SIAM J. SCI. COMPUT.
Vol. 15, No. 1, pp. 89–105, January 1994

A FAMILY OF QUASI-MINIMAL RESIDUAL METHODS FOR NONSYMMETRIC LINEAR SYSTEMS*

CHARLES H. TONG[†]

Abstract. The quasi-minimal residual (QMR) algorithm by Freund, Gutknecht, and Nachtigal [*Research Institute for Advanced Computer Science Tech. Report* 91.09, NASA Ames Research Center, Moffett Field, CA; *Numer. Math.*, 60 (1991), pp. 315–339], in addition to its ability to avoid breakdowns, also improves the irregular convergence behavior encountered by the biconjugate gradient (BiCG) method through the incorporation of quasi minimization. This paper studies this quasi-minimal residual approach applied, for simplicity, to the nonlook-ahead version of the QMR method. In particular, a family of quasi-minimal residual methods is defined where on both ends lie the original QMR method and a minimal residual method while lying in between are the variants derived here. These variants are shown to consume a little more computational work per iterations than the original QMR method, but generally give an even better convergence behavior as well as lower iteration counts. The same idea was also applied to a recently proposed transpose-free quasi-minimal residual method. In particular, a few quasi-minimal residual variants of the Van der Vorst BiCGSTAB method are derived. Numerical results are presented comparing the quasi-minimal residual and transpose-free quasi-minimal residual variants derived in this paper.

Key words. Lanczos method, quasi-minimal residual approach, nonsymmetric linear systems, transpose-free methods

AMS subject classifications. 65F10, 65N20

1. Introduction.

Conjugate gradient-like methods for nonsymmetric problems generally fall into three classes: orthogonalization methods, e.g., GMRES [14]; biorthogonalization methods, e.g., biconjugate gradient [6], [11]; and methods based on normal equation approach, e.g., conjugate gradient normal residual (CGNR) [10]. Due to its short recurrences, and thus lower storage requirements, the class of biorthogonalization methods has been studied extensively in recent years. The earliest algorithm in this class is the biconjugate gradient (BiCG) algorithm, which was first proposed by Lanczos [11], and then rediscovered by Fletcher [6]. Despite its advantage, BiCG also suffers from a few well-known drawbacks. These drawbacks are the need for adjoint matrix vector multiplication ($A^T y$), the possibility of breakdowns or near breakdowns, and its irregular convergence behavior. Subsequently, transpose-free methods such as conjugate gradient squared (CGS) [16], BiCGSTAB [18], etc., were developed to alleviate the need for adjoint matrix vector multiplies. To address the breakdown or near-breakdown problem, Freund, Gutknecht, and Nachtigal [8], [9] proposed the quasi-minimal residual (QMR) algorithm, which is based on a look-ahead Lanczos procedure to overcome curable breakdowns. The quasi-minimization step, in addition to curing one of the breakdown problems, also gives a smoother convergence curve. Such a quasi-minimization step can be applied to the transpose methods such as CGS and BiCGSTAB, giving rise to the recently developed algorithms such as Freund's transpose-free QMR algorithm [7], the transpose-free QMR method of Chan, De Pillis, and Van der Vorst [3], and QMRCGSTABx algorithms of Chan et al. [4].

While Faber and Manteuffel [5] proved that generally there exists no conjugate gradient-type scheme that incorporates both short recurrences and a minimization property, the quasi-minimal residual approach represents a step toward a compromise between these two conflicting but desirable features. This paper shows that a family of quasi-minimal residual methods (for simplicity, the nonlook-ahead version is used here) can be defined that is based on the use of different weight matrices (to be defined later) in the quasi-minimization step. We show that

*Received by the editors April 28, 1992; accepted for publication (in revised form) February 15, 1993.

[†]Center for Computational Engineering, Sandia National Laboratory, Livermore, California 94551-0969 (chtong@california.sandia.gov).

when this weight matrix is very sparse (i.e., diagonal), we have a simplified (nonlook-ahead) version of the Freund, Gutknecht, and Nachtigal quasi-minimal residual method. On the other hand, if we allow the weight matrix to be very dense (i.e., upper triangular), we have a minimal residual method. However, this minimal residual method suffers again from long recurrences. The variants in this paper are derived by allowing the weight matrices to be other than the two extremes mentioned above. In particular, we consider weight matrices that are block diagonal having upper triangular blocks. We show that these QMR (and BiCGSTAB-based transpose-free QMR) variants generally give smoother convergence curves and lower iteration counts with increasing block sizes at the expense of little extra computation per iteration.

