BiCGstab(l) and other hybrid Bi-CG methods

G.L.G. Sleijpen, H.A. van der Vorst and D.R. Fokkema
Mathematical Institute, University of Utrecht, P.O. Box 80.010, NL-3508 TA Utrecht, The Netherlands

Received 29 October 1993; revised 2 March 1994
Communicated by C. Brezinski

It is well-known that Bi-CG can be adapted so that the operations with A^T can be avoided, and hybrid methods can be constructed in which it is attempted to further improve the convergence behaviour. Examples of this are CGS, Bi-CGSTAB, and the more general BiCGstab(l) method. In this paper it is shown that BiCGstab(l) can be implemented in different ways. Each of the suggested approaches has its own advantages and disadvantages. Our implementations allow for combinations of Bi-CG with arbitrary polynomial methods. The choice for a specific implementation can also be made for reasons of numerical stability. This aspect receives much attention. Various effects have been illustrated by numerical examples.

Keywords: Bi-Conjugate gradients, non-symmetric linear systems, CGS, Bi-CGSTAB, iterative solvers, ORTHODIR, Krylov subspace.

AMS subject classification: 65F10.

1. Introduction and background

The Bi-CG algorithm [3,7] is an iterative solution method for linear systems

$$Ax = b \tag{1}$$

in which A is some given non-singular $n \times n$ matrix and b some given n-vector. Typically, n is large and A is sparse. For ease of presentation, we assume A and b to be real.

Starting with some initial approximation x_0 for x and some "shadow" residual $\tilde{r}_0$ Bi-CG produces iteratively sequences of approximations x_k, residuals r_k and search directions u_k by

$$u_k = r_k - \beta_k u_{k-1}, \quad x_{k+1} = x_k + \alpha_k u_k, \quad r_{k+1} = r_k - \alpha_k A u_k, \tag{2}$$

where α_k and β_k are appropriate scalars, $u_{-1} = 0$, and $r_0 = b - Ax_0$. The residual r_k and the search direction u_k are in the Krylov subspace $\mathcal{K}_{k+1}(A; r_0)$ of order $k+1$ generated by A and r_0. This implies that the residuals r_k can be written as $r_k = \phi_k(A) r_0$ where ϕ_k is a certain polynomial, the so-called Bi-CG polynomial, in the space $\mathcal{P}_k^1$ of all polynomials q of degree k for which $q(0) = 1$. We will use

this polynomial representation frequently as it is a convenient way to express our ideas.

The Bi-CG coefficients α_k and β_k are such that r_k and Au_k are orthogonal to the shadow Krylov subspace $\mathcal{K}_k(A^T; \tilde{r}_0)$. If (ψ_k) is some sequence of polynomials of degree k with a non-trivial leading coefficient θ_k then (see [15] or [14]):

$$\beta_k = \frac{\theta_{k-1}}{\theta_k} \frac{(r_k, \psi_k(A^T)\tilde{r}_0)}{(Au_{k-1}, \psi_{k-1}(A^T)\tilde{r}_0)} \quad \text{and} \quad \alpha_k = \frac{(r_k, \psi_k(A^T)\tilde{r}_0)}{(Au_k, \psi_k(A^T)\tilde{r}_0)}. \tag{3}$$

In standard Bi-CG one takes $\psi_k = \phi_k$.

It was Sonneveld [15] who suggested to rewrite the inner products so as to avoid the operations with A^T, e.g.

$$(r_k, \psi_k(A^T)\tilde{r}_0) = (\psi_k(A)r_k, \tilde{r}_0), \tag{4}$$

and to generate recursions for the vectors $\boldsymbol{r}_k = \psi_k(A)r_k$ hoping that the operator $\psi_k(A)$ would lead to a further reduction of the Bi-CG residual. More specifically, he suggested to take $\psi_k = \phi_k$, which led to the CGS method: $\boldsymbol{r}_k = \phi_k^2(A)r_0$. The corresponding search directions for the corresponding approximation $\boldsymbol{x}_k$ can be constructed in a similar way. In this approach the Bi-CG residuals and search directions are not computed explicitly.

Other hybrid Bi-CG schemes have emerged since then, in which the usage of A^T is avoided. Freund and Nachtigal [4] use the same basis vectors as in CGS, but they propose to construct an $\boldsymbol{x}_k$ for which $\|b - A\boldsymbol{x}_k\|_2$ is quasi-minimal. This has led to the TFQMR method. TFQMR has typically a convergence behavior that is much smoother than CGS has, but in average the method is about as fast as CGS in terms of the number of iteration steps necessary to achieve a given reduction of the residual norm.

In [17] it was suggested to choose ψ_k as a product of linear factors, which were constructed to minimize residuals in one direction. This led to the Bi-CGSTAB algorithm. Gutknecht [6] was the first to investigate the use of quadratic factors: BiCGStab2. In [14] this was further generalized to a composite of higher degree factors which minimize residuals over l-dimensional subspaces: BiCGstab(l) following a different approach as suggested (for quadratic factors) in [6]. By experiments also it was shown in [14] that their approach was often superior over Gutknecht's approach. There are many different possibilities for the choice of the factors as well as for the implementation of the corresponding methods, apart from the approach suggested in [6]. These different approaches form the main theme of our discussion, as it appears that they have a noticeable different behavior with respect to numerical stability.

The necessity to pay attention to stability aspects of these hybrid methods was already apparent in [14] by the observation that the construction of higher-degree factors from a simple power basis could lead to a degradation of the speed of convergence. We will discuss several computational schemes aiming at a stable scheme that is also efficient in terms of computational complexity and memory requirements.

Our paper has been organized as follows. We start with a discussion on the effects of evaluation errors caused by finite precision arithmetic, since they play an important role in the design and choice of our algorithms. In section 3 the general idea for the construction of hybrid Bi-CG schemes is presented. These schemes consist of two parts: a *Bi-CG part* where the ϕ_k is constructed, and a *polynomial part* (POL part) where ψ_k is constructed. For ψ_k we have to construct a basis for relevant l-dimensional subspaces. In section 4 we follow a similar approach as in BiCGstab(l) [14]. This leads sometimes to a poor basis. A more stable approach is suggested in section 5. It turns out that this may lead to a poor construction of the Bi-CG search directions, and for that reason we discuss a suitable compromise for the basis construction in section 6. We conclude our paper with some instructive numerical experiments.

2. Finite precision arithmetic

Because the actual computational work is done in finite precision arithmetic, evaluation errors are introduced in any of the computational steps. These errors affect

(i) the numerical accuracy and
(ii) the speed of convergence.

2.1. *Numerical accuracy*

From the relations (2) and (3) we learn that the approximations $\boldsymbol{x}_k$ are not involved in the recurrence relations by which the residuals $\boldsymbol{r}_k$ are produced. That is, there is no correction mechanism that influences the $\boldsymbol{r}_k$ if the $\boldsymbol{x}_k$ become polluted by rounding errors. Both vectors are updated by different vectors and the vectors $\boldsymbol{x}_k$ do not influence the vectors $\boldsymbol{r}_k$. However, in most applications we will be mainly interested in the $\boldsymbol{x}_k$, and since we can only judge the quality of these approximations by the values of $\boldsymbol{r}_k$, we want to have $\boldsymbol{r}_k$ which are sufficiently close to $b - A\boldsymbol{x}_k$.

In exact arithmetic, we would have that $\boldsymbol{r}_k = b - A\boldsymbol{x}_k$, while in finite precision arithmetic the *updated residual* $\boldsymbol{r}_k$ may differ from the *true residual* $b - A\boldsymbol{x}_k$. In our algorithms we try to keep this inaccuracy limited but the norm of the updated residual may still be much smaller than the norm of the true residual. It is not advisable to replace the updated residual in the updating formula for $\boldsymbol{u}_k$ by the true residual and to continue the iteration, since such a strategy may disturb the incorporated Bi-CG process completely. In fact, the underlying bi-orthogonalization for the r_k may then be destroyed. For an example of this see [17].

It is a waste of computational effort to continue the computation as soon as the *true* and the *updated residual* differ too much, and therefore it would be nice to identify the phase in the iteration process where this is going to happen. We will make suggestions to this goal.

To compute the norm $N_k(x) \equiv \|b - Ax_k\|$ of the *true residual* would be too expensive since it requires one additional matrix multiplication (MV) in each step.

An alternative might be to exploit techniques that compute cheaply the norm of the *true residual* of an associated process [13,18,21]. These minimal residual smoothing (MRS) techniques generate at each iteration step a new approximation y_k and a residual s_k with the following properties (see [21]: algorithm 2.2 and algorithm 3.2.2):

(1) y_k and s_k can be updated whenever x_k and r_k are updated. For this purpose, the algorithm that computes x_k and r_k has to be extended by only a few lines, requiring an additional few vector updates (AXPYs) and inner products (DOTs) only (no additional MV).
(2) It was observed in experiments that in finite precision arithmetic $N_k(y) \equiv \|b - Ay_k\|$ and $\|s_k\|$ do not differ too much, as long as both are not too small with respect to $\|s_0\|$.
(3) $N_k(y)$ is smaller than $N_k(x)$. Moreover, $(N_k(y))$ converges much less irregularly than $(N_k(x))$, but not faster (that is, $\min_{j<k} N_j(y) \approx \min_{j<k} N_j(x)$).
(4) It was also observed in numerical experiments that in finite precision arithmetic, $N_k(x)$ as well as $N_k(y)$ will stagnate at a certain "*accuracy level*". For both sequences, the optimal accuracy will be on approximately the same level (see [21]).

If the attainable accuracy is not sufficient we have to restart at some suitable point, see [16]. However, since the convergence of optimal Krylov subspace methods tends to accelerate (superlinear convergence) we want to avoid restarts as much as possible, and therefore we aim at implementations with a sufficiently small accuracy level. We will concentrate on the (implementation of the) BiCG-stab(l) algorithm itself. The MRS techniques may be used on top of our algorithms in order to further smooth the convergence and to monitor when the accuracy level has been reached.

We will say that an algorithm is *accurate* for a certain problem if the *updated residual* r_j and the *true residual* $b - Ax_j$ are of comparable size for the j's of interest.

The best we can hope for is that for each j the error in the residual is only the result of applying A to the update w_{j+1} for x_j in finite precision arithmetic:

$$r_{j+1} = r_j - Aw_{j+1} - \Delta_A w_{j+1} \quad \text{if} \quad x_{j+1} = x_j + w_{j+1}, \quad \text{for each } j, \tag{5}$$

where Δ_A is an $n \times n$ matrix for which $|\Delta_A| \preceq n_A \bar{\xi} |A|$: n_A is the maximum number of non-zero matrix entries per row of A, $|B| \equiv (|b_{ij}|)$ if $B = (b_{ij})$, $\bar{\xi}$ is the relative machine precision, the inequality $\preceq$ refers to element-wise $\leqslant$. In the Bi-CG part, we compute explicitly the update Aw_j for the residual r_j from the update w_j for the approximation x_j by matrix multiplication: for this part, (5) describes well the local deviations caused by evaluation errors. In the polynomial part other errors may be introduced because of a different update strategy for r_j and x_j (other than with a simple multiplication with A). This will be the subject of an error analysis in section 3.3.

In the "ideal" case (i.e. situation (5) whenever we update the approximation) we have that

$$r_k - (b - Ax_k) = \sum_{j=1}^{k} \Delta_A w_j = \sum_{j=1}^{k} \Delta_A (e_{j-1} - e_j), \tag{6}$$

where the perturbation matrix Δ_A may depend on j and e_j is the approximation error in the jth approximation: $e_j \equiv x - x_j$. Hence,

$$\begin{aligned} \big| \|r_k\| - \|b - Ax_k\| \big| &\leqslant 2kn_A \bar{\xi} \, \big\| |A| \big\| \max_j \|e_j\| \\ &\leqslant 2kn_A \bar{\xi} \, \big\| |A| \big\| \, \big\| A^{-1} \big\| \max_j \|r_j\|. \end{aligned} \tag{7}$$

Except for the factor k, the first upper-bound appears to be rather sharp. We see that approximations with large approximation errors may ultimately lead to an inaccurate result. Such large local approximation errors are typical for CGS, and in [17] an example of the resulting numerical inaccuracy is given. If there are a number of approximations with comparable large approximation errors, then their multiplicity may replace the factor k, otherwise it will be only the largest approximation error that makes up virtually the bound for the deviation.

2.2. *Errors in the iteration coefficients*

Any evaluation error in the computation of the residual r_k, the search direction u_k or the Bi-CG coefficients β_k or α_k may disturb the underlying Bi-CG process and hence affect the speed of convergence of the composite method.

In the polynomial part we compute polynomial coefficients γ_i. They represent the polynomial by which we wish to achieve a further reduction of the residual. We would like to see that the composite scheme converges faster than Bi-CG by at least a factor which is the product of all reductions due to the polynomial part. In order to achieve this goal it is necessary that the polynomial parts do not deteriorate the Bi-CG part. In fact one might accept a polynomial which leads to a less than optimal reduction in a given step, but which improves the numerical stability of the Bi-CG iteration coefficients. Therefore, we are not so much interested in an accurate computation of the γ_i.

We will now discuss a possible source of errors in the iteration coefficients α_k and β_k due to an unfortunate choice of the polynomial parts in the hybrid Bi-CG scheme. In our discussion we focus on the BiCGstab(l) algorithm, but the ideas carry over easily to more general composite Bi-CG algorithms.

