha = \alpha_{k+j-1} \quad \text{and} \quad \rho_0 = (\widehat{r}_{j-1}, \tilde{r}_0).\end{aligned}$$

Note that

$$\widehat{u}_i = A^{i-1} Q_k A u_{k+j-1} = A^{i-1} Q_k c_{k+j-1}.$$

Then the $(j+1)$-th step proceeds as follows (the "old" vectors "$\widehat{u}$", "$\widehat{r}$" and $\widehat{x}_0$ may be replaced by the new ones. For clarity of explanation we label the new vectors with a $'$). Below, we comment on the computation of α_{k+j} and β_{k+j}.

$$\begin{aligned}&\rho_1 = (\widehat{r}_j, \tilde{r}_0) = (A^j Q_k r_{k+j}, \tilde{r}_0), \quad \beta = \beta_{k+j} = \alpha \tfrac{\rho_1}{\rho_0}, \quad \rho_0 = \rho_1,\\ &\widehat{u}'_i = A^i Q_k u_{k+j} = A^i Q_k (r_{k+j} - \beta_{k+j} u_{k+j-1}) = \widehat{r}_i - \beta\, \widehat{u}_i \qquad (i = 0, \ldots, j),\\ &\widehat{u}'_{j+1} = A\widehat{u}'_j \qquad \text{(multiplication by } A),\\ &\gamma = (\widehat{u}'_{j+1}, \tilde{r}_0) = (A^j Q_k c_{k+j}, \tilde{r}_0), \quad \alpha = \alpha_{k+j} = \tfrac{\rho_0}{\gamma},\\ \\ &\widehat{r}'_i = A^i Q_k r_{k+j+1} = A^i Q_k (r_{k+j} - \alpha_{k+j} c_{k+j}) = \widehat{r}_i - \alpha\, \widehat{u}'_{i+1} \qquad (i = 0, \ldots, j),\\ &\widehat{r}'_{j+1} = A\widehat{r}'_j \qquad \text{(multiplication by } A),\\ \\ &\widehat{x}'_0 = \widehat{x}_0 + \alpha\, \widehat{u}'_0 \qquad (b - A\widehat{x}'_0 = \widehat{r}_0 - \alpha\, A\widehat{u}'_0 = \widehat{r}_0 - \alpha\, \widehat{u}'_1 = \widehat{r}'_0).\end{aligned}$$

Now, drop the $'$ and repeat this step for $j = 0, \ldots, l-1$.

The computation of the Bi-CG iteration coefficients. Consider some $j \in \{0, \ldots, l-1\}$. We define $\gamma = (A^j Q_k c_{k+j}, \tilde{r}_0)$ and $\rho_1 = (A^j Q_k r_{k+j}, \tilde{r}_0)$.

For $j = 0$, let $\rho_0 = (A^{l-1} Q_{k-l} r_{k-1}, \tilde{r}_0)$. The leading coefficient of $q_k(t)$ is equal to the leading coefficient of $t^{l-1} q_{k-l}(t)$ times $-\omega_{m-1}$, where $-\omega_{m-1}$ is the leading coefficient of the "MR polynomial" p_{m-1} for which $q_k = p_{m-1} q_{k-l}$. Hence, by (3.2), (3.3) and (3.4), we have that

$$\beta_k = -\frac{\alpha_{k-1}}{\omega_{m-1}} \frac{\rho_1}{\rho_0} \quad \text{and} \quad \alpha_k = \frac{\rho_1}{\gamma}.$$

In case $j > 0$, let $\rho_0 = (A^{j-1} Q_k r_{k+j-1}, \tilde{r}_0)$. Now, the polynomials $t^j q_k(t)$ and $t^{j-1} q_k(t)$ have the same leading coefficient. Therefore, again by (3.2), (3.3) and (3.4), we have that

$$\beta_{k+j} = \alpha_{k+j-1} \frac{\rho_1}{\rho_0} \quad \text{and} \quad \alpha_{k+j} = \frac{\rho_1}{\gamma}.$$

The MR part. Suppose $\widehat{x}_0$, $\widehat{r}_j$, $\widehat{u}_j$ are known for $j = 0, \ldots, l$ such that

$$\widehat{r}_0 = b - A\widehat{x}_0 \quad \text{and} \quad \widehat{r}_j = A\widehat{r}_{j-1}, \quad \widehat{u}_j = A\widehat{u}_{j-1} \quad (j = 1, \ldots, l)$$

(as after the l steps of the Bi-CG part).

Let $\sum_{j=1}^{l} \gamma_j \widehat{r}_j$ the orthogonal projection of $\widehat{r}_0$ onto the span of $\widehat{r}_1, \ldots, \widehat{r}_l$. With $\quad p_m(t) = 1 - \gamma_1 t - \ldots - \gamma_l t^l \quad (t \in \mathbf{R}) \quad$ we have that

$$\mathbf{r}_{k+l} = \widehat{r}_0 - \sum_{j=1}^{l} \gamma_j \widehat{r}_j = p_m(A)\widehat{r}_0 = p_m(A) Q_k r_{k+l} = Q_{k+l} r_{k+l}.$$

Further,

$$\mathbf{u}_{k+l-1} = \widehat{u}_0 - \sum_{j=1}^{l} \gamma_j \widehat{u}_j = p_m(A)\widehat{u}_0 = Q_{k+l}u_{k+l-1}$$

and

$$\mathbf{x}_{k+l} = \widehat{x}_0 + \sum_{j=1}^{l} \gamma_j \widehat{r}_{j-1}.$$

We wish to compute these quantities as efficient as possible.

The orthogonal vectors $q_1, \ldots, q_l$ are computed by modified Gram-Schmidt from $\widehat{r}_1, \ldots, \widehat{r}_l$; the arrays for $\widehat{r}_1, \ldots, \widehat{r}_l$ may be used to store $q_1, \ldots, q_l$.
For ease of discussion, we consider the $n \times l$ matrices R, Q and U for which

$$Re_j = \widehat{r}_j, \quad Qe_j = q_j, \quad Ue_j = \widehat{u}_j \quad \text{for} \quad j = 1, \ldots, l.$$