The outline of this report is as follows. In §2, the basic ideas of the QMR approach are described and a few QMR variants are then derived. In §3, implementation details of the QMR variants as well as their computational complexity are given. In §4, numerical results are presented and discussed. In §5, we consider the transpose-free QMR algorithms and then derive a few quasi-minimal residual variants of the Van der Vorst BiCGSTAB method [18]. In §6, implementation details of the transpose-free QMR variants are given. In §7, numerical results are presented and discussed. Finally, in §8, some concluding remarks are made.

2. A family of quasi-minimal residual algorithms.

2.1. The quasi-minimal residual approach. We give a nonsingular and nonsymmetric $N \times N$ matrix A and two initial nonzero vectors $v_1 \in \Re^N$ and $w_1 \in \Re^N$ such that $w_1^T v_1 \neq 0$. The classical nonsymmetric Lanczos procedure, in exact arithmetic and in the absence of breakdowns, generates two sequences of vectors

$$V_n = \{v_1\ v_2\ \cdots\ v_n\} \quad \text{and} \quad W_n = \{w_1\ w_2\ \cdots\ w_n\}, \tag{2.1}$$

which satisfy the following two properties. (i) The two sequences span two Krylov subspaces. That is,

$$\text{span}\{v_1, v_2, \ldots, v_n\} = \text{span}\{v_1, Av_1, \ldots, A^{n-1}v_1\} = K_n(v_1, A), \tag{2.2}$$

$$\text{span}\{w_1, w_2, \ldots, w_n\} = \text{span}\{w_1, A^T w_1, \ldots, (A^T)^{n-1}w_1\} = K_n(w_1, A^T). \tag{2.3}$$

(ii) The two sequences satisfy a biorthogonality condition

$$w_i^T v_j = \delta_{ij}, \tag{2.4}$$

where δ_{ij} is the Kronecker symbol (that is, $\delta_{ij} = 1$, when $i = j$, and 0 otherwise).

The nonsymmetric Lanczos procedure can be written in matrix form as

$$AV_n = V_{n+1}H_n^e, \tag{2.5}$$

where H_n^e is a $(n+1) \times n$ tridiagonal matrix defined by parameters generated in the Lanczos process.

This nonsymmetric procedure can be used to find the solution of a linear system $Ax = b$. In particular, let $x_0 \in \Re^N$ be any initial guess. The initial residual vector is then $r_0 = b - Ax_0$. If we let $v_1 = r_0/ \parallel r_0 \parallel$ and w_1 be arbitrary with unit 2-norm (for example, $w_1 = v_1$), then the iterate x_n (solution at iteration n) is

$$x_n = x_0 + V_n z, \ z \in \Re^N. \tag{2.6}$$

Moreover, after some manipulations, the residual vector can be written as

$$r_n = V_{n+1}\Omega_{n+1}^{-1}(d_n - \Omega_{n+1}H_n^e z), \tag{2.7}$$

where Ω_{n+1} is an $(n+1) \times (n+1)$ diagonal weight matrix (following the Freund and Nachtigal notation [9]), $e_1^{(n+1)} = [1\ 0\ \cdots\ 0]^T \in \Re^{n+1}$, $\rho_0 = \| r_0 \|$, and $d_n = \omega_1 \rho_0 e_1^{(n+1)}$.

Instead of minimizing the 2-norm of the residual $\| r_n \|$, which takes prohibitively large number of arithmetic operations, the QMR approach minimizes only the Euclidean norm of the expression $(d_n - \Omega_{n+1} H_n^e z)$. The solution of this least squares problem can be computed by a standard QR decomposition of $\Omega_{n+1} H_n^e$ giving

$$\Omega_{n+1} H_n^e = Q_n^T \begin{bmatrix} R_n \\ 0 \end{bmatrix}, \tag{2.8}$$

where Q_n is an orthogonal $(n+1) \times (n+1)$ matrix, and R_n is a nonsingular upper triangular $n \times n$ matrix. This gives

$$\min_{z \in \Re^n} \| d_n - \Omega_{n+1} H_n^e z \|_2 = \min_{z \in \Re^n} \left\| \left(Q_n d_n - \begin{bmatrix} R_n \\ 0 \end{bmatrix} z \right) \right\|_2 . \tag{2.9}$$

If we let t_n be an $n \times 1$ vector consisting of the first n elements of the $(n+1) \times 1$ vector $Q_n d_n$, then z_n and the solution x_n are given by

$$z_n = R_n^{-1} t_n \quad \text{and} \quad x_n = x_0 + V_n z_n, \tag{2.10}$$

respectively.