At least in exact arithmetic one would expect BiCGstab(l) to converge faster than Bi-CG, since the additional factors are equivalent with GMRES(l)-processes [12] and hence should not increase the Bi-CG residuals in norm. However, we sometimes observe that for instance Bi-CGSTAB (= BiCGstab(1)) is slower than Bi-CG, or even fails to converge in some cases where Bi-CG does converge. In our experience this happens to be the case when the additional GMRES(l) process (temporarily)

leads to a poor reduction of the current residual vector. It is well-known that if the GMRES(l) factor gives no reduction at all at a given step of the process then the BiCGstab(l) process breaks down at the next step. Although one might guess that a poor reduction of GMRES(l) might lead to a near-breakdown situation, we will now give arguments that such a poor reduction, even if only in one phase of the process, may spoil the iteration process considerably from then on, also in cases where in further phases the reduction by GMRES(l) is not small.

The Bi-CG process is, amongst others, determined by the iteration coefficients α_k and β_k. Both coefficients are computed from inner products and they have the inner product $(r_k, \psi_k(A^{\mathrm{T}})\tilde{r}_0) = (\phi_k(A)r_0, \psi_k(A^{\mathrm{T}})\tilde{r}_0)$ in common (see (3)). It is obvious that, in addition to other error sources, any computational error in this inner product will directly lead to a similar relative perturbation in both coefficients.

The set of vectors $\psi_j(A^{\mathrm{T}})\tilde{r}_0$ form a basis for the Krylov subspace $\mathcal{K}_k(A^{\mathrm{T}}; \tilde{r}_0)$. For simplifying our arguments these vectors can be assumed to be normalized in 1-norm.

Let

$$\psi_k(A^{\mathrm{T}})\tilde{r}_0 = \sum_{j=1}^{k-1} \zeta_j^{(k)} \psi_j(A^{\mathrm{T}})\tilde{r}_0 + \zeta_k v_k$$

with $\|\psi_k(A^{\mathrm{T}})\tilde{r}_0\|_1 = 1$ and $\|v_k\|_1 = 1$, and $v_k \perp \{\tilde{r}_0, \ldots, \psi_{k-1}(A^{\mathrm{T}})\tilde{r}_0\}$. Then it follows that

$$|(\phi_k(A)r_0, \psi_k(A^{\mathrm{T}})\tilde{r}_0)| = |\zeta_k(\phi_k(A)r_0, v_k)| \leqslant |\zeta_k| \|\phi_k(A)r_0\|_\infty.$$

The error in $(\phi_k(A)r_0, \psi_k(A^{\mathrm{T}})\tilde{r}_0)$ due to rounding errors can be realistically bounded by $n\bar{\xi}(|\phi_k(A)r_0|, |\psi_k(A^{\mathrm{T}})\tilde{r}_0|) \leqslant n\bar{\xi}\|\phi_k(A)r_0\|_\infty$. If these inequalities are approximate equalities, then we may expect a relative error in this inner product in the order of $n\bar{\xi}/|\zeta_k|$.

It is well-known that when $\phi_k(A)r_0$ is (nearly) orthogonal with respect to $\phi_k(A^{\mathrm{T}})\tilde{r}_0$ we have (near) breakdown in Bi-CG, but our analysis shows that we may also have severe problems (i.e., poor representation of the iteration coefficients) when ζ_k is small, and this may happen even in situations where $\phi_k(A)r_0$ makes a small angle with $\phi_k(A^{\mathrm{T}})\tilde{r}_0$.

If we apply Bi-CG to a problem for which $A = A^{\mathrm{T}}$, with $r_0 = \tilde{r}_0$, then the set of vectors $\phi_j(A^{\mathrm{T}})\tilde{r}_0$ is orthogonal and hence $\zeta_k = 1$ for all k. In such a case we will have no errors in the iteration process that can be attributed to poor orthogonality of the shadow set (we assume that we have not yet reached phases in the Bi-CG process where some Ritz values have fully converged, which also goes hand in hand with a loss of orthogonality in the $\phi_j(A)r_0$). For other problems, e.g., when $A \neq A^{\mathrm{T}}$ or $r_0 \neq \tilde{r}_0$, the value of a ζ_k may be small and this will, of course, lead to convergence problems in Bi-CG. The question now arises what can be said if we make different (implicit) choices for the shadow basis by selecting some Bi-CG variant like CGS or BiCGstab(l).

The CGS method can be viewed as a method in which the inner products in (3) have been reformulated, but where these inner products could also have been computed by the $\phi_j(A^{\mathrm{T}})\tilde{r}_0$-set which would have been generated by Bi-CG. Therefore, we expect for

CGS similar values for ζ_k as for the corresponding Bi-CG process, and errors in the evaluation of the discussed inner product should be about the same for both processes.

The standard Bi-CGSTAB method can be seen as the composite of Bi-CG and GMRES(1). If GMRES(1) converges sufficiently well, i.e., if we see at least some kind of reasonable reduction *in each* iteration step, then the implicitly formed basis $\psi_j(A^T)\tilde{r}_0$ will very likely be not dependent. Namely, if some direction would dominate then we would see slow convergence in other directions. For such well-converging problems we may expect that the Bi-CG coefficients are fairly well recovered, and hence Bi-CGSTAB may then be expected to converge faster than Bi-CG (the ratio of improvement being determined by the product of all GMRES(1) reductions). If GMRES(1) stagnates, however, even if this happens during a short phase of the iteration process, then we may expect near-dependent $\psi_j(A^T)\tilde{r}_0$ vectors for the stagnation phase, and hence small ζ_j's. In such a case we might get some improvement by selecting different rules for ψ_j, e.g., $\psi_j = t\psi_{j-1}$, as is done in BiCGstab(l) [14]. For the well-known periodic matrix (for which GMRES stagnates until the very last step) this would even lead to an optimal orthogonal basis. After the stagnation phase we would still have the original Bi-CG coefficients represented fairly well and convergence may proceed as expected, whereas in the original Bi-CGSTAB process the coefficients would have been corrupted due to the stagnation and poor convergence would have resulted.

3. Hybrid Bi-CG methods

3.1. *General approach for the polynomial factor*

We will now discuss more general approaches for the selection of ψ_k. It is essential that ψ_k has a non-trivial leading coefficient θ_k and for stability reasons θ_k should not be too small. This is a serious condition, as we have seen in section 2.2. For instance, if we select a minimal residual polynomial of some low degree then due to temporary stagnation this factor may be of lower degree, thus preventing proper continuation of the composite scheme. This may be not only a reason for breakdown in Bi-CGSTAB, but it may also lead to a poor reconstruction of the Bi-CG recursion coefficients (cf. section 2.2). We suggest to circumvent this by introducing factors $\tilde{q}_k$ that serve to avoid degeneration of the ψ_k, but which are not used to reduce the residuals. We will explain this in some more detail.

Consider some sequences (q_k) and $(\tilde{q}_k)$ of polynomials for which $q_k \in \mathcal{P}_k^1$ and $\psi_k = \tilde{q}_k q_k$ is of exact degree k. For q_k and $\tilde{q}_k$, one may take for instance $q_k = \phi_k$ and $\tilde{q}_k = 1$, but we will be interested mainly in other choices. The factor q_k is the part of ψ_k that leads to reduction, and $\tilde{q}_k$ is introduced as a factor to prevent small values of θ_k (the leading coefficient of ψ_k).

Define $\boldsymbol{r}_k = Q_k r_k$ and $\boldsymbol{u}_k = Q_k u_k$, where $Q_k = q_k(A)$, r_k is the Bi-CG residual, and u_k is the Bi-CG search direction. If θ_k denotes the leading coefficient of the

polynomial $\tilde{q}_k q_k$ it follows from (3)–(4) that

$$\beta_k = \frac{\theta_{k-1}}{\theta_k} \frac{\rho_k}{\sigma_{k-1}} \quad \text{and} \quad \alpha_k = \frac{\rho_k}{\sigma_k}, \tag{8}$$

where

$$\rho_k \equiv (\tilde{q}_k(A)\boldsymbol{r}_k, \tilde{r}_0), \quad \sigma_k \equiv (A\tilde{q}_k(A)\boldsymbol{u}_k, \tilde{r}_0). \tag{9}$$

Therefore, if we know the leading coefficients of the polynomials, we can compute the Bi-CG coefficients from $\tilde{q}_k(A)\boldsymbol{r}_k$ and $A\tilde{q}_k(A)\boldsymbol{u}_k$.

We try to find polynomials q_k for which $\|\boldsymbol{r}_k\|_2 \ll \|r_k\|_2$ and instead of r_k and u_k we compute $\boldsymbol{r}_k$, $\boldsymbol{u}_k$, $\tilde{q}_k(A)\boldsymbol{r}_k$ and $A\tilde{q}_k(A)\boldsymbol{u}_k$, for suitable polynomials $\tilde{q}_k$. If the operator Q_k reduces r_k well, then it is very attractive to proceed in this way. Since $q_k(0) = 1$ it is possible to update efficiently the approximation $\boldsymbol{x}_k$ as soon as we have the update for $\boldsymbol{r}_k$. This is due to the fact that we may write formally $q_k(A) = I - As_{k-1}(A)$, and hence with $\boldsymbol{r}_k = q_k(A)r_k$ there corresponds $\boldsymbol{x}_k = x_k + s_{k-1}(A)r_k$.

A class of hybrid Bi-CG methods can be constructed from the following choice for q_k and $\tilde{q}_k$. For $k = ml + j$ $(j = 0, \ldots, l-1)$, we define $q_k \equiv p_{m-1} \cdot \ldots \cdot p_0$ as the product of certain polynomials p_j of exact degree l with $p_j(0) = 1$, and $\tilde{q}_k = t^j$ (or some other polynomial of exact degree j). The BiCGstab(l) [14] is an example of such a hybrid Bi-CG method. In this method p_m solves the minimization problem

$$\min \| p(A)Q_{ml}\boldsymbol{r}_{ml+l} \|_2, \tag{10}$$

where the minimum is taken over all $p \in \mathscr{P}_l^1$: the p_j's are l-step minimal residual polynomials, and $\tilde{q}_k(t) = t^j$.

Actually, we are mainly interested in the approximations $\boldsymbol{x}_k$ after a full reduction cycle, i.e., for $k = ml$, but the algorithms produce approximations for other values of k as well. Most of these "intermediate" approximations serve computational purposes only. Note that they may have been constructed with $\tilde{q}_k$ which are only aimed at accuracy in the iteration coefficients.

In our discussion, we refer to the part of the algorithm consisting of the computational steps that are needed to compute $\boldsymbol{x}_{ml+l}$, $\boldsymbol{r}_{ml+l}$, $\boldsymbol{u}_{ml+l-1}$ from $\boldsymbol{x}_{ml}$, $\boldsymbol{r}_{ml}$ and $\boldsymbol{u}_{ml-1}$ as the *inner iteration*. Any inner iteration consists of two parts. In the Bi-CG part, we increase the Bi-CG indices; for instance, we compute $Q_{ml}\boldsymbol{r}_{ml+j}$ for consecutive j's. At the end of this part we have $\hat{\boldsymbol{r}}_0 = Q_{ml}\boldsymbol{r}_{ml+l}$ and, depending on the implementation, we also may have $A\hat{\boldsymbol{r}}_0, \ldots, A^l\hat{\boldsymbol{r}}_0$. In the POL part, we compute $\boldsymbol{r}_{ml+l} = p_m(A)\hat{\boldsymbol{r}}_0$, for instance by applying l steps GMRES or by forming the appropriate linear combination of the $A^i\hat{\boldsymbol{r}}_0$ (the vectors which represent $\tilde{q}_i$).

Suppose that for some $k = ml$ we have already available $\boldsymbol{u}_{k-1}$, $\boldsymbol{r}_k$, and $\boldsymbol{x}_k$, which satisfy the following relations

$$\boldsymbol{u}_{k-1} = Q_k u_{k-1}, \quad \boldsymbol{r}_k = Q_k r_k \quad \text{and} \quad \boldsymbol{x}_k, \tag{11}$$

$$\text{where } Q_k = q_k(A) \quad \text{and} \quad \boldsymbol{x}_k \text{ is such that } \boldsymbol{r}_k = b - A\boldsymbol{x}_k. \tag{12}$$

Then we may compute

$$r_{k+l}(=p_m(A)r_k), \quad u_{k+l-1}(=p_m(A)u_{k-1}), \quad \text{and} \quad x_{k+l} \tag{13}$$

explicitly as follows.

In the Bi-CG part of the inner iteration, we compute

$$\hat{r}_0 = Q_k r_{k+l}, \ \hat{u}_0 = Q_k u_{k+l-1}, \quad \text{and} \quad \hat{x}_0, \qquad \text{for which } \hat{r}_0 = b - A\hat{x}_0, \tag{14}$$

and we may compute $A^i\hat{r}_0$, $A^i\hat{u}_0$, for $i = 1, \ldots, l$.

In the POL part we form the appropriate linear combinations of the vectors $A^i\hat{r}_0$ and $A^i\hat{u}_0$ (formally described by a polynomial $p_m \in \mathcal{P}_l^1$).

With $p_m(t) = 1 - \gamma_1 t - \ldots - \gamma_l t^l$ we have that (for $k = ml$)

$$r_{k+l} = \hat{r}_0 - \sum_{i=1}^{l} \gamma_i A^i \hat{r}_0, \tag{15}$$

$$u_{k+l-1} = \hat{u}_0 - \sum_{i=1}^{l} \gamma_i A^i \hat{u}_0, \tag{16}$$

$$x_{k+l} = \hat{x}_0 + \sum_{i=1}^{l} \gamma_i A^{i-1} \hat{r}_0. \tag{17}$$

The choice of p_m can be based either on local information as in (10) (a minimal residual approach) or on information from previous steps (a Chebyshev iteration approach). In principle, one may do l steps of any Krylov subspace method to solve the equation $Ay = \hat{r}_0$ with starting residual $\hat{r}_0 = Q_k r_{k+l}$.