Moreover, we consider the $l \times l$ matrices T, D, S, where T is the upper triangular matrix for which $R = QT$, D is the diagonal matrix given by $Q^T Q = D$, and S is given by $Se_1 = 0$ and $Se_j = e_{j-1} \quad (j = 2, \ldots, l)$. If $\vec{\gamma} \in \mathbf{R}^l$ minimizes

$$\|\widehat{r}_0 - R\vec{\gamma}\,\|_2 = \|\widehat{r}_0 - QT\vec{\gamma}\,\|_2,$$

or equivalently, $\vec{\gamma}$ is the least square solution of $QT\vec{\gamma} = \widehat{r}_0$, then $\quad \vec{\gamma} = T^{-1}D^{-1}Q^T\widehat{r}_0$ and

$$\begin{aligned}
\mathbf{r}_{k+l} &= \widehat{r}_0 - R\vec{\gamma} = \widehat{r}_0 - QD^{-1}Q^T\widehat{r}_0, \\
\mathbf{u}_{k+l-1} &= \widehat{u}_0 - U\vec{\gamma}, \\
\mathbf{x}_{k+l} &= \widehat{x}_0 + \gamma_1\widehat{r}_0 + RS\vec{\gamma} = \widehat{x}_0 + \gamma_1\widehat{r}_0 + QTS\vec{\gamma}.
\end{aligned}$$

or, with

$$\vec{\gamma}' = D^{-1}Q^T\widehat{r}_0, \quad \vec{\gamma} = T^{-1}\vec{\gamma}' \quad \text{and} \quad \vec{\gamma}'' = TS\vec{\gamma},$$

we have

$$\mathbf{r}_{k+l} = \widehat{r}_0 - Q\vec{\gamma}', \ \mathbf{u}_{k+l-1} = \widehat{u}_0 - U\vec{\gamma}, \ \mathbf{x}_{k+l} = \widehat{x}_0 + \gamma_1\widehat{r}_0 + Q\vec{\gamma}''.$$

Since $-\gamma_l$ is the leading coefficient of the polynomial p_m we have $\omega_m = \gamma_l \quad$ (ω in the algorithm).

In the algorithm we use the same arrays for $\widehat{r}_j$ and q_j. Therefore, q_j is written as $\widehat{r}_j$ in the algorithm.

Remark. In Bi-CG as well as in several other iterative methods the vector Au_k is a scalar multiple of $r_{k+1} - r_k$. Unfortunately, $A\mathbf{u}_{k+l-1}$ is not a multiple of $\mathbf{r}_{k+l} - \mathbf{r}_k$ nor of $\mathbf{r}_{k+l} - Q_k r_{k+l}$ which would facilitate the computation of $A\mathbf{u}_{k+l-1}$. For similar reasons one can not save on the costs of the computation of $\mathbf{u}_{k+l-1}$, unless one is willing to rearrange the Bi-CG part (see [13] or our discussion in 3.5).

```
$k = -l$,
    choose $x_0$, $\tilde{r}_0$,
    compute $\mathbf{r}_0 = b - Ax_0$,
    take $\mathbf{u}_{-1} = 0$, $\mathbf{x}_0 = x_0$, $\rho_0 = 1$, $\alpha = 0$, $\omega = 1$.
repeat until $\|\mathbf{r}_{k+l}\|$ is small enough
$k = k + l$,
    Put $\widehat{u}_0 = \mathbf{u}_{k-1}$, $\widehat{r}_0 = \mathbf{r}_k$ and $\widehat{x}_0 = \mathbf{x}_k$.
    $\rho_0 = -\omega\rho_0$

    For $j = 0, \ldots, l-1$                                  (Bi-CG PART)
        $\rho_1 = (\widehat{r}_j, \tilde{r}_0)$, $\beta = \beta_{k+j} = \alpha\frac{\rho_1}{\rho_0}$, $\rho_0 = \rho_1$
        For $i = 0, \ldots, j$
            $\widehat{u}_i = \widehat{r}_i - \beta\widehat{u}_i$
        end
        $\widehat{u}_{j+1} = A\widehat{u}_j$
        $\gamma = (\widehat{u}_{j+1}, \tilde{r}_0)$, $\alpha = \alpha_{k+j} = \frac{\rho_0}{\gamma}$
        For $i = 0, \ldots, j$
            $\widehat{r}_i = \widehat{r}_i - \alpha\widehat{u}_{i+1}$
        end
        $\widehat{r}_{j+1} = A\widehat{r}_j$, $\widehat{x}_0 = \widehat{x}_0 + \alpha\widehat{u}_0$
    end

    For $j = 1, \ldots, l$                           (mod.G–S)   (MR PART)
        For $i = 1, \ldots, j-1$
            $\tau_{ij} = \frac{1}{\sigma_i}(\widehat{r}_j, \widehat{r}_i)$
            $\widehat{r}_j = \widehat{r}_j - \tau_{ij}\widehat{r}_i$
        end
        $\sigma_j = (\widehat{r}_j, \widehat{r}_j)$, $\gamma'_j = \frac{1}{\sigma_j}(\widehat{r}_0, \widehat{r}_j)$
    end

    $\gamma_l = \gamma'_l$,    $\omega = \gamma_l$
    For $j = l-1, \ldots, 1$                           $(\vec{\gamma} = T^{-1}\vec{\gamma}')$
        $\gamma_j = \gamma'_j - \sum_{i=j+1}^{l} \tau_{ji}\gamma_i$
    end
    For $j = 1, \ldots, l-1$                           $(\vec{\gamma}'' = TS\vec{\gamma})$
        $\gamma''_j = \gamma_{j+1} + \sum_{i=j+1}^{l-1} \tau_{ji}\gamma_{i+1}$
    end

    $\widehat{x}_0 = \widehat{x}_0 + \gamma_1\widehat{r}_0$, $\widehat{r}_0 = \widehat{r}_0 - \gamma'_l\widehat{r}_l$, $\widehat{u}_0 = \widehat{u}_0 - \gamma_l\widehat{u}_l$    (update)
    For $j = 1, \ldots, l-1$
        $\widehat{u}_0 = \widehat{u}_0 - \gamma_j\widehat{u}_j$              BLAS2    or   BLAS3
        $\widehat{x}_0 = \widehat{x}_0 + \gamma''_j\widehat{r}_j$              _GEMV          _GEMM
        $\widehat{r}_0 = \widehat{r}_0 - \gamma'_j\widehat{r}_j$
    end

    Put $\mathbf{u}_{k+l-1} = \widehat{u}_0$, $\mathbf{r}_{k+l} = \widehat{r}_0$ and $\mathbf{x}_{k+l} = \widehat{x}_0$.
```

ALGORITHM 3.1. *The BiCGstab(l) algorithm.*

3.3. The computational cost and memory requirements. BiCGstab(l) as well as for instance GMRES(l) and CGS are Krylov subspace methods. These methods compute iteratively a sequence (x_k) (or (x_{ml})) of approximations of x for which, for every k, x_k belongs to the k-dimensional Krylov subspace $\mathcal{K}_k(A; r_0)$ (or $x_{ml} \in \mathcal{K}_{ml}(A; r_0)$ for every m; actually, the approximation $x_k - x_0$ of $x - x_0$ belongs to this Krylov subspace. Without loss of generality we assume $x_0 = 0$). The success of such a method depends on

- its capability to find a good (best) approximation in the Krylov subspace $\mathcal{K}_k(A, r_0)$ (also in the presence of evaluation errors),
- the efficiency to compute the next approximation x_{k+1} from the ones of the previous step(s),
- the memory space that is required to store the vectors that are needed for the computation.