2.2. Generalization of the QMR approach. In the simplified (nonlook-ahead) QMR algorithm, the weight matrix Ω_{n+1} was chosen to be a diagonal matrix. In general, Ω_{n+1} does not have to be diagonal. In fact, by allowing Ω_{n+1}^{-1} to be upper triangular, it is possible to construct an Ω_{n+1} such that $V_{n+1}\Omega_{n+1}^{-1}$ is orthonormal. This gives a minimal residual algorithm since

$$\begin{aligned} \min_{z \in \Re^n} \| r_n \|_2 &= \min_{z \in \Re^n} \| V_{n+1} \Omega_{n+1}^{-1} (\rho_0 \Omega_{n+1} e_1^{(n+1)} - \Omega_{n+1} H_n^e z) \|_2 \\ &= \min_{z \in \Re^n} \| \rho_0 \Omega_{n+1} e_1^{(n+1)} - \Omega_{n+1} H_n^e z \|_2 . \end{aligned} \tag{2.11}$$

However, since now the matrix $\Omega_{n+1} H_n^e$ is dense upper Hessenberg instead of tridiagonal, the computation of z becomes very expensive and takes up a prohibitively large amount of storage. This algorithm is in fact similar to GMRES except that it is susceptible to breakdowns.

Another alternative is to find a sparse Ω_{n+1} (other than diagonal) so that $V_{n+1}\Omega_{n+1}^{-1}$ becomes closer to orthonormal at the expense of requiring to store and use a few more of the previous direction vectors. A simple variant is to make Ω_{n+1}^{-1} block diagonal where each block is a 2×2 upper triangular submatrix. This can be accomplished by orthonormalizing consecutive pairs of v_i's, i.e., orthonormalize v_2 against v_1, v_4 against v_3, and so on. We call this new weight matrix $(B_{n+1}^{(2)})^{-1}$ and then we solve a different least squares problem

$$\| d_n - (B_{n+1}^{(2)})^{-1} H_n^e z_n \|_2 = \min_{z \in \Re^n} \| d_n - (B_{n+1}^{(2)})^{-1} H_n^e z \|_2, \tag{2.12}$$

where d_n is the same as before ($(B_{n+1}^{(2)})^{-1} d_n = d_n$ since all except the first element of d_n is nonzero and $B_{n+1}^{(2)}(1,1) = 1$). As B_{n+1} is block diagonal with a block size of 2, $(B_{n+1}^{(2)})^{-1} H_n^e$ will have one more superdiagonal than H_n^e. Thus only one additional direction vector needs to be kept for computing the solution update compared to the QMR method. Also, as will be shown in later sections, this algorithm takes only one more inner product and one more daxpy

($y = ax + b$, where x, y, b are vectors and a is a scalar) operation per iteration compared to the QMR method. We call this QMR variant BQMR(2) in subsequent sections.

We can further generalize this approach by allowing the weight matrix to be a block diagonal matrix of block size 3, 4, and so on. In this paper, we implement up to a block size of 3 (which we call BQMR(3) and the weight matrix for this case is $(B_{n+1}^{(3)})^{-1}$). It becomes obvious that when the block size is $n+1$, $V_{n+1}B_{n+1}^{(n+1)}$ has orthonormal columns, and we have a minimal residual algorithm similar to GMRES. When the block size is 1, we have the simplified (based on the classical Lanczos procedure) QMR algorithm (to be consistent in the use of notations, the simplified QMR algorithm will also be called BQMR(1) in subsequent sections).

2.3. A residual bound for the BQMR(k) algorithms. Freund and Nachtigal [9] have given a residual bound for the QMR algorithm based on the the following theorem [9].

THEOREM 2.1. *Suppose that the $m \times m$ matrix H_m generated by m steps of the* QMR *algorithm is diagonalizable* (*m denotes the termination index of the look-ahead Lanczos, for details refer to* [9]), *and set*

$$H = \Omega_{m-1} H_m \Omega_{m-1}^{-1}. \tag{2.13}$$

Then, for $n = 1, 2, \ldots, m-1$, the residual vectors of the QMR *algorithm satisfy*

$$\| r_n \| \leq \| r_0 \| \kappa(H)\sqrt{n+1}\, \vartheta_n \max_{j=1,\ldots,n+1} (\omega_1/\omega_j), \tag{2.14}$$

where

$$\kappa(H) = \min_{X: X^{-1}HX \text{diagonal}} \| X \| \cdot \| X^{-1} \|, \tag{2.15}$$

and

$$\vartheta_n = \min_{P\in\Pi_n : P(0)=1} \max_{\lambda\in\lambda(A)} |P(\lambda)|. \tag{2.16}$$

This convergence theorem is contingent on the use of a look-ahead Lanczos procedure. The formulation of BQMR(k) in the last section, however, is based on the classical Lanczos procedure. Thus this theorem does not hold for BQMR(k), in general, because of the possibility of breakdowns. Nevertheless, suppose that for a given test problem, BQMR(k) does not encounter breakdowns of the type $w_n^T v_n = 0$, $v_n \neq 0$, $w_n \neq 0$. Then we can derive a similar convergence bound for the BQMR(k) algorithms. Since this bound is rather restricted or not rigorous, we will state it as a remark instead of a theorem.