BiCGstab(l) is the hybrid Bi-CG method that combines minimal residual polynomials as in (10). In the POL part of an implementation of BiCGstab(l) we may use l-step GMRES [12] or l-step ORTHODIR [20] or any other implementation of the minimal residual method. We will discuss implementations in which not all of the basis vectors are computed explicitly of the form $A^i\hat{r}_0$, $A^i\hat{u}_0$, and in which the construction of p_m is implicit and intertwined with the (implicit) computation of the basis vectors. All the computational steps need special attention since they can be arranged in a number of different ways.

We will consider three different algorithms for the Bi-CG part. The first one is the most straightforward one (similar to the one proposed in [14]). The second one is aimed for more accuracy in x_k (cf. section 2.1) but it does so at the cost of a few operations with A^T at the beginning. The third approach is a compromise between the other two.

We discuss and classify the variants according to their realization of the Bi-CG part. In each of the three sections 4, 5, and 6, we first present one of the three algorithms for the Bi-CG part. Then we discuss algorithms for the POL part that seem to combine well with the algorithm in the Bi-CG part.

3.2. *Representation of computational schemes*

We will represent the major computational steps of an algorithm by some scheme, very much like a dependency graph, see, for instance, scheme 1. Each row in such a scheme represents an update phase for the most relevant vectors. Only the vectors that change in the next row or that are required for the computation of the next row are shown. The first row contains the vectors that are known at the start of the computation of the part of the algorithm that is indicated in the scheme and the last row shows the results of this part. The vector updates are derived from the Bi-CG relations (2) and are indicated by arrows. For instance, in the transition from the first row to the second row in scheme 1 we have used $Q_k u_k = Q_k(r_k - \beta_k u_{k-1}) = Q_k r_k - \beta_k Q_k u_{k-1}$. Other vector updates have not been represented explicitly. E.g., the schemes do not show how the approximations x_j are updated. However, the computation of the approximations is similar to the computation of the residuals $Q_k r_{k+j}$: if a residual is updated by adding a vector of the form Aw then the approximation is updated by adding the vector $-w$.

Vectors that require an operation with the matrix A are framed: they arise by multiplying some vector in the same row. Vectors that are needed to compute the Bi-CG coefficients α_k and β_k are underlined. The first column (the one on the left side of the line) specifies the scalars that are computed in between two successive phases. Some of these scalars and vectors have to be stored.

3.3. *The accuracy in the POL part*

As we will see, our update for the approximation in the polynomial part does not necessarily correspond very well with the update for the residual (i.e., there may be quite some difference between the *true* and the *updated residual*).

For instance, if we use (15) and (17), we should expect a local deviation in the updated residual of the form $\sum_{i=1}^{l} \gamma_i \Delta_A A^{i-1} \hat{r}_0$ (instead of the "ideal" $\Delta_A w_{ml+l}$ where $w_{ml+l} \equiv \sum_{i=1}^{l} \gamma_i A^{i-1} \hat{r}_0$, cf. (5)). The largest term in this summation may be expected to contribute to the inaccuracy in the updated residual in norm by at most $n_A \bar{\xi} \| |A| \| \max_i \| |\gamma_i| |A^{i-1} \hat{r}_0| \|$. For large $\|\gamma_i A^i \hat{r}_0\|$ this contribution may be large, and this is our motivation to consider in more detail the effect of the updates in the POL part to the inaccuracy of the hybrid scheme.

The computation in the POL part exploits basis vectors $\hat{r}_1, \ldots, \hat{r}_l$ for $\mathcal{K}_l(A; A\hat{r}_0)$ (that is, $\mathcal{K}_j(A; A\hat{r}_0)$ is spanned by $\hat{r}_1, \ldots, \hat{r}_j$ for $j = 1, \ldots, l$). For given scalars $\eta_1, \ldots, \eta_l$ we have that

$$r_{k+l} = \hat{r}_0 - \sum_{j=1}^{l} \eta_j \hat{r}_j. \tag{18}$$

The corresponding approximation x_{k+l} is of the form

$$x_{k+l} = \hat{x}_0 + \sum_{j=0}^{l-1} \tilde{\eta}_j \hat{r}_j. \tag{19}$$

The scalars $\tilde{\eta}_j$ follow from

$$\tilde{\eta} = (\tilde{\eta}_0, \ldots, \tilde{\eta}_{l-1})^T = U^{-1}\vec{\eta} \quad \text{where} \quad \vec{\eta} = (\eta_1, \ldots, \eta_l)^T, \tag{20}$$

and where the upper triangular $l \times l$ matrix $U = (u_{ij})$ is such that

$$A\hat{r}_{j-1} = \sum_{i=1}^{j} u_{ij}\hat{r}_i \quad \text{for} \quad j = 1, \ldots, l. \tag{21}$$

For instance, if we use explicitly the power basis then $\hat{r}_j = A^j\hat{r}_0$ and $U = I$. ORTHODIR produces an orthonormal basis $\hat{r}_1, \ldots, \hat{r}_l$ for $\mathcal{K}_l(A; A\hat{r}_0)$ and the upper triangular matrix U. If $v_0, \ldots, v_{l-1}$ is the orthonormal basis of $\mathcal{K}_l(A; \hat{r}_0)$ as produced by GMRES then $\hat{r}_j = Av_{j-1}$ and U is composed from the Hessenberg matrix that is generated by this method, augmented with a first column of which only the first element is non-zero.

To facilitate our presentation, we introduce the matrix V with column vectors $\hat{r}_1, \ldots, \hat{r}_l$ and the matrix V' with column vectors $\hat{r}_0, \ldots, \hat{r}_{l-1}$.

We update $\hat{x}_0$ with $w \equiv fl(\sum \tilde{\eta}_j\hat{r}_j)$ (cf. (19)) and $\hat{r}_0$ is updated with $-w' \equiv -fl(\sum \eta_j\hat{r}_j)$ (cf. (18)). The operator fl indicates that both vectors and all intermediate results are computed in finite precision arithmetic. In order to analyze the accuracy we want to compare Aw (with exact MV) with w', since in exact arithmetic both vectors are equal.

If we follow explicitly the computational scheme as suggested by (18)–(21), using the basis vectors $\hat{r}_j$, the computation of w $(= fl(V'U^{-1}\vec{\eta}))$ as well as $w'(= fl(V\vec{\eta}))$ requires the vector $\vec{\eta}$. The matrices V', V, and U are, in exact arithmetic, related through $AV' = VU$. If we assume that the construction of U is exact, given the V, then we have that $fl(AV') = VU$, or $VU = AV' + \Delta_A$, with $|\Delta_A| \preceq n_A\bar{\xi}|A||V'|$. For the computed V we find that it satisfies $V = AV'U^{-1} + \Delta_A U^{-1}$.

In this approach one may avoid to take into account explicitly the computational errors in the basis vectors $\hat{r}_j$ and in the vector $\vec{\eta}$.

We note that $w' = fl(V\vec{\eta}) = V\vec{\eta} + \vec{\delta}_1 = AV'U^{-1}\vec{\eta} + \Delta_A U^{-1}\vec{\eta} + \vec{\delta}_1$ with $|\vec{\delta}_1| \preceq l\bar{\xi}|V||\vec{\eta}|$ (neglecting $O(\bar{\xi}^2)$-terms) and $w = V'fl(\vec{\eta}) + \vec{\delta}_2$ with $|\vec{\delta}_2| \preceq l\bar{\xi}|V'||fl(\vec{\eta})|$. Furthermore, $fl(\tilde{\eta}) = fl(U^{-1}\vec{\eta}) = U^{-1}\vec{\eta} + \delta_3$ with $|\vec{\delta}_3| \preceq l\bar{\xi}|U^{-1}||\vec{\eta}|$.

Hence,

$$\begin{aligned} \|Aw - w'\| &= \|A\vec{\delta}_2 + AV'\vec{\delta}_3 - \vec{\delta}_1 - \Delta_A U^{-1}\vec{\eta}\| \\ &\leqslant \bar{\xi}(2l + n_A)\big\||A||V'||U^{-1}||\vec{\eta}|\big\| + \bar{\xi}l\big\||V||\vec{\eta}|\big\|. \end{aligned} \tag{22}$$

If $U = I$, $V = [A\hat{r}_0, \ldots, A^l\hat{r}_0]$ then $\vec{\eta} = \vec{\gamma}$, $V = AV'$, and

$$\|Aw - w'\| \leqslant \bar{\xi}l^{1/2}(3l + n_A)\max_j |\gamma_j|\big\||A||A^{j-1}\hat{r}_0|\big\|. \tag{23}$$

As argued before, the value of $\|\gamma_j A^j\hat{r}_0\|$ may be large, and this corresponds to a large right hand side in (23). In relevant situations one may create basis vectors for the POL part which lead to smaller upper bounds. One way to do this, is through the ORTHODIR approach.

The V obtained through ORTHODIR is orthonormal and $\|\vec{\eta}\|$ is then equal to the norm $\|r_{k+l} - \hat{r}_0\|$ of the update for $\hat{r}_0$. Since this update is the projection of $\hat{r}_0$ onto the Krylov subspace $\mathcal{K}_l(A; A\hat{r}_0)$ (in exact arithmetic), we may expect that $\|\vec{\eta}\| \leqslant \|\hat{r}_0\|$. Furthermore, because of $AV' = VU$ we have that $U^{-1} = V'^{\mathrm{T}} A^{-1} V$. Since $\|V\|_2 = 1$, and $\|V'^{\mathrm{T}}\|_2 \leqslant \|V'^{\mathrm{T}}\|_F = \sqrt{l}$, it follows that $\|U^{-1}\|_2 \leqslant \sqrt{l}\|A^{-1}\|_2$.

Hence, for the ORTHODIR construction of V we have that

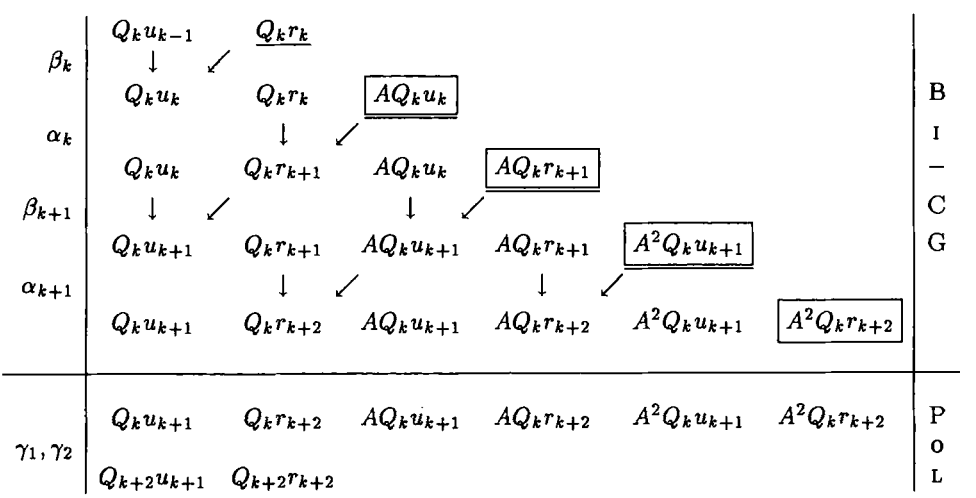
$$\|Aw - w'\| \lesssim \bar{\xi} l(3l + n_A)\||A|\|\|A^{-1}\|\|\hat{r}_0\|. \tag{24}$$

It is difficult to compare the upper bounds in (23) and (24). The expression $\sum_j \gamma_j A^j$, as factor for $\hat{r}_0$, may be interpreted as a truncated powerseries approximation for A^{-1} (cf. (17)). In many situations one will have that the maximum value of the norms $\|\gamma_j A^j\|$ of the terms in the powerseries will be (much) larger than the norm of the sum, i.e., $\|A^{-1}\|$. This may help to explain that in our experiments (with only modest values for l) we have observed better accuracy with the ORTHODIR approach.

4. The power basis

The most straightforward algorithm for the Bi-CG part is pictured in scheme 1 for $l = 2$. This scheme is suitable for the construction of a polynomial part by the so-called power basis. This approach has also been followed in the original BiCGstab(l) algorithm [14].

As we will see, this algorithm for the Bi-CG part produces explicitly the power basis $\hat{r}_0, A\hat{r}_0, \ldots, A^l\hat{r}_0$. For l steps of GMRES in the POL part we would need another l MVs to produce an orthonormal basis for the Krylov subspace $\mathcal{K}_{l+1}(A; \hat{r}_0)$, for which we have already a basis, and hence this would be very expensive. Of course, we could have orthogonalized the power basis afterwards in an



Scheme 1. Computational scheme for the "power basis variant" of BiCGstab(l) for $l = 2$.

attempt to mimic GMRES, but that would not have prevented the negative effects of a poorly conditioned power basis. Therefore, since GMRES is so attractive for stability reasons, we want to investigate a possible combination of Bi-CG and GMRES, i.e., we want to construct the basis for the POL part just as is done in GMRES. We will do this in section 5. There we construct the Bi-CG part with the help of $l-1$ matrix vector products with A^{T}, that have to be carried out only once at the beginning of the algorithm. This gives us all freedom in the construction of suitable polynomials for the POL part, in particular we may use the standard GMRES algorithm for the POL part.