For none of the methods, are all the conditions optimally fulfilled (unless the linear problem to be solved is symmetric, or otherwise nice). For instance, in some sense GMRES finds the best approximation in the Krylov subspace (it finds the approximation with the smallest residual), but the steps are increasingly expensive in computational cost as well as in memory requirement. Bi-CG proceeds efficiently from step to step, but it does not find the best approximation. This makes it hard to compare the methods of this type analytically.

It is hard to get access to the convergence behavior, to its capability to find good approximations of x. Nevertheless one can easily investigate the computational cost per iteration step, which we will do now. Note that some methods do not aim to find a good approximation in Krylov subspaces of all dimensions; CGS and Bi-CGSTAB head for even dimensions, while the BiCGstab(l) approximation $\mathbf{x}_k$ is only computed for $k = ml$, for $\mathbf{x}_{ml} \in \mathcal{K}_{2ml}(A; r_0)$. In addition the computational cost may vary from step to step, as is the case for GMRES(l) and BiCGstab(l). For these reasons we give the average costs to increase the dimension of the approximating Krylov subspace by one. If, for a certain linear system, the methods we wish to compare are all able to find an equally good approximation in the Krylov subspace of interest, then this average cost represents the overall efficiency of the methods well. If some less efficient method finds better approximations, then it depends on the number of iteration steps which one is the best. We assume that the problem size n is large and therefore that the costs of small vector operations (involving vectors of dimension l) are negligible.

In Table 3.1 we list the computational cost and the memory requirements for a number of Krylov subspace methods. GMRESR(l, m) was introduced in [19] (see also [11]); in this modification of GMRES(l) (or, more appropriate, of GCR(l)), GMRES(m) is used as a preconditioner. GMRES(l) as well as GMRESR(l, m) avoid excessive use of memory by restarting after a certain number of steps.

The column "computational costs" contains the average amount of large vector operations that is needed to increase the dimension of the relevant Krylov subspace by one.
Furthermore, the table shows the maximum number of n-vectors that have to be stored during the computation; we do not count the locations needed to store the matrix (but our count includes b, $\tilde{r}_0$, $\mathbf{x}_k$ and $\mathbf{r}_k$).

3.4. Remarks on the implementation of the algorithm. (a) Actually we only introduced the vectors $\mathbf{u}_{k-1}$, $\mathbf{x}_k$ and $\mathbf{r}_k$ for ease of presentation. Neither of them have to be stored: they can overwrite $\widehat{u}_0$, $\widehat{x}_0$ and $\widehat{r}_0$.
(b) The computation of $\widehat{u}_0$ in the MR part of Algorithm 3.1 involves a number of

TABLE 3.1
The average cost per Krylov dimension.

METHOD	COMPUTATIONAL COSTS			MEMORY
	MV(S)	AXPY	DOT	REQUIREMENTS
Bi-CG	2	6.5	2	7
CGS	1	3.25	1	7
Bi-CGSTAB	1	3	2	7
BiCGStab2	1	5.5	2.75	10
BiCGstab(2)	1	3.75	2.25	9
BiCGstab(l)	1	$0.75(l+3)$	$0.25(l+7)$	$2l+5$
GMRES(l)	1	$\approx 0.5(l+3)$	$\approx 0.5(l+1)$	$l+3$
GMRESR(l,m)	1	$\approx g_{l,m}+1$	$\approx g_{m,l}$	$m+2l+4$

where $g_{l,m} = 0.5(m+3) + (l+2)/(2m)$

The algorithm BiCGStab2 [5], may be improved slightly. For instance, it computes certain vectors in each step while it suffices to compute them in the even steps only. Our list above is based on the improved algorithm.

vector updates. In order to restrict memory traffic, it is worth postponing the updates to the end and to combine them in the next Bi-CG part (when $j, i = 0$) where $\widehat{u}_0$ has to be updated again. A similar remark applies to the final update of $\widehat{x}_0$ in the Bi-CG part and the first update of $\widehat{x}_0$ in the MR part. One can also gain computational speed by computing inner products together with the appropriate vector updates or matrix multiplications. For instance ρ_1 in the Bi-CG part can be computed in combination with the last vector update for $\widehat{r}_0$ in the MR part.
(c) The final updates in the MR part should be implemented using the BLAS2 subroutine _GEMV (or _GEMM from BLAS3) instead of the BLAS1 subroutine _AXPY. Depending on the computer architecture this will improve efficiency.
(d) Another change in the algorithm that would reduce significantly the amount of work involves the modified Gram-Schmidt process. One can use the generalized inverse of R in order to compute the necessary coefficients γ_i for the MR-polynomial (see "The MR part" in 3.2). More precisely one can compute these coefficients from the normal equations as $\vec{\gamma} = (R^T R)^{-1} R^T \widehat{r}_0$. Now we do not have to compute an orthogonal basis for range(R) and we have saved $(l-1)/4$ vector updates per Krylov dimension.

This approach not only reduces the total amount of work, but it also makes the algorithm more suitable for *parallel* implementation.

However, when l is large this approach may be more unstable than the one based on modified Gram-Schmidt. Consequently one might not expect to obtain the best possible reduction for $\widehat{r}_0$. We discuss this variant and we analyze its stability in [13].

3.5. Variants. Many variants of the above process are feasible. We will mention only a few. For a detailed discussion, numerical experiments and conclusions, we refer to [13].

Dynamic choice of l. For a number of problems the BiCGstab(l) algorithm (with $l > 1$) converges about as fast as the BiCGstab(1) algorithm, i.e. the average reduction per step of the residuals in one algorithm is comparable to the reduction in the other. In such a case it is more efficient to work with $l = 1$, since for larger l the average cost per step is higher. However, particularly if BiCGstab(1) stagnates (if $\omega \approx 0$), one should take an l larger than 1. It may be advantageous not to fix l at the start of the process, but to choose l for each new inner loop depending on information from the previous inner loop.

If for larger l a significant reduction can be obtained locally, it is also worth switching to larger l.

It is not obvious how the switch can be realized and what the correct switching criterion would be. We further discuss this issue in [13]. The switch that we will discuss there is based on a BiCGstab(l) algorithm in which, in the inner loop, the MR part and the Bi-CG part are reversed. This costs slightly more memory and vector updates, but it facilitates the selection of an appropriate l before any stagnation or breakdown occurs.