Remark. Suppose an $N \times N$ linear system is given and in exact arithmetic, the BQMR(k) for some $k \leq N$ terminates at iteration $m \leq N$ with $\| v_{m+1} \| = 0$ or $\| w_{m+1} \| = 0$ (note that this index m is different from that defined in [9]), and no breakdown of the type $v_n^T w_n = 0$, $v_n \neq 0$, $w_n \neq 0$ occurs at iterations 1 through m. Furthermore, suppose the $m \times m$ matrix H_m generated by m steps of the BQMR(k) algorithm is diagonalizable, and set

$$H = (B_{m-1}^{(k)})^{-1} H_m B_{m-1}^{(k)}. \tag{2.17}$$

Then, for $n = 1, 2, \ldots, m-1$, the residual vectors of the BQMR(k) algorithm satisfy

$$\| r_n \| \leq \| r_0 \| \kappa(H) \sqrt{\left\lceil \frac{n+1}{k} \right\rceil}\, \vartheta_n, \tag{2.18}$$

where

$$\vartheta_n = \min_{P\in\Pi_n : P(0)=1} \max_{\lambda\in\lambda(A)} |P(\lambda)|. \tag{2.19}$$

Proof. The proof of this remark is similar to that of the Freund and Nachtigal theorem stated above. In this theorem, the factor $\sqrt{n+1}$ arises from bounding $\| V_{n+1} \|$. A tighter bound (by a factor of $\approx \sqrt{k}$) can be prescribed by realizing that $V_{n+1}B_{n+1}^{(k)}$ is k-orthonormal (columns 1 through k are orthonormal, columns k+1 through 2k are orthonormal, etc.). Moreover, since H_n^e is tridiagonal and $B_{n+1}^{(k)}$ is block diagonal with upper triangular blocks and nonzero diagonal elements, $(B_{n+1}^{(k)})^{-1}H_n^e B_{n+1}^{(k)}$ is upper Hessenberg with nonzero subdiagonal. Using this fact, together with the proof in [9], we obtain the same ϑ_n as in [9], and thus we arrive at the above remark. $\square$

The above remark is given to shed some light on the convergence behavior of these QMR variants. This remark basically implies that the larger the block size (k) used, the tighter are the residual bounds if no breakdown occurs. In the limit when $k = n + 1$ (or BQMR(∞)) and in absence of breakdowns, the residual bound is the same as the GMRES residual bound.

3. Implementation and complexity of the BQMR(k) algorithms.

3.1. Implementation details.

3.1.1. The BQMR(2) algorithm. Considering the BQMR(2) algorithm, solving the least squares problem

$$\| d_n - (B_{n+1}^{(2)})^{-1}H_n^e z_n \|_2 = \min_{z\in\Re^n} \| d_n - (B_{n+1}^{(2)})^{-1}H_n^e z \|_2 \tag{3.1}$$

basically involves three steps.

Step 1. Update $B_{n+1}^{(2)}$.

Step 2. Update the relatively sparse matrix $\tilde{H}_n^e = (B_{n+1}^{(2)})^{-1}H_n^e$.

Step 3. Perform QR decomposition on $\tilde{H}_n^e$ to get R_n, and apply the same Q_n to d_n to get $t_n = [\tau_1 \cdots \tau_n]^T$.

The solution can then be updated by

$$x_n = x_{n-1} + V_n R_n^{-1}[0 \ \cdots \ 0 \ \tau_n]^T. \tag{3.2}$$

Since the update processes for the first and second step are different for odd and even iterations, we shall describe them separately.