The methods for the POL part that we will discuss in this section all employ the power basis and do not require additional MVs in the POL part.

4.1. *The Bi-CG part*

For $k = ml, j = 0, \ldots, l-1$ and $i = 0, \ldots, j+1$, we employ the Bi-CG relations (2) to increase the Bi-CG indices:

$$A^i Q_k u_{k+j} = A^i Q_k (r_{k+j} - \beta_{k+j} u_{k+j-1}) = A^i Q_k r_{k+j} - \beta_{k+j} A^i Q_k u_{k+j-1}$$

and

$$A^i Q_k r_{k+j+1} = A^i Q_k (r_{k+j} - \alpha_{k+j} A u_{k+j}) = A^i Q_k r_{k+j} - \alpha_{k+j} A^{i+1} Q_k u_{k+j}.$$

The Bi-CG coefficients can be computed from (8), for $j = 0, \ldots, l-1$, as

$$\beta_{k+j} = \frac{1}{\vartheta_{k+j}} \frac{\rho_{k+j}}{\sigma_{k+j-1}} \quad \text{and} \quad \alpha_{k+j} = \frac{\rho_{k+j}}{\sigma_{k+j}}, \tag{25}$$

where $\vartheta_{k+i} \equiv 1$ $(i = 1, \ldots, l-1)$ and $-\vartheta_k$ is the leading coefficient of p_{m-1},

$$\rho_{k+j} \equiv (A^j Q_k r_{k+j}, \tilde{r}_0) \quad \text{and} \quad \sigma_{k+j} \equiv (A^{j+1} Q_k u_{k+j}, \tilde{r}_0). \tag{26}$$

Note that σ_{k-1} is available from the previous inner iteration.

This algorithm for the Bi-CG part requires 1 MV/Kd (MV for increasing the Krylov dimension by 1), $\frac{1}{2}$ DOT/Kd and $\frac{1}{2}(l+2)$ AXPY/Kd (cf. section 7).

We obtain the vectors $A^{j+1} Q_k u_{k+j}$ and $A^{j+1} Q_k r_{k+j+1}$ (the vectors along the diagonal in the Bi-CG part in scheme 1) by matrix vector multiplications. These vectors serve a double purpose: they are used in the computation of the Bi-CG coefficients (see (25)–(26)), and through vector updates they lead to the vectors $A^i \hat{r}_0 = A^i Q_k r_{k+l}$ and $A^i \hat{u}_0 = A^i Q_k u_{k+l-1}$ (the vectors along the last row in the Bi-CG part in scheme 1). As soon as we have these vectors we can compute r_{k+l} and u_{k+l-1} without additional matrix multiplications.

4.2. *The polynomial part*

We will further assume that the vectors $A^j \hat{r}_0$ have been computed explicitly in the Bi-CG part. They will serve as basis vectors for the construction of suitable polynomials in the POL part of the hybrid schemes.

4.2.1. Minimal residual polynomials

In this subsection we discuss two approaches that produce, either implicitly or explicitly, the scalars

$$\gamma_1, \ldots, \gamma_l \text{ that minimize } \|\hat{r}_0 - \sum_{j=1}^{l} \gamma_j A^j \hat{r}_0\|_2 \text{ over all possible } \gamma_j. \tag{27}$$

By V we denote the $n \times l$ matrix with columns $A\hat{r}_0, \ldots, A^l\hat{r}_0$, and we write $\vec{\gamma} \equiv (\gamma_1, \ldots, \gamma_l)^T$. According to (27) $\vec{\gamma}$ is the least squares solution of the problem $V\vec{\gamma} = \hat{r}_0$. The vector r_{ml+l} is then obtained as in (15), and the corresponding approximation x_{ml+l} as in (17).

The normal equations and Cholesky. The vector $\vec{\gamma}$ is the solution of the normal equations:

$$(V^T V)\vec{\gamma} = V^T \hat{r}_0. \tag{28}$$

This equation can be solved efficiently by either Cholesky- or LU decomposition. The computation of $V^T V$ requires $\frac{1}{2}l(l+1)$ inner products, and $V^T\hat{r}_0$ requires l inner products. No other long vector operations are necessary for the computation of $\vec{\gamma}$. It is well-known that if V has a large condition number then problem (28) will be less stable than (27), since the square of the condition number of V is involved, even if $\hat{r}_0$ makes a small angle with the range of V [5].

Hence, an alternative may be to solve (27) directly by the QR method in order to improve stability.

QR with Gram–Schmidt orthogonalization. Let $V = QR$, where Q is an orthogonal $n \times l$ matrix and R is an $l \times l$ upper triangular matrix: $Q^T Q = D$ is diagonal. Then

$$\vec{\gamma} = R^{-1} D^{-1} Q^T \hat{r}_0. \tag{29}$$

This approach is more expensive than the "normal equations with Cholesky" approach because of the additional $\frac{1}{2}l(l+1)$ vector updates. For further details see [5].

Bunch–Kaufman decomposition. In finite precision arithmetic, the Cholesky decomposition as well as the modified Gram–Schmidt orthogonalization may lead to poor results when V is near rank deficient. Indeed, in some of our experiments, this occurred for large l ($l > 8$). This is not a big surprise, since the columns of the matrix V are generated by the power method.

Since it is not always strictly necessary to have an accurate solution of the least squares problem (see section 2), we have considered other approaches to avoid (near) breakdown of the computations. For instance, one might use Householder QR, with column pivoting. However, this is quite expensive and in our experience it did not improve the convergence behaviour of the iteration method. We had favourable experiences with the use of an LD_2L^T decomposition of $V^T V$ [2]. This is almost as stable as Gaussian elimination with complete pivoting and the computational costs are modest. Moreover, the accuracy of the iteration method

(see our definition in section 2.1) is better maintained, since additional back transformations for the computation of $\hat{u}_0$ and $\hat{x}_0$ are avoided, and this avoids another possible source for differences between the updated and the true residual.

Discussion. Our experiments did not show any clear difference that could be related to the stability of the least squares solution. Much to our surprise the POL part appears to be quite stable, when working with the notorious power basis.

Of course, we may expect with the QR method to obtain a solution $\vec{\gamma}$ that is more accurate than by solving the normal equations. However, as argued in section 2, the accuracy of the polynomial coefficients does not affect so much the speed of convergence nor the accuracy. We should have accurate vectors $A^i\hat{r}_0$ and $A^i\hat{u}_0$, since x_k and r_k are updated with these vectors, and these vectors have been produced already in the Bi-CG part. In the POL part we only construct a specific combination of these vectors with the purpose to further reduce the residual. Any perturbation in the coefficients of the POL representation is acceptable as long as it will lead to about the same reduction of the residual (the worst case would be when the residual is not reduced at all: stagnation, see section 2.2).

In some of our experiments we have observed extremely large condition numbers for the matrices V (10^{50}, say), but notwithstanding this both versions did quite well. To understand why these methods did so well even in such extreme circumstances, we should realize that we are not interested in $\vec{\gamma}$ but rather in $V\vec{\gamma}$ (and the polynomial values $p_m(A)\hat{r}_0$, $p_m(A)\hat{u}_0, \ldots$). Therefore, we may as well scale the minimization problem. That is, we may as well solve the least squares problem

$$\min_{\tilde{\gamma}} \| V D^{-1} \tilde{\gamma} - \hat{r}_0 \|,$$

where D is an $l \times l$ diagonal matrix with diagonal coefficients $\|A^i\hat{r}_0\|$ and $\tilde{\gamma} \equiv D\vec{\gamma}$. It is easily shown that the condition number of the scaled matrix $V D^{-1}$ dictates the stability of $V\vec{\gamma}$ even if we obtain $\vec{\gamma}$ through solving the unscaled least square problem.

In the reported experiments, the scaled matrices were all quite well-conditioned (condition numbers less than 10^5). We do not understand this remarkable phenomenon yet.

Nevertheless, we believe there is reason to discuss the possibility of avoiding the explicit use of the vectors $\gamma_i A^i\hat{r}_0$ and $\gamma_i A^i\hat{u}_0$, since these vectors tend to be large while their linear combinations $\sum \gamma_i A^i\hat{r}_0, \ldots$, are small in general. While the linear combination may have errors proportional to $\sum |\gamma_i| \|A^i\hat{r}_0\|$, these errors may be expected to be large (cf. sections 2 and 3.3).

In order to stabilize the algorithm we should avoid to create explicitly the power basis. If we construct an orthonormal basis for the Krylov subspace $\mathcal{K}_{l+1}(A; \hat{r}_0)$ right from the start, as in the Arnoldi method, the matrix in the resulting least squares problem (and the least squares solution) may be expected to be better conditioned.

In view of efficiency considerations, however, this approach would be attractive only if it does not increase the number of MVs in the inner iteration. In sections 6 and 7, we will show how this can be achieved.

4.2.2. Fixed polynomials

In this approach for the POL part, we select the same polynomial p (i.e. $p_m = p$) for all inner iterations (or for a number of consecutive inner iterations). We will show that this can be implemented rather efficiently, but of course it depends on the situation whether a selected fixed polynomial leads to a fast enough converging overall process. The following obvious choices come to mind.

If information on the spectrum of A is available one might select scaled and shifted Chebychev polynomials for p_m [9]. The required information might be extracted from the Bi-CG coefficients α_k, β_k.

Another possibility might be to proceed with the minimal residual polynomial of the previous inner iteration (if that polynomial has led to sufficient reduction). A similar approach, in connection with trying to improve parallel properties in GMRES, has been suggested in [1], see also [10].

If we have p_m explicitly available in factorized form

$$p_m = (1 - \zeta_1 t)(1 - \zeta_2 t) \ldots (1 - \zeta_m t),$$

at the beginning of the inner iteration, we can save on the computational costs and memory space.

For $k = ml$ write

$$Q_{k+j} = (I - \zeta_j A)Q_{k+j-1} = (I - \zeta_j A) \ldots (I - \zeta_1 A)Q_k.$$

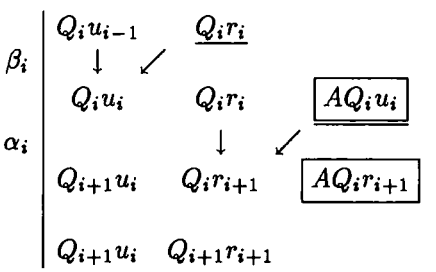
We can compute $Q_{k+j+1}u_{k+j}$ and $Q_{k+j+1}r_{k+j+1}$ from $Q_{k+j}u_{k+j-1}$ and $Q_{k+j}r_{k+j}$ as indicated in scheme 2 (where $i = k + j$). The computation of

$$Q_{k+j+1}u_k = Q_{k+j}u_{k+j} - \zeta A Q_{k+j}u_{k+j}$$

and

$$Q_{k+j+1}r_{k+j+1} = Q_{k+j}r_{k+j+1} - \zeta A Q_{k+j}r_{k+j+1}, \quad \text{with} \quad \zeta = \zeta_{j+1},$$

involves two vector updates. These updates are not indicated in the scheme.



Scheme 2. Computational scheme for the hybrid Bi-CG with polynomial update of degree one.

Of course, the computation of the zero's of p_m may be unstable. However, we are not so much interested in accurate zero's but rather in accurate polynomial values $p_m(A)\hat{r}_0$, and these values may be accurate even when the zero's are not accurate at all [19]. To keep the errors in the polynomial values small it may be necessary to pay attention to their enumeration (see, for instance, the algorithm of Leja [8]).

This approach requires the storage of only 6 vectors (the three vectors as indicated in the scheme plus $\hat{x}_0$, $\tilde{r}_0$ and b). Furthermore, the computation requires 3 AXPY/Kd and 1 MV/Kd, which is extremely efficient. One may avoid complex arithmetic if possible complex zero's come in complex conjugate pairs. In that case the polynomial can be factorized as a product of second degree polynomials with real coefficients. Scheme 2 can be adapted in a straightforward manner to a scheme that updates by second degree polynomials, i.e., a scheme that computes $Q_{k+2}u_{k+1}$ and $Q_{k+2}r_{k+2}$ from $Q_k u_{k-1}$ and $Q_k r_k$ using real arithmetic. This would not increase the computational cost per Krylov dimension, but it would require more memory space (8 vectors instead of 6).

5. The orthogonal basis

In scheme 3 we have represented, for $l = 2$, the second approach for the Bi-CG part. It turns out that this approach requires only 1 AXPY/Kd, but more memory space. We save MVs in the Bi-CG part, and this allows to consider other methods in the POL part. Actually, we now have to employ more expensive methods in the POL part, since the Krylov subspace can only be extended by one dimension by at least one multiplication by the matrix A. The advantage of this "column variant" is that it allows for the construction of an orthogonal basis for the polynomial part without additional MVs in the overall hybrid algorithm, as compared with the power basis approach.