Bi-CG combined with some polynomial iteration. The MR part does not require any additional MVs but it needs quite a number of AXPYs and DOTs due to the orthogonalization process. If one knows the coefficients γ_i of the polynomial $p_m(t) = 1 - \sum_{j=1}^{l} \gamma_j t^j$, then one can skip the orthogonalization. Unfortunately, the optimal γ_j will not be known a priori, but one might hope that the γ_j from previous steps work as well (at least for a number of consecutive steps). Further, since the Bi-CG iteration coefficients provide information on the spectrum of A, one might use this information to construct a shifted Chebychev polynomial of degree l and take this for p_m. Of course, one may update the polynomial in each step. Note that the construction of the Chebychev polynomial does not involve extra operations with n-vectors.

Bi-CG combined with standard GMRES or Bi-CG. Instead of computing $\mathbf{r}_{k+l}$ by correcting $\widehat{r}_0$ with some explicit linear combination of vectors $A^j\widehat{r}_0$ as we do, one can apply l steps of standard GMRES with starting residual $\widehat{r}_0$. This approach would require l MVs to obtain $\mathbf{r}_{k+l}$. One has to compute the γ_j from the GMRES results in order to construct $\mathbf{u}_{k+l-1}$ (see also the remark on the MR part in 3.2). If one decides to pursue this last approach, one can save l MVs and a number of AXPYs and DOTs in the Bi-CG part as follows. The Bi-CG iteration coefficients α_{k+j} and β_{k+j} can also be computed from the vectors $AQ_k u_{k+j}$, $Q_k r_{k+j-1}$, $Q_k r_{k+j}$ (the $\widehat{u}_1$, $\widehat{r}_0$ in the algorithm) and the shadow residuals $\tilde{r}_{j-1}$, $\tilde{r}_j$. Instead of building a triangular scheme of residuals and search directions (see SCHEME 3.1) one can stick to a scheme of three columns of $Q_k u_{k+j}$, $Q_k r_{k+j}$, $AQ_k u_{k+j}$. The shadow residuals $\tilde{r}_1, \ldots, \tilde{r}_{l-1}$ need only be computed and stored once.

If these shadow residuals are available, it is tempting to apply l steps of Bi-CG to compute $\mathbf{r}_{k+l}$ starting with $Q_k r_{k+l}$. This saves a number of AXPYs and DOTs in the "MR part". The search direction $\mathbf{u}_{k+l-1}$ has also to be computed. This can be done without additional MVs: from the Bi-CG relations (2.1) it follows that the $A^j\mathbf{u}_{k+l-1}$ are linear combination of $A^i Q_k r_{k+l}$ and $AQ_k u_{k+l-i}$ $(i = 1, \ldots, j)$. The scalars in the linear combination can be expressed in terms of the Bi-CG iteration coefficients β_{k+j}, α_{k+j}. Hence, $\mathbf{u}_{k+l-1}$ can be computed by updating $Q_k u_{k+l-1}$ using the previously computed vectors $AQ_k u_{k+l-j}$ and $A^j Q_k r_{k+l}$ $(j = 1, \ldots, l)$.

3.6. The stability. We obtain $\mathbf{r}_{k+l}$ by subtracting some explicit linear combination of vectors $A^j\widehat{r}_0$ from $\widehat{r}_0$. One may object that this approach might be unstable especially if l is not small. However, we restrict ourselves to small l ($l \leq 8$). Our strategy resembles somewhat the look ahead strategy in the Lanczos algorithms in [4, 6, 7]. In our numerical experiments the convergence did not seem to suffer from such instabilities. On the contrary, the residual reduction of BiCGstab(l) proceeds

more smoothly than those of Bi-CG or CGS. Bi-CG and l-step MR seem to improve their mutual stability. The following two observations may help to understand why the γ_i may be affected by non-small errors without spoiling the convergence.
— The polynomial p_m must be non-degenerate (the contribution $\gamma_l A^l \widehat{r}_0$ should be significant also in finite precision arithmetic) and the same scalars should be used to update the residual, the search direction and the approximation. The *Bi-CG part* does not impose other restrictions on the γ_i used in the actual computation.
— In the *MR part*, any reduction is welcome even it is not the optimal one.

As an alternative to our approach, one may gain stability by computing $\mathbf{r}_{k+l}$ by l steps of GMRES with starting residual $\widehat{r}_0$ (see the "GMRES variant" in 3.5). One also has to keep track of the search directions. Since this GMRES stability "cure" is directed towards the residuals it is not clear whether this approach would improve the stability of the computation of the search directions. We return to these stability questions in [13].

In [5], for $l = 2$, Gutknecht "avoids" the instability caused by working with the "naive" basis for the Krylov subspace. He computes $\mathbf{r}_{k+l}$ from $\widehat{r}_0$ by a GCR-type method (in his algorithm the GCR part and the Bi-CG part are intertwined), thus incorporating the breakdown dangers of GCR.

4. The preconditioned BiCGstab(l) algorithm.

Let H_0 be a preconditioning matrix. Instead of solving $Ax = b$, one may as well solve

$$H_0 A = H_0 b \tag{4.1}$$

or

$$A H_0 y = b \quad \text{with} \quad x = H_0 y. \tag{4.2}$$

Therefore, by replacing A by $H_0 A$ and $\mathbf{r}_0 = b - Ax_0$ by $\mathbf{r}_0 = H_0(b - Ax_0)$, we have an algorithm that solves (4.1) iteratively. In this case the computed residuals $\mathbf{r}_k$ are not the real residuals (even not in exact arithmetic) but $\mathbf{r}_k = H_0(b - A\mathbf{x}_k)$.

By replacing A by AH_0 and $\mathbf{x}_0 = x_0$ by $\mathbf{x}_0 = H_0^{-1} x_0$, we have an algorithm that solves iteratively (4.2). The computed residuals are the real ones, that is, $\mathbf{r}_k = b - AH_0\mathbf{x}_k$, but now $\mathbf{x}_k$ is not the approximation we are interested in: we would like to have $H_0\mathbf{x}_k$ instead. If we do not want to monitor the approximations of the exact solution x, it suffices to compute $H_0\mathbf{x}_k$ only after termination.

In both variants, the BiCGstab(l) algorithm may converge faster, due to the preconditioning. However, in order to get either the real residual (in (4.1)) or the real approximate (in (4.2)), some additional work is required. In contrast to algorithms as Bi-CG and GCR, there is no variant of preconditioned BiCGstab(l) that generates the real residual and the approximations of interest without additional computational work or additional storage requirement.

5. Numerical examples.

In this section we will discuss some numerical experiments. These experiments are intended to show the characteristic behavior of BiCGstab(l) for certain linear systems. We do not pretend that the problems are solved in the best possible way. For instance, in some experiments we used a preconditioner, whereas in others we did not. With a suitable preconditioner all methods can be made to converge efficiently, but this is not the point we would like to make. The experiments are used to show that BiCGstab(l) may be a good alternative for certain problems. The algorithm was implemented as in Algorithm 3.1.