At the odd iterations, $B_{n+1}^{(2)}$ and H_n^e are of the form

$$B_{n+1}^{(2)} = \begin{bmatrix} B_n^{(2)} & \begin{matrix} 0 \\ \vdots \\ \delta_{n+1} \end{matrix} \\ 0 \ \ldots & \epsilon_{n+1} \end{bmatrix}, \quad H_n^e = \begin{bmatrix} H_{n-1}^e & \begin{matrix} 0 \\ \vdots \\ \beta_n \\ \alpha_n \end{matrix} \\ 0 \ \ldots \ 0 & \gamma_{n+1} \end{bmatrix}. \tag{3.3}$$

The new entries in $B_{n+1}^{(2)}$, namely, δ_{n+1} and ϵ_{n+1} can be computed by orthonormalizing v_{n+1} against v_n.

Let the nth column of H_n^e be denoted by $h_n = [0 \ \ldots \ \beta_n \ \alpha_n \ \gamma_{n+1}]^T$ (which has a vector of length $n + 1$). Then, at the nth iteration, $\tilde{H}_n^e$ can be updated by performing a U-solve (solving a linear system where the matrix is upper triangular), that is, to find a vector u_n such that

$$B_{n+1}^{(2)}u_n = h_n, \tag{3.4}$$

where u_n will have the form $[0 \cdots 0\ \mu_4\ \mu_3\ \mu_2\ \mu_1]^T$.

At the even iterations, the derivation can be carried out similarly. Here no orthonormalization needs to be performed. Also $B^{(2)}_{n+1}$ is in a different form:

$$B^{(2)}_{n+1} = \begin{bmatrix} & & 0 \\ & B^{(2)}_n & \vdots \\ & & 0 \\ 0 & \ldots & 1 \end{bmatrix}. \tag{3.5}$$

Again, a U-solve is performed on $B^{(2)}_{n+1} u_n = h_n$, where now the fourth-to-the-last entry (μ_4 in the odd case) in u_n is 0.

Step 3 first applies the previous Givens rotations to u_n's so that

$$Q_{n-1}u_n = [0 \cdots 0\ \nu_4\ \nu_3\ \nu_2\ t_1\ \mu_1]^T. \tag{3.6}$$

Next a new Givens rotation is used to zero out the entry corresponding to μ_1 giving

$$Q_n u_n = [0 \cdots 0\ \nu_4\ \nu_3\ \nu_2\ \nu_1\ 0]^T, \tag{3.7}$$

which is the last column of R_n. The computation of $t_n = Q_n d_n$ can be performed the same way as in BQMR(1).

Finally, the expression $V_n R_n^{-1}[0 \cdots 0\ \tau_n]^T$ needs to be evaluated. This can be achieved by defining a sequence of vectors y_i where

$$Y_n = [y_1\ y_2 \cdots y_n] = V_n R_n^{-1}, \tag{3.8}$$

and the solution can be updated by

$$x_n = x_{n-1} + \tau_n * y_n. \tag{3.9}$$

It is straightforward to show that

$$y_n = (v_n - \nu_2 y_{n-1} - \nu_3 y_{n-2} - \nu_4 y_{n-3})/\nu_1. \tag{3.10}$$

In [9], it was shown that the BQMR(1) residual can be updated by

$$r_n^{\text{BQMR(1)}} = s_{n+1}^2 r_{n-1}^{\text{BQMR(1)}} - c_{n+1}\tilde{\tau}_{n+1} v_{n+1}. \tag{3.11}$$

The BQMR(2) residual can be updated slightly differently by

$$r_n^{\text{BQMR(2)}} = s_{n+1}^2 r_{n-1}^{\text{BQMR(2)}} - c_{n+1}\tilde{\tau}_{n+1} \hat{v}_{n+1}, \tag{3.12}$$

where $\hat{v}_{n+1}$ is the $(n+1)$th column of $V_{n+1}B^{(2)}_{n+1}$.