5.2. *The Bi-CG part*

Again, we employ the Bi-CG relations in (2) to increase the Bi-CG indices:

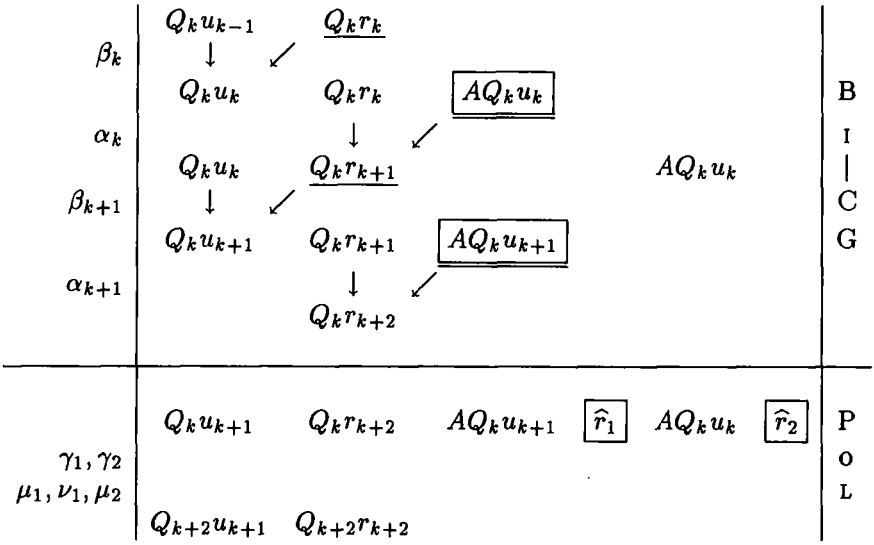
$$Q_k u_{k+j} = Q_k r_{k+j} - \beta_{k+j} Q_k u_{k+j-1} \tag{30}$$

and

$$Q_k r_{k+j+1} = Q_k r_{k+j} - \alpha_{k+j} A Q_k u_{k+j}, \tag{31}$$

leading to $\hat{r}_0 \equiv Q_k r_{k+l}$ and $\hat{u}_0 \equiv Q_k u_{k+l-1}$ ($\hat{x}_0$ is easily obtained by appropriate updates).

We have not available the vectors $A^j Q_k r_{k+j}$ and $A^j Q_k u_{k+j}$ that we used before for the computation of the Bi-CG coefficients (cf. (25) and (26)). Instead of the polynomial $t^j q_{ml}$ which was used in (26), we may as well take $\tilde{q} q_{ml}$ where $\tilde{q}$



Scheme 3. Computational scheme for the "orthogonal basis variant" of BiCGstab(l) for $l = 2$.

is a polynomial of degree j with leading coefficient 1. For instance, we may take $\tilde{\phi}_j$, where $\tilde{\phi}_j$ is the scaled jth Bi-CG polynomial ϕ_j. This is an obvious choice since ϕ_j can be easily constructed in the Bi-CG phase of the first inner iteration.

In algorithm 1 we have indicated how the shadow residuals $\tilde{r}_j$, for which $\tilde{r}_j \equiv \tilde{\phi}_j(A^T)\tilde{r}_0$, for $j = 0, \ldots, l-1$, can be obtained. Then β_{k+j} and α_{k+j} can be computed as in (25) with

$$\rho_{k+j} \equiv (Q_k r_{k+j}, \tilde{r}_j), \quad \sigma_{k+j} \equiv (AQ_k u_{k+j}, \tilde{r}_j), \tag{32}$$

instead of the relations (26).

Note that the $\tilde{r}_j$ have to be computed only once, at the start of the process. However, they have to be stored in memory for further inner iterations, and we need also $l-1$ multiplications with A^T.

Apparently, the $\hat{x}_0$, $\hat{r}_0 \equiv Q_{ml} r_{ml+l}$, $\hat{u}_0 \equiv Q_{ml} u_{ml+l-1}$ can be computed at the cost of $\frac{1}{2}$ MV/Kd, $\frac{1}{2}$ DOT/Kd, and 1 AXPY/Kd. The other $\frac{1}{2}$ MV/Kd can be used to compute in some convenient way the vectors $\hat{r}_1, \ldots, \hat{r}_l$ which form a basis for $\mathcal{K}_l(A; A\hat{r}_0)$.

It would make the algorithm expensive if another $\frac{1}{2}$ MV/Kd were necessary for the computation of the $A^i \hat{u}_0$. We will now show that these vectors can also be computed from the $\hat{r}_i$ and some intermediate results.

For the vectors $\hat{u}_0$ and u_{k+l-1} we need, according to (14) and (11), the vectors $A\hat{u}_0$, $A^2\hat{u}_0, \ldots, A^l\hat{u}_0$, but we wish to compute these from available vectors, avoiding extra MVs. This can be done by rewriting the Bi-CG recursions (2):

$$A^{i+1} Q_k u_j = \frac{1}{\alpha_j}(A^i Q_k r_j - A^i Q_k r_{j+1}) \tag{33}$$

choose x_0 and $\tilde{r}_0$,
compute $\mathbf{r}_0 = b - Ax_0$,
take $\mathbf{u}_{-1} = 0$, $\tilde{u} = 0$, $\rho_0 = 1$, $\alpha = 1$

For $j = 0, \ldots, \ell - 1$
$\rho_1 = (\mathbf{r}_j, \tilde{r}_j)$, $\beta = \beta_j = \alpha \frac{\rho_1}{\rho_0}$, $\rho_0 = \rho_1$
$\mathbf{u}_j = r_j - \beta \mathbf{u}_{j-1}$,
$\hat{u}_1 = A\mathbf{u}_j$, $\tilde{u} = \tilde{r}_j + \frac{\beta}{\alpha}\tilde{u}$
$\gamma = (\hat{u}_1, \tilde{r}_j)$, $\alpha = \alpha_j = \frac{\rho_0}{\gamma}$,
$\mathbf{x}_{j+1} = \mathbf{x}_j + \alpha \mathbf{u}_j$,
$\mathbf{r}_{j+1} = \mathbf{r}_j - \alpha \hat{u}_1$, $\tilde{r}_{j+1} = A^T \tilde{u} - \frac{1}{\alpha}\tilde{r}_j$
end

Algorithm 1. The computation of l Bi-CG residuals and shadow residuals.

and

$$A^i Q_k r_j = A^i Q_k u_j + \beta_j A^i Q_k u_{j-1} \tag{34}$$

From (33) we see that $A^{i+1} Q_k u_{k+l-1} (= A^{i+1} \hat{u}_0)$ (cf. (14)) can be computed from $A^i Q_k r_{k+l} (= A^i \hat{r}_0)$ (cf. (14)), and $A^i Q_k r_{k+l-1}$. The last vector can, through the recursive application of the expressions in (34) and (33) be expressed in combinations of $AQ_k u_k, \ldots, AQ_k u_{k+l-1}$, $A\hat{r}_0, \ldots, A^{i-1}\hat{r}_0$ and these vectors are available at the completion of the Bi-CG part. Hence

$$A^{i+1}\hat{u}_0 \in \mathcal{K}_i(A; A\hat{r}_0) \oplus \text{span}\{AQ_k u_k, \ldots, AQ_k u_{k+l-1}\}. \tag{35}$$

5.2. *The polynomial part*

In the POL part, we have to compute the element $\boldsymbol{r}_{k+l}$ in the Krylov subspace $\mathcal{K}_{l+1}(A; \hat{r}_0)$. Suppose for the moment that we have already a Krylov basis $\hat{r}_1, \ldots, \hat{r}_l$ for $\mathcal{K}_l(A; A\hat{r}_0)$ and that we have scalars $\eta_1, \ldots, \eta_l$ (see 5.2.4) such that

$$\boldsymbol{r}_{k+l} = \hat{r}_0 - \sum_{j=1}^{l} \eta_j \hat{r}_j. \tag{36}$$

We will explain how to compute efficiently (i.e. without additional MVs) $\boldsymbol{x}_{k+l}$, $\boldsymbol{u}_{k+l-1}$ and the leading coefficient $-\gamma_l$ of p_m. The update of $\hat{x}_0$ belongs to the span of $\hat{r}_0, \ldots, \hat{r}_{l-1}$. Hence

$$\boldsymbol{x}_{k+l} = \hat{x}_0 + \sum_{j=0}^{l-1} \tilde{\eta}_j \hat{r}_j \tag{37}$$

for certain scalars $\tilde{\eta}_0, \ldots, \tilde{\eta}_{l-1}$.

From (35) we know that there exist scalars $\mu_1, \ldots, \mu_l$ and $\nu_1, \ldots \nu_{l-1}$ such that

$$\boldsymbol{u}_{k+l-1} = \hat{u}_0 - \sum_{j=1}^{l-1} \nu_j \hat{r}_j - \sum_{j=0}^{l-1} \mu_{l-j} A Q_k \boldsymbol{u}_{k+j}. \tag{38}$$

The scalars $\tilde{\eta}_j$, μ_j and ν_j depend linearly on $\eta_1, \ldots, \eta_l$ and can be computed with the Bi-CG relations (2), as we will see in section 5.2.2, and this computation will involve only scalar operations.

For the formulation of computational details, we introduce the $l \times l$ upper triangular matrix $U = (u_{ij})$ for which

$$u_{11}\hat{r}_1 = A\hat{r}_0 \quad \text{and} \quad u_{jj}\hat{r}_j = A\hat{r}_{j-1} - \sum_{i=1}^{j-1} u_{ij}\hat{r}_i \quad \text{for } j = 2, \ldots, l. \tag{39}$$

Furthermore, for a given n-vector v, we introduce the $n \times l$ matrix $V[v]$ with column vectors $v_1, \ldots, v_l$ such that

$$u_{11}v_1 = Av \quad \text{and} \quad u_{jj}v_j = Av_{j-1} - \sum_{i=1}^{j-1} u_{ij}v_i \quad \text{for } j = 2, \ldots, l. \tag{40}$$

The expressions in (40) can be rewritten in matrix notation as

$$V[v]U = Ave_1^{\mathrm{T}} + AV[v]S, \tag{41}$$

where S is the $l \times l$ shift matrix defined by $Se_{j+1} = e_j$, $Se_1 = 0$.

5.2.1. The update for the approximate solution

Note that (36), on behalf of (39), can be rewritten as

$$\boldsymbol{r}_{k+l} = \hat{r}_0 - V[\hat{r}_0]\vec{\eta} \quad \text{with} \quad \vec{\eta} \equiv (\eta_1, \ldots, \eta_l)^{\mathrm{T}}. \tag{42}$$

Writing $\tilde{\eta} \equiv U^{-1}\vec{\eta}$ and $\eta_0 \equiv e_1^{\mathrm{T}}\tilde{\eta}$, it follows from (41) that

$$V[\hat{r}_0]\vec{\eta} = V[\hat{r}_0]U\tilde{\eta} = A(\eta_0\hat{r}_0 + V[\hat{r}_0]S\tilde{\eta}), \tag{43}$$

and since the update for $\hat{x}_0$ is the negative of the original of the update for $\hat{r}_0$, we have

$$\boldsymbol{x}_{k+l} = \hat{r}_0 + \eta_0\hat{r}_0 + V[\hat{r}_0](S\tilde{\eta}). \tag{44}$$

5.2.2. The update for the search direction

For the computation of $\boldsymbol{u}_{k+l-1}$, we might copy the construction for $\boldsymbol{r}_{k+l}$. From (15), (16) and (42), it follows that $\boldsymbol{u}_{k+l-1} = \hat{u}_0 - V[\hat{u}_0]\vec{\eta}$. However, we want to avoid the explicit computation of $V[\hat{u}_0]$, since this would require additional MVs.

Since, the matrices $V[v]$ depend linearly on v, it follows from the Bi-CG relations (2) that, for $j = 1, \ldots, l-1$,

$$AV[Q_k u_{k+j}] = \frac{1}{\alpha_{k+j}} (V[Q_k r_{k+j}] - V[Q_k r_{k+j+1}]) \tag{45}$$

and

$$V[Q_k r_{k+j}] = V[Q_k u_{k+j}] + \beta_{k+j} V[Q_k u_{k+j-1}]. \tag{46}$$

With (41) this leads to

$$\begin{aligned} V[Q_k u_{k+l-1}] = (AQ_k u_{k+l-1}) e_1^{\mathrm{T}} U^{-1} - \frac{1}{\alpha_{k+l-1}} V[Q_k r_{k+l}] SU^{-1} \\ + \frac{1}{\alpha_{k+l-1}} V[Q_k u_{k+l-1}] SU^{-1} + \frac{\beta_{k+l-1}}{\alpha_{k+l-1}} V[Q_k u_{k+l-2}] SU^{-1} \end{aligned} \tag{47}$$

and, for $j = 1, \ldots, l-2$,

$$\begin{aligned} V[Q_k u_{k+j}] = (AQ_k u_{k+j}) e_1^{\mathrm{T}} U^{-1} - \frac{1}{\alpha_{k+j}} V[Q_k u_{k+j+1}] SU^{-1} \\ + \frac{1-\beta_{k+j+1}}{\alpha_{k+j}} V[Q_k u_{k+j}] SU^{-1} + \frac{\beta_{k+j}}{\alpha_{k+j}} V[Q_k u_{k+j-1}] SU^{-1}. \end{aligned} \tag{48}$$

Since U is upper triangular, we see that SU^{-1} is nilpotent ($(SU^{-1})^l = 0$). This can be used to eliminate the $V[Q_k u_{k+j}]$ terms in the right hand side of (47). To that end we insert for these $V[Q_k u_{k+j}]$ the right hand side of the expressions (47)–(48). We keep doing this and this adds each time a factor SU^{-1}. Since $(SU^{-1})^l = 0$, we eventually have removed the terms with $V[Q_k u_{k+j}]$ from the right hand side, and we obtain an expression for $V[\hat{u}_0] = V[Q_k u_{k+l-1}]$ in terms of the $AQ_k u_{k+j}$, $V[\hat{r}_0] = V[Q_k r_{k+l}]$, the Bi-CG coefficients, and $(SU^{-1})^j$ $(j = 0, \ldots, l-1)$.