ETNA
Kent State University
etna@mcs.kent.edu

All partial differential equations were discretized with finite volumes discretization. When a preconditioner was used then the explicitly left preconditioned system was solved (see (4.1)). In all cases $x_0 = 0$ was taken as an initial guess. The experiments were done on a CRAY Y-MP 4/464, in a multi-user environment. The iterations were stopped when $\|\mathbf{r}_k\|_2/\|\mathbf{r}_0\|_2 < 10^{-9}$ (except in example 2 where the iterations were stopped when $\|\mathbf{r}_k\|_2/\|\mathbf{r}_0\|_2 < 10^{-12}$), or when the number of matrix multiplications exceeded 1000.

The figures show the convergence behavior of the iterative methods. For Bi-CGstab(l) we have plotted the norms of the residuals $\mathbf{r}_{ml}$ that are computed by Algorithm 3.1, i.e. only every l-th step (see our discussion at the end of section 2). Horizontally the number of matrix multiplications is counted. In exact arithmetic this number represents the dimension of the relevant Krylov subspace, except for Bi-CG, where it should be divided by two.

At the end of this section we give in Table 5.1 an overview of the required CPU-time for the *true* residual norm of several iterative methods. The numbers between brackets () are the log of the ℓ_2-norm of the final *true* residuals: ${}^{10}\log(\|b - A\mathbf{x}_k\|_2)$. The log of the norm of the computed updated residuals can be seen from the figures. A ‘ * ’ in Table 5.1 indicates that the method did not meet the required tolerance before 1000 multiplications of the matrix A. We did our experiments for Bi-CG, CGS and several popular or successful GMRES variants. We selected algorithms that have about the same memory requirements as the BiCGstab(l) algorithms that we tested. If one can store 13, say, n-vectors then one may choose for instance between BiCGstab(4), GMRES(10) and GMRESR(3,4) [19]. In our experiments BiCGstab(4) then seems to be the better choice.

5.1. Example 1. First we consider an advection dominated 2-nd order partial differential equation, with Dirichlet boundary conditions, on the unit cube (this equation was taken from [9]):

$$u_{xx} + u_{yy} + u_{zz} + 1000u_x = F.$$

The function F is defined by the solution $u(x, y, z) = \exp(xyz)\sin(\pi x)\sin(\pi y)\sin(\pi z)$. This equation was discretized using $(52\times52\times52)$ volumes, resulting in a seven-diagonal linear system of order 125000. No preconditioning was used.

In Fig. 5.1 we see a plot of the convergence history. Bi-CGSTAB stagnates as might be anticipated from the fact that this linear system has large complex eigenpairs. Surprisingly, Bi-CGSTAB does even worse than Bi-CG. For this type of matrices this behavior of Bi-CGSTAB is not uncommon and might be explained by the poor first degree minimal residual reductions. In that case the Bi-CG iteration coefficients α_k and β_k are not accurately computed. BiCGstab(2) converges quite nicely and almost twice as fast as Bi-CG (see our discussion in section 2).

5.2. Example 2. Next, we give an example where BiCGStab2 [5] suffers from the underlying Bi-CGSTAB algorithm (see our discussion in the introduction).

The symmetric positive definite linear system stems from a (200×200) discretization of

$$-(Du_x)_x - (Du_y)_y = 1,$$

over the unit square, with Dirichlet boundary conditions along $y = 0$ and Neumann conditions along the other parts of the boundary. The function D is defined as

$$D = 1000 \text{ for } 0.1 \le x, y \le 0.9 \text{ and } D = 1 \text{ elsewhere.}$$

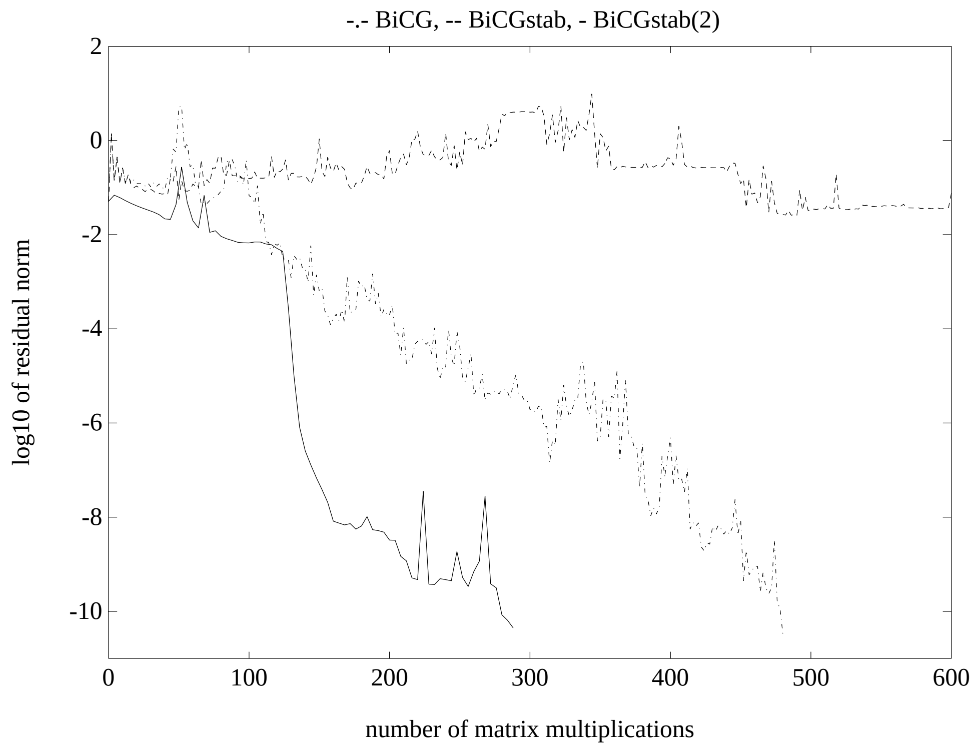

FIG. 5.1. *Convergence plot of example 1.*

Modified Incomplete Cholesky Decomposition was used as preconditioner. This example was taken from [17]. A convergence plot is given in FIG. 5.2.

Here the underlying Bi-CG algorithm looses bi-orthogonality among the residuals in a very early phase and consequently superlinear convergence takes place for none of the methods (in contrast to what might be expected; see, for instance, [14] and [18]), but apparently the BiCGstab(l) algorithm for $l = 2, 4$, has less problems. Gutknecht's BiCGStab2 follows the convergence history of Bi-CGSTAB almost perfectly. This kind of behavior was also observed by Gutknecht. Apparently the polynomials of degree one in the odd steps spoil the overall convergence behavior of BiCGStab2.