The pseudocode for the BQMR(2) algorithm is presented in the following.

```
ALGORITHM BQMR(2)
(1) Initialization
    (a) $r_0 = b - Ax_0$; $v_1 = r_0 / \| r_0 \|$; $w_1$ arbitrary
    (b) $y_{-2} = y_{-1} = y_0 = v_{-1} = v_0 = w_{-1} = w_0 = 0$; $r_0^{\text{BQMR(2)}} = r_0$
    (c) $\tilde{\tau}_1 = \| r_0 \|$; $s_{-2} = s_{-1} = s_0 = 0$; $c_{-2} = c_{-1} = c_0 = -1$
    (d) $\epsilon_0 = 1$; $\delta_0 = \beta_1 = \gamma_1 = 0$
(2) **For** $n = 1, 2, \ldots$ do
```

```
        (a) $\alpha_n = w_n^T A v_n$
        (b) $\tilde{v}_{n+1} = Av_n - \alpha_n v_n - \beta_n v_{n-1}$
        (c) $\tilde{w}_{n+1} = A^T w_n - \alpha_n w_n - \gamma_n w_{n-1}$
        (d) $\gamma_{n+1} = \| \tilde{v}_{n+1} \|$; $\beta_{n+1} = \tilde{w}_{n+1}^T \tilde{v}_{n+1}/\gamma_{n+1}$
        (e) $v_{n+1} = \tilde{v}_{n+1}/\gamma_{n+1}$; $w_{n+1} = \tilde{w}_{n+1}/\beta_{n+1}$;
    (3) U-solve to get the modified H matrix
        (a) if n is odd
            $\bar{v}_{n+1} = v_{n+1} - (v_{n+1}^T v_n)v_n$; $\hat{v}_{n+1} = \bar{v}_{n+1}/ \| \bar{v}_{n+1} \|$
            $\epsilon_{n+1} = 1/ \| \bar{v}_{n+1} \|$; $\delta_{n+1} = -(v_{n+1}^T v_n) * \epsilon_{n+1}$
            $\mu_1 = \gamma_{n+1}/\epsilon_{n+1}$; $\mu_2 = \alpha_n - \delta_{n+1} * \mu_1$; $\mu_3 = \beta_n/\epsilon_{n-1}$; $\mu_4 = -\delta_{n-1} * \mu_3$
        (b) else if n is even
            $\mu_1 = \gamma_{n+1}$; $\mu_2 = \alpha_n/\epsilon_n$; $\mu_3 = \beta_n - \delta_n * \mu_2$; $\mu_4 = 0$
            $\hat{v}_{n+1} = v_{n+1}$
    (4) Perform quasi minimization
        (a) $\nu_4 = s_{n-2} * \mu_4$; $t_3 = -c_{n-2} * \mu_4$
        (b) $\nu_3 = -c_{n-1} * t_3 + s_{n-1} * \mu_3$; $t_2 = -s_{n-1} * t_3 - c_{n-1} * \mu_3$
        (c) $\nu_2 = -c_n * t_2 + s_n * \mu_2$; $t_1 = -s_n * t_2 - c_n * \mu_2$
        (d) if $(|\mu_1| > |t_1|)$ $t_a = -t_1/\mu_1$; $s_{n+1} = 1/\sqrt{1+t_a^2}$; $c_{n+1} = c_{n+1} * t_a$
        (e) else $t_a = -\mu_1/t_1$; $c_{n+1} = 1/\sqrt{1+t_a^2}$; $s_{n+1} = c_{n+1} * t_a$
        (f) $\nu_1 = -c_{n+1} * t_1 + s_{n+1} * \mu_1$; $\tau = -c_{n+1}\tilde{\tau}$; $\tilde{\tau} = -s_{n+1}\tilde{\tau}$
        (g) $y_n = (v_n - \nu_2 y_{n-1} - \nu_3 y_{n-2} - \nu_4 y_{n-3})/\nu_1$
    (5) Update solution and residual
        (a) $x_n = x_{n-1} + \tau y_n$
        (b) $r_n^{\mathrm{BQMR(2)}} = s_{n+1}^2 r_{n-1}^{\mathrm{BQMR(2)}} - c_{n+1}\tilde{\tau}\hat{v}_{n+1}$
    (6) **End for**
```