We obtain the required combination by multiplying these matrix expressions with $\vec{\eta}$. We will give the precise relations, since they are necessary in practical computation. We skip the straightforward but tedious derivation.

Let M be the $l \times l$ upper triangular matrix given by $Me_j \equiv (T_k)^{j-1} e_1$, where T_k is the $l \times l$ tri-diagonal matrix of Bi-CG coefficients formed in the preceding Bi-CG part (except for the enumeration of the basis, T_k is the $l \times l$ right-lower block of the $(k+l)$th order Lanczos matrix of the Bi-CG process):

$$T_k \equiv \begin{bmatrix} \frac{1}{\alpha_{k+l-1}} & \frac{-1}{\alpha_{k+l-2}} & 0 & \cdots \\ \frac{\beta_{k+l-1}}{\alpha_{k+l+1}} & \frac{1-\beta_{k+l-1}}{\alpha_{k+l-2}} & \frac{-1}{\alpha_{k+l-3}} & \ddots \\ 0 & \frac{\beta_{k+l-2}}{\alpha_{k+l-2}} & \frac{1-\beta_{k+l-2}}{\alpha_{k+l-3}} & \ddots \\ \vdots & \ddots & \ddots & \ddots \end{bmatrix}.$$

We also introduce the $l \times l$ matrix E that is given by $Ee_j \equiv (SU^{-1})^{j-1}\vec{\eta}$ and we denote $\vec{\mu} = (\mu_1, \ldots, \mu_l)^T \equiv M(U^{-1}E)^T e_1$.

The expression for $V[\hat{u}_0]\vec{\eta}$ turns out to be

$$V[\hat{u}_0]\vec{\eta} = V[\hat{r}_0]\vec{\nu} + \sum_{j=0}^{l-1} \mu_{l-j} A Q_k u_{k+j}, \quad \text{where} \quad \vec{\nu} \equiv -E\frac{S^T M^T e_1}{\alpha_{k+l-1}}, \tag{49}$$

and hence

$$u_{k+l-1} = \hat{u}_0 - V[\hat{r}_0]\vec{\nu} - \sum_{j=0}^{l-1} \mu_{l-j} A Q_k u_{k+j}. \tag{50}$$

5.2.3. The coefficients of the polynomial p_m

The leading coefficient $-\gamma_l$ of p_m can be computed as follows.

First observe that $\hat{r}_j - u_{jj}^{-1} A\hat{r}_{j-1} \in \mathcal{K}_j(A; \hat{r}_0)$ for $j = 1, \ldots, l$, and hence

$$\hat{r}_j - (u_{jj}u_{j-1\,j-1}\cdots u_{11})^{-1} A^j \hat{r}_0 \in \mathcal{K}_j(A; \hat{r}_0), \tag{51}$$

which shows that

$$\gamma_l = \eta_l/(u_{ll}u_{l-1\,l-1}\cdots u_{11}). \tag{52}$$

If we wish to keep the same polynomial $p_m(t) = 1 - \sum_{j=1}^{l} \gamma_j t^j$ for a number of consecutive sweeps (see section 5.2.2), we also need the other coefficients γ_i. These coefficients can be computed with the matrix E defined in section 6.2.2. Because, by (41) and (42), we have that

$$\begin{aligned}\sum_{j=1}^{l} \gamma_j A^j \hat{r}_0 = V[\hat{r}_0]\vec{\eta} &= A\hat{r}_0 e_1^T U^{-1}\vec{\eta} + AV[\hat{r}_0]SU^{-1}\vec{\eta} \\ &= A\hat{r}_0 e_1^T U^{-1} E e_1 + A(A\hat{r}_0 e_1^T U^{-1} + AV[\hat{r}_0]SU^{-1})Ee_2 \\ &= \cdots = \sum_{j=1}^{l} A^j \hat{r}_0 e_1^T U^{-1} E e_j.\end{aligned}$$

Apparently, $\gamma_j = (U^{-1}Ee_j, e_1)$ for $j = 1, \ldots, l$. Hence

$$\vec{\gamma} \equiv (\gamma_1, \ldots, \gamma_l)^T = E^T (U^{-1})^T e_1. \tag{53}$$

5.2.4. An orthogonal basis for the polynomial part

As previously explained in this section we can compute the updates for $\hat{r}_0$, $\hat{x}_0$ and $\hat{u}_0$ if we have explicitly a Krylov basis of $\mathcal{K}_l(A; A\hat{r}_0)$. Since we have saved $\frac{1}{2}$ MV per Krylov dimension in the Bi-CG part (as compared with the power basis variant of BiCGstab(l)) we may use the MVs to build a more well conditioned Krylov basis.

An obvious way is to construct an orthogonal basis for $\mathcal{K}_l(A; A\hat{r}_0)$, in the same way as is done in ORTHODIR [20].

ORTHODIR constructs an orthonormal basis $\hat{r}_1, \ldots, \hat{r}_l$ for $\mathcal{K}_l(A; A\hat{r}_0)$, where $u_{11}\hat{r}_1 \equiv A\hat{r}_0$ with $u_{11} \equiv \|A\hat{r}_0\|$. If we apply l steps of ORTHODIR for the equation $Ay = \hat{r}_0$ then the resulting residual is the desired r_{k+l}, and by construction $r_{k+l} \perp \mathcal{K}_l(A; A\hat{r}_0)$.

Since the $\hat{r}_j$ form an orthonormal basis, we have that

$$r_{k+l} = \hat{r}_0 - V[\hat{r}_0]\vec{\eta} \quad \text{with} \quad \vec{\eta} \equiv V[\hat{r}_0]^{\mathrm{T}}\hat{r}_0, \quad \text{(cf.(42))} \tag{54}$$

and we can then update x_{k+l} (cf. (44)) and u_{k+l-1} (cf. (50)).

6. A stabilized matrix-variant of BiCGstab(*l*)

As observed in section 2, it would improve the stability in the polynomial part if we could generate some well conditioned basis for $\mathcal{K}_{l+1}(A; \hat{r}_0)$. With the orthogonal basis version (section 5), we may even produce an orthogonal basis. Unfortunately, for efficiency reasons the update of the search direction in the POL part is computed by a scheme (50) that differs from the scheme that is used for the update of the residual (36). That is, in finite precision the polynomial $\tilde{p}_m$, for which $u_{k+l} = \tilde{p}_m(A)\hat{u}_0$, is very likely to be different from the polynomial p_m for which $r_{k+l} = p_m(A)\hat{r}_0$, whereas these polynomials should be the same (cf. (13)).

This difference is also very likely to be larger than the difference in the power basis approach (section 4). Consequently, although in the column approach we may expect more accurate approximations, the convergence may be slower (cf. section 2.2).

Note that it does not help to construct an orthogonal basis from the vectors $A^i\hat{r}_0$, since the condition number of the power basis will affect the stability of any computational method that uses this basis explicitly.

In this section, we try to hit two birds by one stone. In the Bi-CG part, we try to construct a reasonably well conditioned basis for $\mathcal{K}_{l+1}(A; \hat{r}_0)$, such that we can use the same scheme for the computation of the residual and the search direction (without increasing the number of MVs, of course).

The construction that we will present is based on the following idea. Suppose that U_{m-1} describes the orthogonal basis of the $(m-1)$th inner iteration (as in section 5.2.4). Then, in the mth inner iteration we generate basis vectors in the Bi-CG part for $\mathcal{K}_l(A; A\hat{r}_0)$, following the rules dictated by U_{m-1}, and we orthogonalize this basis afterwards to get the U_m for the next inner iteration. Hopefully this gives a reasonably well conditioned basis for the POL part in each inner iteration. This was supported by our numerical experiments.

6.1. *The Bi-CG part*

For $k = ml$, we consider some $l \times l$ upper triangular matrix $U_{m-1} = (u_{ij}^{(m-1)})$

(typically U_{m-1} will describe the orthogonal basis of the previous inner iteration). The matrix U_{m-1} will be updated in the mth BiCGstab(l) sweep. We will show that we can construct in the mth inner iteration in the Bi-CG part a basis $\hat{r}_1, \ldots, \hat{r}_l$ for $\mathcal{K}_l(A; A\hat{r}_0)$ which satisfies the orthogonality relations of the previous inner iteration:

$$u_{11}^{(m-1)}\hat{r}_1 = A\hat{r}_0$$

and (55)

$$u_{jj}^{(m-1)}\hat{r}_j = A\hat{r}_{j-1} - \sum_{i=1}^{j-1} u_{ij}^{(m-1)}\hat{r}_i \quad \text{for } j = 2, \ldots, l.$$

For each vector v (of dimension n) and each $j = 1, \ldots, l$, let $V_j[v]$ be the $n \times j$ matrix with columns $v_1, \ldots, v_j$ such that $v_1, \ldots, v_l$ satisfy (40). By multiplying (41) at the right by S^T, we see that

$$V_j[v]H^{(j)} = AV_{j-1}[v] \quad \text{for } j = 2, \ldots, l. \tag{56}$$

where $H^{(j)}$ is the $j \times (j-1)$ left upper block of $U_{m-1}S^T$; $H^{(j)}$ is a Hessenberg matrix.

The matrices $V_j[v]$ depend linearly on v. Therefore, for $j = 0, \ldots, l-1$, in view of the relations in (2), we have that

$$\begin{aligned} Q_k u_{k+j} &= Q_k r_{k+j} - \beta_{k+j} Q_k u_{k+j-1}, \\ V_j[Q_k u_{k+j}] &= V_j[Q_k r_{k+j}] - \beta_{k+j} V_j[Q_k u_{k+j-1}] \quad \text{if } j > 0, \end{aligned} \tag{57}$$

and, since $AV_j[Q_k u_{k+j}] = V_{j+1}[Q_k u_{k+j}]H^{(j+1)}$ (see (56)),

$$\begin{aligned} Q_k r_{k+j+1} &= Q_k r_{k+j} - \alpha_{k+j} u_{11}^{(m-1)} V_{j+1}[Q_k u_{k+j}]e_1, \\ V_j[Q_k r_{k+j+1}] &= V_j[Q_k r_{k+j}] - \alpha_{k+j} V_{j+1}[Q_k u_{k+j}]H^{(j+1)} \quad \text{if } j > 0. \end{aligned} \tag{58}$$

If we have the scalar β_{k+j}, we can use (57) to compute $V_j[Q_k u_{k+j}]$. Next, at the expense of one MV and j AXPYs, we can compute the last column of $V_{j+1}[Q_k u_{k+j}]$ (cf. (40)). Then, if we have the scalar α_{k+j}, (58) leads to $V_j[Q_k r_{k+j+1}]$. Again at the cost of one MV and j AXPYs, we can compute a new column vector of this matrix (cf. (40)) and find $V_{j+1}[Q_k r_{k+j+1}]$. We can repeat this iteration step: increase j by 1, etcetera, until $j = l-1$. Eventually, we will have $\hat{u}_0$, $\hat{r}_0$, $V_l[\hat{u}_0]$, $V_l[\hat{r}_0]$, and also $\hat{x}_0$.

6.1.1. The computation of the Bi-CG coefficients

We will now explain how to compute the Bi-CG coefficients α_{k+j} and β_{k+j} or, equivalently (see (25)), the scalars ρ_{k+j} and σ_{k+j}.

As in (51), we have that

$$V_j[Q_k r_{k+j}]e_j - (u_{jj}^{(m-1)} u_{j-1j-1}^{(m-1)} \cdots u_{11}^{(m-1)})^{-1} A^j Q_k r_{k+j}$$

belongs to the Krylov subspace $\mathcal{K}_j(A; Q_k r_{k+j})$. Since $\tilde{r}_0$ is orthogonal to this space, it follows that

$$\rho_{k+j} \equiv (A^j Q_k r_{k+j}, \tilde{r}_0) = u_{jj}^{(m-1)} \cdot \ldots \cdot u_{11}^{(m-1)} \cdot (V_j[Q_k r_{k+j}]e_j, \tilde{r}_0). \tag{59}$$

Similarly, $V_j[Q_k u_{k+j}]e_j - (u_{jj}^{(m-1)} \cdot \ldots \cdot u_{11}^{(m-1)})^{-1} A^j Q_k u_{k+j}$ belongs to $\mathcal{K}_j(A; Q_k u_{k+j})$. Since $A\mathcal{K}_j(A; Qu_{k+j})$ is orthogonal to $\tilde{r}_0$, we see that

$$\sigma_{k+j} \equiv (A^{j+1} Q_k u_{k+j}, \tilde{r}_0) = u_{jj}^{(m-1)} \cdot \ldots \cdot u_{11}^{(m-1)} \cdot (A(V_j[Q_k u_{k+j}]e_j), \tilde{r}_0). \tag{60}$$

Note that we also have to compute the vector $A(V_j[Q_k u_{k+j}]e_j)$ explicitly in order to expand the matrix $V_j[Q_k u_{k+j}]$ with one new column.

6.2. *The polynomial part*

We will update $\hat{r}_0$, $\hat{x}_0$ and $\hat{u}_0$ in a similar way as for the ORTHODIR approach (see section 5.2.4).