In exact arithmetic we have that $\mathbf{r}_k = b - A\mathbf{x}_k$. In finite precision arithmetic the true residual $b - A\mathbf{x}_k$ and the recursively computed $\mathbf{r}_k$ may differ. The difference will be more significant if the convergence history of the residuals shows large peaks. In our algorithm the updates for the approximations follow very closely the updates for the residuals: in each step we have $\widehat{x}_0 = \widehat{x}_0 + w$ where $\widehat{r}_0 = \widehat{r}_0 - Aw$. In Gutknecht's version the formulas that describe the update of the approximations are quite different from the ones for the residuals. Therefore, if the true residuals and the computed ones drift apart this is much more apparent in Gutknecht's version. In this experiment the final computed preconditioned residual norms were of order 10^{-8}, whereas the *true* preconditioned residual norms were of order 10^{-4} for BiCGstab(l), $l = 1, 2, 4$, but only of order 10^{-1} for BiCGStab2 (see TABLE 5.1).

Although BiCGstab(l) becomes more expensive with respect to the number of inner products and vector updates as l increases, the convergence may be faster,

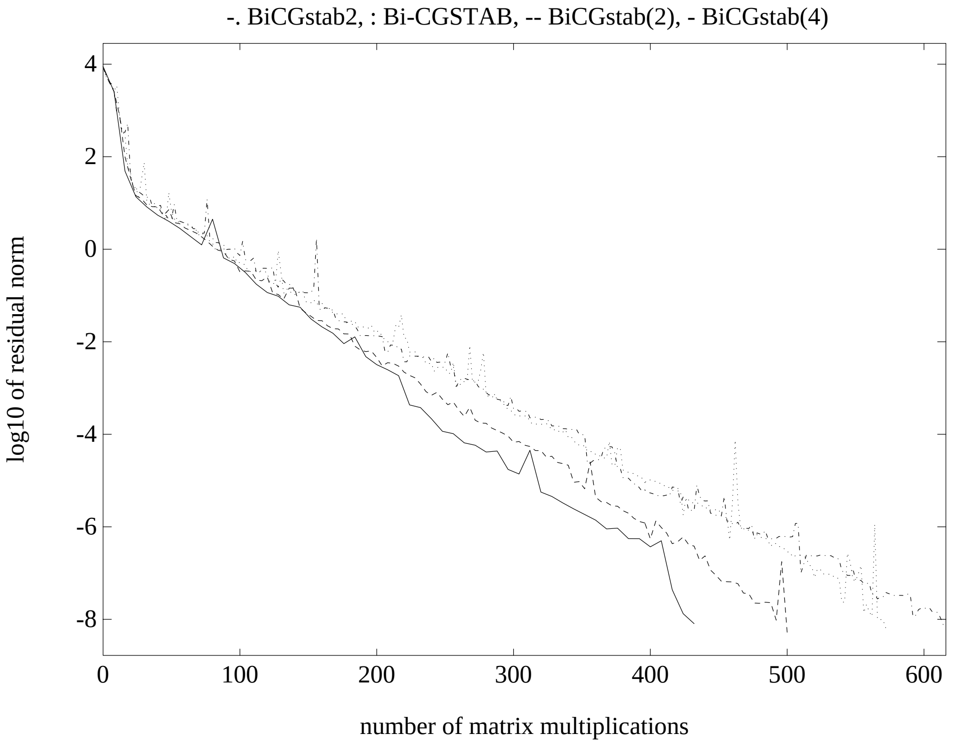

FIG. 5.2. *Convergence plot for example 2.*

and therefore, the total CPU-time needed to find an accurate approximation may *decrease*. In this example the BiCGstab(4) algorithm (for instance) is faster than the BiCGstab(2) algorithm (see TABLE 5.1). So it is sometimes more profitable to use an $l > 2$ (see also our next example).

5.3. Example 3. Our third example shows more clearly that taking $l > 2$ may be beneficial. Here BiCGstab(2) converges very slowly, whereas BiCGstab(4) does not seem to have any problem: it converges quite nicely, although linearly. This example was taken from [10].

The nonsymmetric linear system comes from a (201×201) finite volume discretization of

$$-\epsilon(u_{xx} + u_{yy}) + a(x, y)u_x + b(x, y)u_y = 0,$$

on the unit square, where

$$a(x, y) = 4x(x - 1)(1 - 2y), \qquad b(x, y) = 4y(1 - y)(1 - 2x),$$

with Dirichlet boundary conditions $u(x, y) = \sin(\pi x) + \sin(13\pi x) + \sin(\pi y) + \sin(13\pi y)$. We took $\epsilon = 10^{-1}$ and did not use any preconditioning. A convergence plot is shown in FIG. 5.3.

5.4. Example 4. Our last example shows that even if Bi-CGSTAB converges well, BiCGstab(l), $l = 2, 4, \ldots$, may be good competitors. Moreover, when the

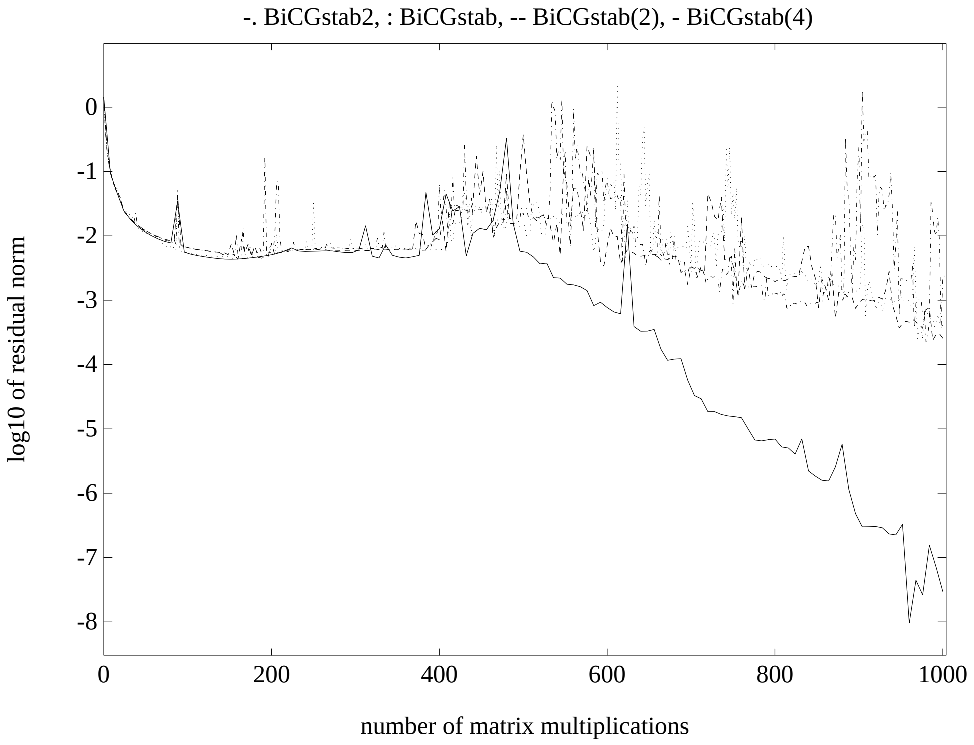

FIG. 5.3. *Convergence plot for example 3.*

problem is discretized on a finer grid BiCGstab(2) seems to be a better choice for solving this problem. The problem was taken from [17].