3.1.2. The BQMR(3) algorithm. The BQMR(3) algorithm can be derived in a similar way. The pseudocode for the BQMR(3) algorithm is presented here.

```
    ALGORITHM BQMR(3)
    (1) Initialization
        (a) $r_0 = b - Ax_0$; $v_1 = r_0/ \| r_0 \|$; $w_1$ arbitrary
        (b) $y_{-3} = y_{-2} = y_{-1} = y_0 = v_{-1} = v_0 = w_{-1} = w_0 = 0$; $r_0^{\mathrm{BQMR(3)}} = r_0$
        (c) $\tilde{\tau}_1 = \| r_0 \|$; $s_{-3} = s_{-2} = s_{-1} = s_0 = 0$
        (d) $c_{-3} = c_{-2} = c_{-1} = c_0 = -1$; $\epsilon_0 = \epsilon_{-1} = 1$; $\delta_0 = \delta_{-1} = \eta = 0$
    (2) **For** $n = 1, 2, \ldots$ do
        same as step (2a)–(2e) in Algorithm BQMR(2)
    (3) U-solve to get the modified H matrix
        (a) if mod(n,3) = 1
            $\bar{v}_{n+1} = v_{n+1} - (v_{n+1}^T v_n)v_n$; $\hat{v}_{n+1} = \bar{v}_{n+1}/ \| \bar{v}_{n+1} \|$
            $\epsilon_{n+1} = 1/ \| \bar{v}_{n+1} \|$; $\delta_{n+1} = -(v_{n+1}^T v_n) * \epsilon_{n+1}$
            $\mu_1 = \gamma_{n+1}/\epsilon_{n+1}$; $\mu_2 = \alpha_n - \delta_{n+1} * \mu_1$; $\mu_3 = \beta_n/\epsilon_{n-1}$
            $\mu_4 = -\delta_{n-1} * \mu_3/\epsilon_{n-2}$; $\mu 5 = -\delta_{n-2} * \mu_4 - \eta * \mu_3$
        (b) else if mod(n,3) = 2
            $\bar{v}_{n+1} = v_{n+1} - (v_{n+1}^T v_{n-1})v_{n-1} - (v_{n+1}^T \hat{v}_n)\hat{v}_n$; $\hat{v}_{n+1} = \bar{v}_{n+1}/ \| \bar{v}_{n+1} \|$
            $\epsilon_{n+1} = 1/ \| \bar{v}_{n+1} \|$; $\delta_{n+1} = -(v_{n+1}^T \hat{v}_n) * \epsilon_{n+1} * \epsilon_n$
            $\eta = -\epsilon_{n+1} * v_{n+1}^T v_{n-1} - \epsilon_{n+1} * (v_{n+1}^T \hat{v}_n) * \delta_n$
            $\mu_1 = \gamma_{n+1}/\epsilon_{n+1}$; $\mu_2 = (\alpha_n - \delta_{n+1} * \mu_1)/\epsilon_n$
            $\mu_3 = \beta_n - \delta_n * \mu_2 - \eta * \mu_1$; $\mu_4 = \mu_5 = 0$
    (c) else if mod(n,3) = 0
        $\mu_1 = \gamma_{n+1}$; $\mu_2 = \alpha_n/\epsilon_n$; $\mu_3 = (\beta_n - \delta_n * \mu_2)/\epsilon_{n-1}$
        $\mu_4 = -\delta_{n-1} * \mu_3 - \eta * \mu_2$; $\mu_5 = 0$; $\hat{v}_{n+1} = v_{n+1}$
(4) Perform quasi minimization
    (a) $\nu_5 = s_{n-3} * \mu_5$; $t_4 = -c_{n-3} * \mu_5$
    (b) $\nu_4 = -c_{n-2} * t_4 + s_{n-2} * \mu_4$; $t_3 = -s_{n-2} * t_4 - c_{n-2} * \mu_4$
    (c) $\nu_3 = -c_{n-1} * t_3 + s_{n-1} * \mu_3$; $t_2 = -s_{n-1} * t_3 - c_{n-1} * \mu_3$
    (c) $\nu_2 = -c_n * t_2 + s_n * \mu_2$; $t_1 = -s_n * t_2 - c_n * \mu_2$
    (d) if ($|\mu_1| > |t_1|$) $t_a = -t_1/\mu_1$; $s_{n+1} = 1/\sqrt{1+t_a^2}$; $c_{n+1} = c_{n+1} * t_a$
    (e) else $t_a = -\mu_1/t_1$; $c_{n+1} = 1/\sqrt{1+t_a^2}$; $s_{n+1} = c_{n+1} * t_a$
    (f) $\nu_1 = -c_{n+1} * t_1 + s_{n+1} * \mu_1$; $\tau = -c_{n+1}\tilde{\tau}$; $\tilde{\tau} = -s_{n+1}\tilde{\tau}$
    (g) $y_n = (v_n - \nu_2 y_{n-1} - \nu_3 y_{n-2} - \nu_4 y_{n-3} - \nu_5 y_{n-4})/\nu_1$
(5) Update solution and residual
    (a) $x_n = x_{n-1} + \tau y_n$
    (b) $r_n^{BQMR(3)} = s_{n+1}^2 r_{n-1}^{BQMR(3)} - c_{n+1}\tilde{\tau}\hat{v}_{n+1}$
(6) **End for**
```

3.2. Complexity and storage analysis. Table 3.1 shows the operation counts required for the BQMR(1), BQMR(2), and BQMR(3) algorithms. Since the operation counts for the BQMR(2) algorithm are different between odd and even iterations, only their average is listed. The same is true for BQMR(3). BQMR(2) and BQMR(3) are only slightly more expensive than BQMR(1). When large block size k is used, the operation count for BQMR(k) per iteration grows with the square of the block size.

TABLE 3.1
Complexity and storage statistics per iteration.