Although we may expect that $V_l[\hat{r}_0]$ is well conditioned, we may not expect that it is orthonormal, but we may orthonormalize this matrix by (modified) Gram–Schmidt: for some non-singular $l \times l$ upper triangular matrix R we have

$$V_l[\hat{r}_0] = \hat{V}R \quad \text{where} \quad \hat{V}^T\hat{V} = I. \tag{61}$$

The columns of $\hat{V}$ form an orthonormal basis of $\mathcal{K}_l(A; A\hat{r}_0)$. Hence

$$r_{k+l} = \hat{r}_0 - \hat{V}\vec{\eta} \quad \text{where} \quad \vec{\eta} \equiv \hat{V}^T\hat{r}_0. \tag{62}$$

Since $\hat{V}\vec{\eta} = V_l[\hat{r}_0]R^{-1}\vec{\eta}$, we see that

$$u_{k+l} = \hat{u}_0 - V_l[\hat{u}_0]R^{-1}\vec{\eta}. \tag{63}$$

Because the columns of $\hat{V}$ form an orthonormal Krylov basis of $\mathcal{K}_l(A; A\hat{r}_0)$, it follows that

$$\hat{V}U_m = A\hat{r}_0 e_1^T + A\hat{V}S \tag{64}$$

for some $l \times l$ upper triangular matrix U_m, that will be used as near-orthogonalization matrix for the $(m+1)$th inner iteration. Straightforward computation leads to

$$U_m = RU_{m-1}(e_1 e_1^T + S^T RS)^{-1}. \tag{65}$$

We can use U_m to further update $\hat{x}_0$, because (cf. (44))

$$x_{k+l} = \hat{x}_0 + \eta_0\hat{r}_0 + \hat{V}(S\tilde{\eta}) \quad \text{where} \quad \tilde{\eta} \equiv U_m^{-1}\vec{\eta}. \tag{66}$$

Finally, formula (52) gives the leading coefficient $-\gamma_l$ of the ORTHODIR polynomial p_m:

$$\gamma_l = \eta_l/(u_{ll}^{(m)} \cdot \ldots \cdot u_{11}^{(m)}) \quad \text{where} \quad \eta_l \equiv (\vec{\eta}, e_l). \tag{67}$$

Initialization. As in all variants of BiCGstab(l) we have to choose an initial approximation x_0 and a shadow residual $\tilde{r}_0$. However, here we also need an initial $l \times l$ upper triangular matrix U_0. In our experiments we have considered three strategies, which we will describe now.

First choose a shadow residual $\tilde{r}_0$ and an approximation x_0'.

By x_0'' and U', we denote the approximation and the upper triangular matrix, respectively, produced by l steps of ORTHODIR with initial approximation x_0'. Our three different strategies can then be characterized as:

(i) $x_0 = x_0'$ and $U_0 = I$;
(ii) $x_0 = x_0''$ and $U_0 = U'$;
(iii) $x_0 = x_0'$ and $U_0 = U'$.

All strategies turned out to be equally accurate, except for the first step, that is: in each step, except for the first one, each of the strategies gave a deviation of the updated residual from the true residual of comparable size. The first step is rather important for the accuracy, since the norm of the deviation may be expected to be proportional to the norm of the residual (cf. section 2).

For certain problems, the inexpensive strategy (i) leads to a matrix R in the first step with a condition number that is relatively large (as compared with the condition number of the R of the subsequent steps). For these problems, strategy (ii) leads to a first matrix R that is somewhat better conditioned but still may have a relatively large condition number, while the first matrix R of strategy (iii) is relatively well conditioned.

Discussion. The matrix version of the BiCGstab(l) algorithm is more expensive than the power basis version. In the Bi-CG part we need more vector updates (AXPYs). However, any of the computational steps involving long vectors can be implemented with the BLAS2 subroutine _GEMV (or even the BLAS3 subrouting _GEMM). If the matrix $V_l[\hat{r}_0]$, with which we start the polynomial part, is well conditioned then modified Gram–Schmidt does not lead to more accurate results than Gram–Schmidt.

7. The costs of BiCGstab(l) variants

Table 1 gives the average cost to increase the Krylov subspace dimension by one for the separate parts of the algorithms. For the Bi-CG part we have counted the long vector operations for this part (in schemes 1 and 3 the operations before the horizontal line): in the POL part we also took into account the vector updates needed for the linear combination to form $r_{k+l}, \ldots$. All scalar operations have been neglected. "Power basis", "Orthogonal basis", and "Stabilized matrix" refers to the implementation of BiCGstab(l) as discussed in sections 4, 5, and 6, respectively. The maximum number of long vectors that have to be stored (including b) is listed in the last column of table 1 in the rows for the POL part (since the required memory space is not additive (as is the computational cost) we have not listed the required space per part).

Table 1
The average cost per Krylov dimension.

Method	Computational cost			Memory requirement
	MV	AXPY	DOT	
Bi-CG	2	13/2	2	7
CGS	1	13/4	1	7
Bi-CGSTAB	1	3	2	7
BiCGstab(l)				
Power basis	1	$(l+2)/2$	1	
Normal eqs.	0	3/2	$(l+3)/4$	$2l+5$
Gram–Schmidt	0	$(l+5)/4$	$(l+3)/4$	$2l+5$
Orthogonal basis	1/2	3/2	1	
ORTHODIR	1/2	$(l+5)/4$	$(l+3)/4$	$3l+4$
Stabilized matrix	1	$(l^2+9l+8)/12$	1	
ORTHODIR	0	$(l+5)/4$	$(l+3)/4$	$2l+5$

In contrast to the other approaches, the orthogonal basis approach requires a few multiplications by A^T ($l-1$ for the complete iteration process). This may be a serious drawback (when A^T is not easily available) which can not always be compensated by the fact that the orthogonal basis approach is inexpensive as far as it concerns AXPYs and DOTs per Krylov dimension.

8. Numerical experiments

In this section we will report on some numerical experiments that illustrate the effects of different implementations of BiCGstab(l). We are particularly interested in the stability of the different implementations and in the consequences for accuracy and for the rate of convergence.

All computations have been carried out on a Sun Sparc ELC workstation in double precision with a relative machine precision $\xi \approx 1.1 \times 10^{-16}$.

8.1. *The accuracy level*

The accuracy that an iterative method can achieve is significantly influenced by the size of the residual during the iterations (see section 2).

In figure 1 we have plotted the convergence history of the *true* and the *updated residual* for CGS and for the three different variants of BiCGstab(4) for the test matrix Sherman4 of the Harwell–Boeing collection. In figure 2 similar convergence histories are given for $l = 8$. The power basis variant has been implemented with an LD_2L^T decomposition (see also section 8.1.1). The stabilized matrix variant was initialized with a redundant ORTHODIR step (see also section 8.1.2).

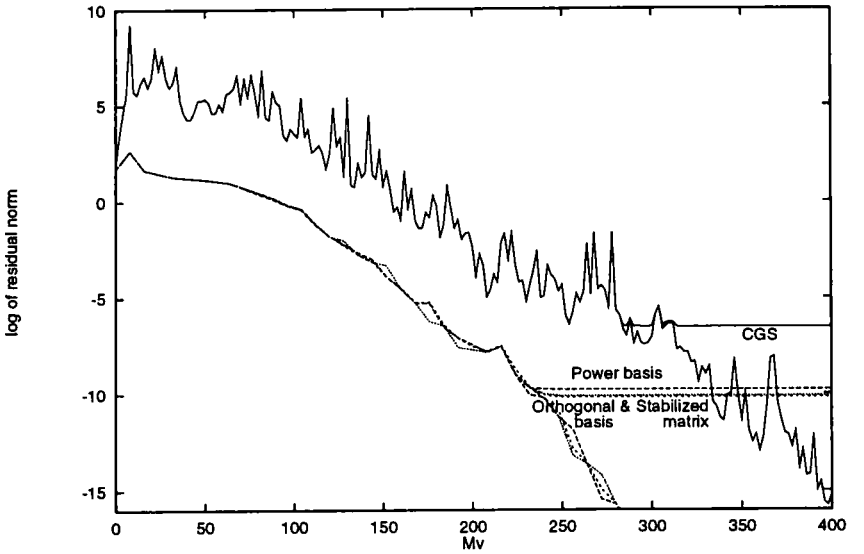


Fig. 1. Sherman4, 3786 nonzeros, $l = 4$.

ORTHODIR was implemented with modified Gram–Schmidt. No preconditioning has been used in this example.

We see how the convergence history of the *updated residual* and the *true residual* drift apart after some point. After this point the convergence history of the *true residual* levels off and becomes stationary. We will call this level the *accuracy level.*

The accuracy of CGS is in agreement with our discussion in section 2. Indeed, the difference between the largest residual norm and the *accuracy level* is approximately $1/\bar{\xi}$. Since the convergence of BiCGstab(l) is smoother, one may expect a more

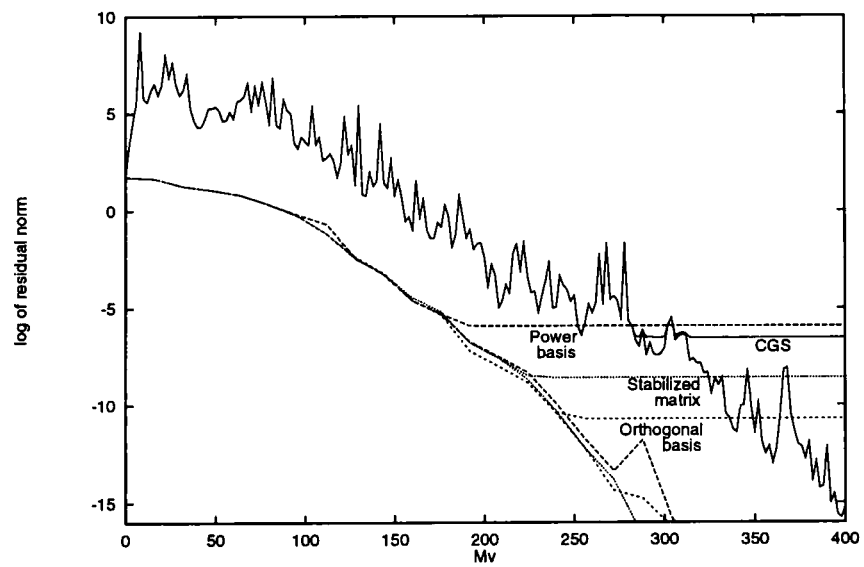


Fig. 2. Sherman4, 3786 nonzeros, $l = 8$.

accurate *true residual*. On the other hand, in the Bi-CG part as well as in the POL part, some accuracy may have been lost, due to the computation of intermediate residuals (see section 3.3) and evaluation errors.

In figure 1 we see that all BiCGstab(4) variants perform better than CGS. We also see that there is not much difference between the three implementations. This is in contrast with figure 2 which shows a much different situation. Here, the power basis variant of BiCGstab(8) is even worse than CGS. Apparently the use of a power basis of degree 8 spoils the accuracy dramatically. As expected the orthogonal basis variant performs quite satisfactorily: it is accurate while maintaining the same rate of convergence. The performance of the stabilized matrix variant is, as expected, somewhere in between the other two implementations of BiCGstab(8).

In subsections 8.1.1–8.1.3, we will compare experimentally the accuracy of several variants of the three implementations of BiCGstab(l) for $l = 2, 4, 8, 16$. We will report on experiments for the test matrix Sherman3 (Harwell–Boeing collection) with ILU(0) preconditioning. Motivated by the results shown in previous figures, we have terminated the iterations as soon as $|\log(\|b - Ax_k\|_2/\|r_k\|_2)| > 0.1$: at that stage we may expect that the *accuracy level* has been reached.

8.1.1. Power basis implementations

Out of the many available methods to solve the least squares problem in the minimum residual part of BiCGstab(l) we have considered three possible ones (cf. section 4.2.1):

- Solve the normal equations with a Cholesky decomposition CC^T;
- Solve the normal equations with an LD_2L^T decomposition with diagonal pivoting, i.e. D_2 is block diagonal with block sizes 1×1 or 2×2 [2];
- Solve the LS-problem with a Modified Gram–Schmidt QR decomposition.

We might expect that these three approaches will not differ much for moderate l. For larger values of l the power basis may become (near) rank deficient. In that case, solving the LS-problem with an LD_2L^T decomposition may help to reduce the effects of near singularity.

In table 2 we have illustrated this for the Sherman3 test matrix. The results in this table show the *accuracy level* (a) of a specific implementation and the number of MVs at which this *accuracy level* is achieved.

Table 2
BiCGstab(l): Sherman3, 20033 nonzeros, ILU(0), a = $\log \|b - Ax_k\|_2$.

l	CC^T		LD_2L^T		Mod. GS QR	
	a	MV	a	MV	a	MV
2	−13.40	228	−13.61	220	−13.55	212
4	−13.09	216	−12.23	208	−12.62	208
8	−9.08	128	−9.08	128	−9.08	128
16	breakdown		−4.27	96	−2.80	64

Table 3
BiCGstab(l): Sherman3, 20033 nonzeros, ILU(0), a = $\log \|b - Ax_k\|_2$.

l	$x_0 = x_0'$, $U_0 = I$		$x_0 = x_0''$, $U_0 = U'$		$x_0 = x_0'$, $U_0 = U'$	
	a	MV	a	MV	a	MV
2	−13.60	216	−13.40	216	−13.67	220
4	−13.26	208	−10.64	152	−13.60	216
8	−9.02	128	−7.97	112	−12.47	208
16	−2.72	64	−2.53	32	−10.27	160

We see that for $l = 16$ the Cholesky decomposition breaks down. In this case the LD_2L^T decomposition with pivoting appears to be a useful alternative.

In table 2 we also observe that the more expensive QR decomposition does not result in a significantly better *accuracy level*.

We conclude that for reasons of efficiency (and *also* accuracy) the power basis variant, in combination with an LD_2L^T decomposition, is an attractive implementation of BiCGstab(l) for moderate values of l.