The two nonsymmetric linear systems come from a (129×129) and a (201×201) finite volume discretization of the partial differential equation

$$-(Au_x)_x - (Au_y)_y + B(x, y)u_x = F$$

over the unit square, with $B(x, y) = 2\exp(2(x^2 + y^2))$. Along the boundaries we have Dirichlet conditions: $u = 1$ for $y = 0, x = 0$ and $x = 1$, and $u = 0$ for $y = 1$. The function A is defined as shown in figure 5.6; $F = 0$ everywhere, except for the small subsquare in the center where $F = 100$. Incomplete LU factorization was used as a preconditioner.

From FIG. 5.4 we observe that Bi-CGSTAB and BiCGstab(2) behave similarly for the coarser grid with BiCGstab(2) slightly faster, but on the finer grid (FIG. 5.5) BiCGstab(2) performs much better than Bi-CGSTAB. BiCGstab(4) and BiCGstab(8) have a similar convergence history as BiCGstab(2). Compare also TABLE 5.1.

5.5. Conclusions. From our experiments we have learned that the BiCGstab(l) may be an attractive method. The algorithm is a generalization of van der Vorst's Bi-CGSTAB [17]. For $l = 1$ BiCGstab(l) computes exactly the same approximation $\mathbf{x}_k$ as Bi-CGSTAB does.

For $l > 1$ it seems that BiCGstab(l) is less affected by relatively large complex eigenpairs (as one encounters in advection dominated partial differential equations). Its computational work and memory requirement is modest.

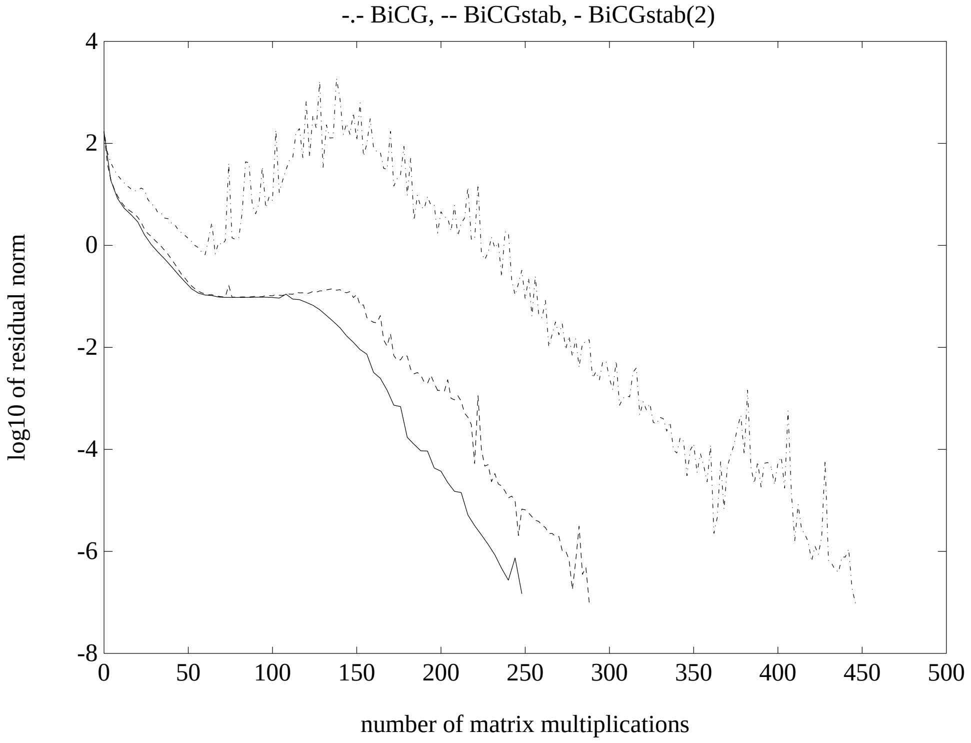

FIG. 5.4. *Convergence plot for example 4* (129 × 129).

TABLE 5.1
CPU-time and log10 of the true *residual norm (see the introduction of 5).*

METHOD	Example 1	Example 2	Example 3	Ex. 4 (129)	Ex. 4 (201)
Bi-CG	4.96 (-10.5)	6.35 (-4.2)	2.71 (-2.4)*	1.60 (-7.0)	4.58 (-7.0)
CGS	divergence	4.54 (-4.4)	break down	divergence	divergence
Bi-CGSTAB	stagnation	4.67 (-4.3)	3.22 (-3.4)*	1.10 (-7.0)	7.87 (-6.2)
BiCGStab2	4.42 (-10.7)	5.63 (-1.9)	4.26 (-2.6)*	1.31 (-6.8)	3.77 (-6.7)
BiCGstab(2)	3.88 (-10.4)	4.45 (-4.5)	4.00 (-3.6)*	1.01 (-6.8)	3.68 (-6.7)
BiCGstab(4)	4.17 (-10.9)	4.00 (-4.5)	4.35 (-7.5)	1.11 (-6.8)	3.68 (-6.8)
BiCGstab(8)	5.03 (-11.1)	4.36 (-3.5)	5.54 (-6.9)	1.27 (-7.5)	4.06 (-6.9)
GMRES(6)	5.27 (-10.3)	stagnation	4.69 (-2.7)*	stagnation	stagnation
GMRES(10)	6.30 (-10.3)	stagnation	5.16 (-3.5)*	stagnation	stagnation
GMRESR(2,2)	8.85 (-10.3)	stagnation	5.71 (-2.5)*	stagnation	stagnation
GMRESR(3,4)	6.25 (-10.3)	stagnation	5.16 (-2.6)*	stagnation	stagnation

BiCGstab(2) is, in exact arithmetic, equivalent with Gutknecht's BiCGStab2 [5]. However we have given arguments and experimental evidence for the superiority of our version.

Therefore, we conclude that BiCGstab(l) may be considered as a competitive algorithm to solve nonsymmetric linear systems of equations.