8.1.2. Stabilized matrix implementations

In section 6 the stabilized matrix variant of BiCGstab(l) has been proposed as an alternative for the power basis approach for larger values of l. It was anticipated that this approach would lead to a well conditioned basis and thus to more accuracy in the Bi-CG as well as in the POL part. Furthermore, the number of iterations should be comparable with the power basis implementation. In table 3 we have shown the importance of the choice for x_0 and U_0. In our experience, $x_0 = x_0'$ and $U_0 = U'$ is the best choice (see section 6.2). Note, however, that the stabilized matrix variant may be quite expensive (see section 7).

8.1.3. Comparing the accuracy

Although experimental evidence indicates that the accuracy is improved with the stabilized matrix variant, this approach is not so attractive since it is expensive. The orthogonal basis variant seems to be a good alternative. The average cost for this variant is less than the average cost for the power basis variant, but, unfortunately, its memory requirement is about 50% more. Also it is not clear whether the number of iterations will be (more or less) the same. In table 4 we give an overview of the results for the three different implementations.

For moderate l ($l \leqslant 4$) all implementations are comparable. For larger l the power basis variant and the stabilized matrix variant are less accurate, while the *accuracy level* of the orthogonal basis variant is almost invariant. With respect to accuracy, the orthogonal basis variant is certainly an improvement, but it seems to display a somewhat slower rate of convergence. This effect is more clearly seen in our example in section 8.2.

Table 4
BiCGstab(l): Sherman3, 20033 nonzeros, ILU(0), a = $\log \|b - Ax_k\|_2$.

l	Power basis		Orthogonal basis		Stabilized matrix	
	a	MV	a	MV	a	MV
2	−13.45	216	−13.49	220	−13.67	220
4	−12.31	208	−13.48	216	−13.60	216
8	−9.06	128	−13.39	224	−12.47	208
16	−4.15	96	−13.48	192	−10.27	160

8.2. *Comparing the rate of convergence*

For this comparison we have taken the matrix that arises from a (66 × 66) finite volume discretization of

$$-u_{xx} - u_{yy} + 1000(xu_x + yu_y) + 10u = f \tag{68}$$

on the unit cube with Dirichlet boundary conditions. This example has been suggested in [11] (but we have not used a preconditioner).

We have chosen the right hand side b such that the solution x of the equation $Ax = b$ is the vector $(1, 1, \ldots, 1)^T$. The zero vector was used as an initial guess.

In order to show the effect of different implementations on the rate of convergence we have used a standard stopping criterion: the iteration was terminated as soon as $\|r_k\|_2/\|r_0\|_2 < 10^{-12}$.

In table 5 we have listed the norms of the *updated residual* (u), the *true residual* (t), and the required number of matrix–vector products (MV).

The power basis variant was implemented with the LD_2L^T decomposition for the normal equations. The stabilized matrix variant was implemented with a redundant ORTHODIR step in order to initialize x_0 and U_0.

Our results seem to support our heuristic arguments on the accuracy and on the rate of convergence (see section 2). With respect to accuracy, the orthogonal basis variant is the most accurate and the power basis variant is the least accurate (as l increases).

With respect to the rate of convergence, i.e. the number of MVs, the stabilized matrix is the best one, and the number of MVs is largest for the orthogonal basis variant.

Table 5
BiCGstab(l): PDE problem, 20224 nonzeros, u = $\log \|r_k\|_2$, t = $\log \|b - Ax_k\|_2$.

l	Power basis			Orthogonal basis			Stabilized basis		
	u	t	MV	u	t	MV	u	t	MV
2	−10.74	−10.74	1300	−10.46	−10.46	1336	−10.61	−10.61	1236
4	−10.95	−10.95	1096	−11.10	−11.10	984	−10.84	−10.81	944
8	−11.83	−11.26	928	−10.67	−10.67	1648	−11.36	−11.32	832
16	−10.46	−7.84	992	−10.91	−10.44	2240	−10.67	−10.65	768

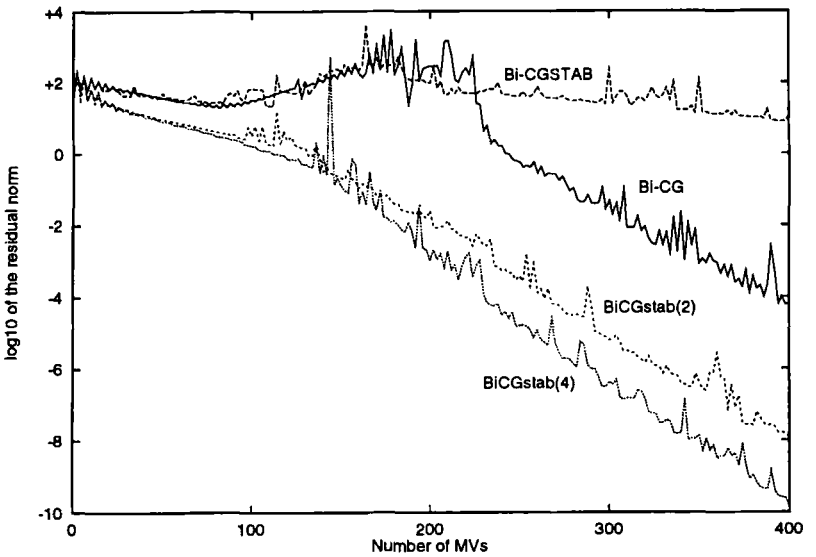


Fig. 3. Using Bi-CGstab(l) coefficients.

8.3. *Errors in the iteration coefficients*

As argued in section 2.2, we expect relatively large errors in the Bi-CG coefficients that are produced by BiCGstab(l) when the minimal residual step in the POL part makes little progress in some phase of the iteration process. This will most likely affect the speed of convergence from then on.

We have taken the same problem as in section 8.2, but now the right hand side f in equation (68) is such that $u(x, y) = \exp(xy)\sin(\pi x)\sin(\pi y)$ is the solution, and we have taken a grid size 33×33. Also in this case we have not used a preconditioner.

The number of MVs along the horizontal axis in figures 3 and 4 includes, in the case of Bi-CG, the number of multiplications with A^{T}.

In figure 3 we see that Bi-CGSTAB seems to stagnate, obviously due to a poor GMRES(1) reduction in a number of steps. If we replace the Bi-CG coefficients in the Bi-CGSTAB process by the coefficients that would have been computed in the standard Bi-CG method[1] then we see that the Bi-CGSTAB and Bi-CG are doing about equally well, as figure 4 shows (and as might have been expected in situations where the additional GMRES(1) steps give no additional improvement).

Apparently, the stagnation in Bi-CGSTAB is caused by poorly recovered Bi-CG

[1] That is: instead of computing the α_k and β_k in BiCGstab(l), we simply take the α_k and β_k computed in a separate run of Bi-CG itself (i.e. not combined with any other polynomial method). We did not change the other computational steps of BiCGstab(l). Of course this highly artificial approach would be silly in practical circumstances, but here it helps to make visible the effect of inaccurate Bi-CG coefficients.

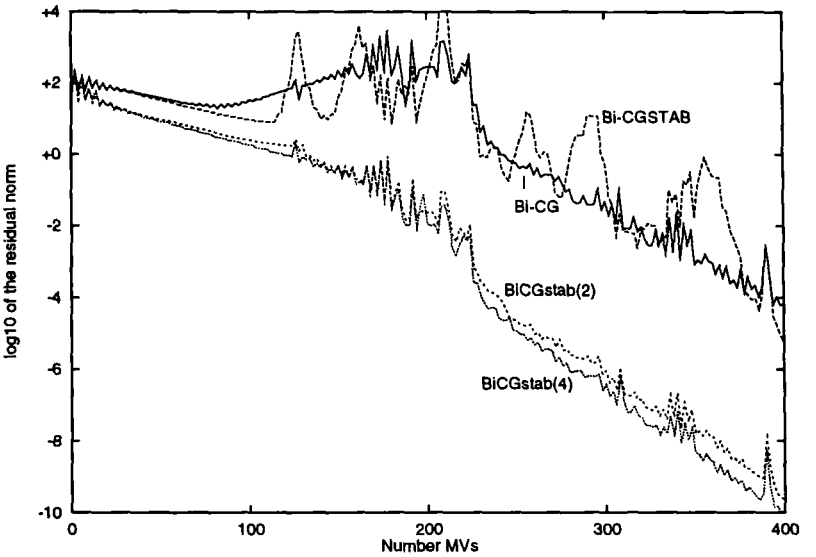


Fig. 4. Using the Bi-CG coefficients.

coefficients, BiCGstab(2) converges faster than Bi-CG: apparently the minimal residual steps (or order 2) in the POL part give some reduction, and hence we might expect also a better accuracy in the Bi-CG iteration coefficients (better than with Bi-CGSTAB).

But also with BiCGstab(2) the coefficients are not recovered as well as they should have been for optimal convergence, since we observe faster convergence when inserting the Bi-CG coefficients from Bi-CG itself in BiCGstab(2). About the same improvement can be obtained by using BiCGstab(4), with its "own" Bi-CG coefficients. Apparently, with BiCGstab(4) the GMRES(4) steps give enough reduction as to make an accurate enough recovery of the iteration coefficients possible.

9. Conclusions

We have shown that BiCGstab(l) can be implemented in different ways. The choice of the implementation is important for the overall performance of the algorithm. We have proposed several specific implementations and we have discussed some of their properties.

Three vectors have to be updated in BiCGstab(l): the approximate solution, the residual, and the search direction. It has been shown that it is difficult to maintain both accuracy and optimal speed of convergence. This problem is inherent to the nature of the respective updates, and one can not completely avoid it. Of course, one may try to find a compromise for which we have offered suggestions.

We have illustrated, with numerical examples, some consequences of different

implementations. In our experiences, the power basis version does quite well for moderate values of l. For larger values of l this approach leads to unacceptable loss of accuracy and instabilities, since the power basis can become (nearly) rank deficient. In some cases this problem can be alleviated with an LD_2L^T decomposition instead of a Cholesky decomposition for the determination of the minimal residual reduction.

If memory space is not a limiting factor then a more robust implementation may be used: the orthogonal basis variant. This variant is even less expensive in terms of computational complexity, but it can lead to a slower convergence rate due to inaccurate search directions. A compromise is the so-called stabilized matrix variant. Unfortunately, this implementation is rather expensive for larger values of l.

For a given linear system it is difficult, if not impossible, to predict which implementation of BiCGstab(l) should be selected. In our experiences, the orthogonal basis variant for BiCGstab(4) seems to be a good candidate for a robust iterative solver (with appropriate preconditioning, of course).

Acknowledgements

We thank both referees for their suggestions that helped us to improve the presentation of our ideas.

References

[1] Z. Bai, D. Hu and L. Reichel, A Newton basis GMRES implementation, University of Kentucky, Technical Report 91-03 (1991).

[2] J.R. Bunch and L. Kaufman, Some stable methods for calculating inertia and solving symmetric linear systems, Math. Comp. 31 (1977) 162–179.

[3] R. Fletcher, Conjugate gradient methods for indefinite systems, in: *Proc. Dundee Biennial Conf. on Numerical Analysis*, ed. G. Watson (Springer, New York, 1975).

[4] R.W. Freund and N.M. Nachtigal, A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems, RIACS, NASA Ames Research Center, Technical Report 91.18 (1991).

[5] G.H. Golub and C.F. Van Loan, *Matrix Computations*, 2nd ed. (The Johns Hopkins University Press, Baltimore and London, 1989).

[6] M.H. Gutknecht, Variants of BiCGStab for matrices with complex spectrum, SIAM J. Sci. Comput. 14 (1993) 1020–1033.

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

[8] F. Leja, Sur certaines suites liées aux ensemble plans et leur application à la representation conforme, Ann. Polon. Math. 4 (1957) 8–13.

[9] T.A. Manteuffel, The Tchebychev iteration for nonsymmetric linear systems, Numer. Math. 28 (1977) 307–327.

[10] N.M. Nachtigal, L. Reichel and L.N. Trefethen, A hybrid GMRES algorithm for non-symmetric linear systems, SIMAX 13 (1992) 796–825.

[11] Y. Saad, A flexible inner-outer preconditioned GMRES algorithm, SIAM J. Sci. Statist. Comput. 14 (1993) 461–469.
[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) 856–869.
[13] W. Schönauer, *Scientific Computing on Vector Computers* (North-Holland, Amsterdam/New York/Oxford/Tokyo, 1987).
[14] G.L.G. Sleijpen and D.R. Fokkema, BiCGstab(*l*) for linear equations involving matrices with complex spectrum, ETNA 1 (1993) 11–32.
[15] P. Sonneveld, CGS, a fast Lanczos-type solver for nonsymmetric linear systems, SIAM J. Sci. Statist. Comput. 10 (1989) 36–52.
[16] K. Turner and H.F. Walker, Efficient High Accuracy Solutions with GMRES(*m*), SIAM J. Sci. Statist. Comput. 13 (1992) 815–825.
[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) 631–644.
[18] R. Weiss, Convergence behavior of generalized conjugate gradient methods, PhD thesis, University of Karlsruhe (1990).
[19] J.H. Wilkinson, *The Algebraic Eigenvalue Problem* (Oxford University Press, Oxford, 1965).
[20] D.M. Young and K.C. Jea, Generalized conjugate-gradient acceleration of nonsymmetrizable iterative methods, Lin. Alg. Appl. 34 (1980) 159–194.
[21] L. Zhou and H.F. Walker, Residual smoothing techniques for iterative methods, SIAM J. Sci. Comput. 15 (1994) 297–312.