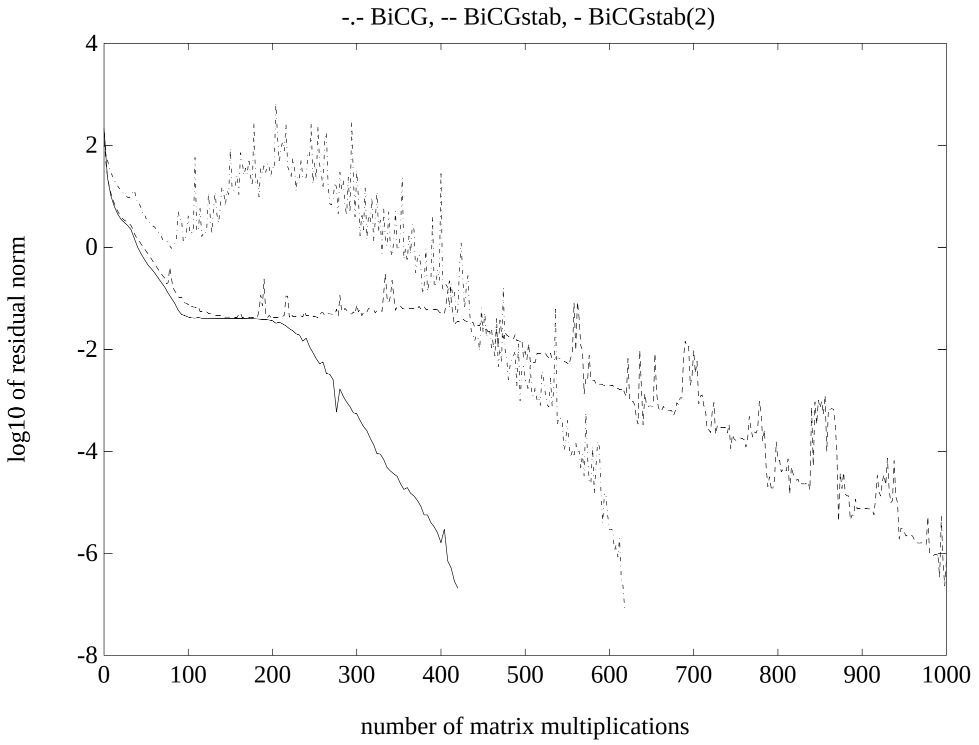

FIG. 5.5. *Convergence plot for example 4* (201 × 201).

Acknowledgement. The authors are grateful to Henk van der Vorst for encouragement, helpful comments and inspiring discussions.

REFERENCES

[1] R.E. Bank and T.F. Chan, *A composite step bi-conjugate gradient algorithm for nonsymmetric linear systems*, Tech. Report, University of California, 1992.

[2] R. Fletcher, *Conjugate gradient methods for indefinite systems*, in Proc. of the Dundee Biennial Conference on Numerical Analysis, G. Watson, ed., Springer-Verlag, New York, 1975.

[3] R.W. Freund and N.M. Nachtigal, *QMR: a quasi-minimal residual method for non-Hermitian linear systems*, Numer. Math., 60 (1991), pp. 315–339.

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

[5] M.H. Gutknecht, *Variants of BiCGStab for matrices with complex spectrum*, IPS Research Report No. 91-14, 1991.

[6] M.H. Gutknecht, *A completed theory of the unsymmetric Lanczos process and related algorithms, Part 1*, SIAM J. Matrix Anal. Appl., 13 (1992), No. 2 pp. 594–639.

[7] M.H. Gutknecht, *A completed theory of the unsymmetric Lanczos process and related algorithms, Part 2*, SIAM J. Matrix Anal. Appl., 15 (1994), No. 1.

[8] C. Lanczos, *Solution of systems of linear equations by minimized iteration*, J. Res. Nat. Bur. Stand., 49 (1952), pp. 33–53.

[9] U. Meier Yang, *Preconditioned Conjugate Gradient-Like methods for Nonsymmetric Linear Systems*, Preprint, Center for Research and Development, University of Illinois at Urbana-Champaign, 1992.

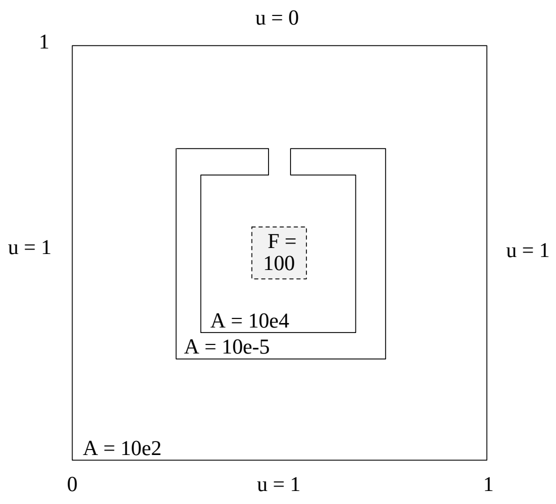

FIG. 5.6. *The coefficients for example 4.*

[10] J. RUGE AND K. STÜBEN, *Efficient solution of finite difference and finite element equations*, in Multigrid methods for integral and differential equations, D.J. Paddon and H. Holstein, ed., pp. 200–203, Clarendon Press, Oxford, 1985.
[11] Y. SAAD, *A flexible inner-outer preconditioned GMRES algorithm*, SIAM J. Sci. Statist. Comput., 14, (1993).
[12] Y. SAAD AND M.H. SCHULTZ, *GMRES: A generalized minimum residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869.
[13] G.L.G. SLEIJPEN, D.R. FOKKEMA AND H.A. VAN DER VORST, *Bi-CGstab(pol); a class of efficient solvers of large systems of linear equations*, Preprint RUU, 1993, in preparation.
[14] A. VAN DER SLUIS AND H. A. VAN DER VORST, *The rate of convergence of conjugate gradients*, Numerische Mathematik, 48 (1986), pp. 543–560.
[15] P. SONNEVELD, *CGS, a fast Lanczos-type solver for nonsymetric linear systems*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 36–52.
[16] H.A. VAN DER VORST, *The convergence behavior of preconditioned CG and CG-S in the presence of rounding errors*, Lecture Notes in Math., 1457, pp. 126–136, Springer-Verlag, Berlin, 1990.
[17] 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., 13 (1992), pp. 631–644.
[18] H.A. VAN DER VORST AND G.L.G. SLEIJPEN, *The effect of incomplete decomposition preconditioning on the convergence of conjugate gradients*, in Incomplete Decompositions, Proceedings of the Eigth GAMM Seminar, Vieweg Verlag, Braunschweig, 1992.
[19] H.A. VAN DER VORST AND C. VUIK, *GMRESR: A family of nested GMRES methods*, Tech. Report 91–80, Delft University of Technology, Faculty of Tech. Math., Delft, 1991.