Method	(1)	(2)	(3)	(4)	Total op. count	Storage
BQMR(1)	2	4	7	6	28N+2 (MVP + P)	12N+A
BQMR(2)	2	5	8	6	32N+2 (MVP + P)	13N+A
BQMR(3)	2	6	9	6	36N+2 (MVP + P)	14N+A

(1) Number of matrix vector multiplications
(2) Number of inner product calculations
(3) Number of daxpy operations
(4) Number of other (e.g., vector add) operations
P = preconditioning
MVP = matrix vector products.

4. Numerical experiments with BQMR(k) algorithms. The numerical examples used in this experiment range from a two-dimensional convection diffusion equation to the problems in the Harwell–Boeing collections [2]. cde31 and cde63 are the convection-diffusion equations

$$-\Delta u + \gamma\left(x\frac{\partial u}{\partial x} + y\frac{\partial u}{\partial y}\right) + \beta u = f, \tag{4.1}$$

where $\gamma = 50$ and 100, $\beta = -25$ and -100 for $n = 31$ (i.e., 31×31 grid) and $n = 63$, respectively. The rest of the problems were obtained from the Harwell–Boeing collection.

Table 4.1 shows the iteration counts for the different algorithms without preconditioning and with right ILUT(0) preconditioning. (This ILUT(0) was originally proposed by Saad [13]. The version used in this experiment, however, was borrowed from the implementation by Freund and Nachtigal.) The result for GMRES (nonrestarted version) is also included for comparison purposes since, when breakdown does not occur, GMRES is mathematically equivalent to BQMR(∞). The stopping criterion used here is $\| r_k \| / \| r_0 \| < 10^{-8}$. Moreover, $x_0 = 0$ and $\tilde{r}_0 = r_0$.

TABLE 4.1
Iteration counts for different problems.

Problem	BQMR(1)		BQMR(2)		BQMR(3)		GMRES
	(U)	(P)	(U)	(P)	(U)	(P)	(P)
cde31	101	26	101	26	91	26	24
cde63	259	45	259	43	259	43	39
orsreg1	323	29	323	27	318	26	24
orsirr1	1026	21	1020	21	1016	20	19
orsirr2	721	22	720	22	718	21	19
sherman1	437	33	437	32	437	32	28
sherman3	(a)	82	(a)	82	(a)	82	76
sherman4	134	35	134	35	134	34	33
sherman5	(a)	24	(a)	23	(a)	23	22
saylr1	(a)	14	(a)	14	(a)	14	11
saylr3	435	33	435	32	435	32	28
saylr4	(a)	50	(a)	50	(a)	49	44

(U) No preconditioning
(P) ILUT(0) preconditioning used
(a) > 2000 iterations

From Table 4.1, it can also be observed that the number of iterations needed for the BQMR(k) algorithms in most cases is not much more than the GMRES algorithm. The number of iterations for BQMR(k) decreases typically when k increases from one to three. However, Table 4.1 does not reflect the total cost for each method on a given problem, since the different methods have unequal cost per iteration.

If the total operation count is the sole criterion for determining efficiency, then BQMR(1) is more efficient than BQMR(2) and BQMR(3) in most problems considered here. However, for some problems such as orsreg1, BQMR(2) with preconditioning is more efficient. An analysis is given as follows.

- orsreg1 has $N = 2205$ and number of nonzeros $NNZ = 14133$.
- A matrix vector multiply takes $NNZ/N \times 2 \approx 12.8N$ operations.
- There are three matrix vector multiply and two ILUT(0) preconditioning per iteration => cost = $12.8N \times 5 = 64N$ operations.
- Total operation count per iteration for BQMR(1) = $64N + 28N = 92N$.
- Total operation count per iteration for BQMR(2) = $64N + 32N = 96N$.
- Total operation count for BQMR(1) = $92N * 29 = 2668N$ operations.

- Total operation count for BQMR(2) = $96N * 27 = 2592N$ operations.

Even though BQMR(1) seems to be more efficient in most problems, there are other factors that can affect performance. For example, it is well known that matrix vector multiplication and preconditioning are much more expensive than vector operations, especially on some parallel computers and on supercomputers without efficient gather/scatter hardware support. Thus, on these advanced computers, BQMR(2) and BQMR(3) are potentially more efficient, in addition to giving smoother convergence curves than BQMR(1).

The convergence history of a convection diffusion equation cde63 is also plotted as shown in Fig. 1. The plots basically show that BQMR(1) does help to smooth out the irregular convergence behavior of BCG; and BQMR(2) and BQMR(3) most of the time give an even smoother convergence behavior.

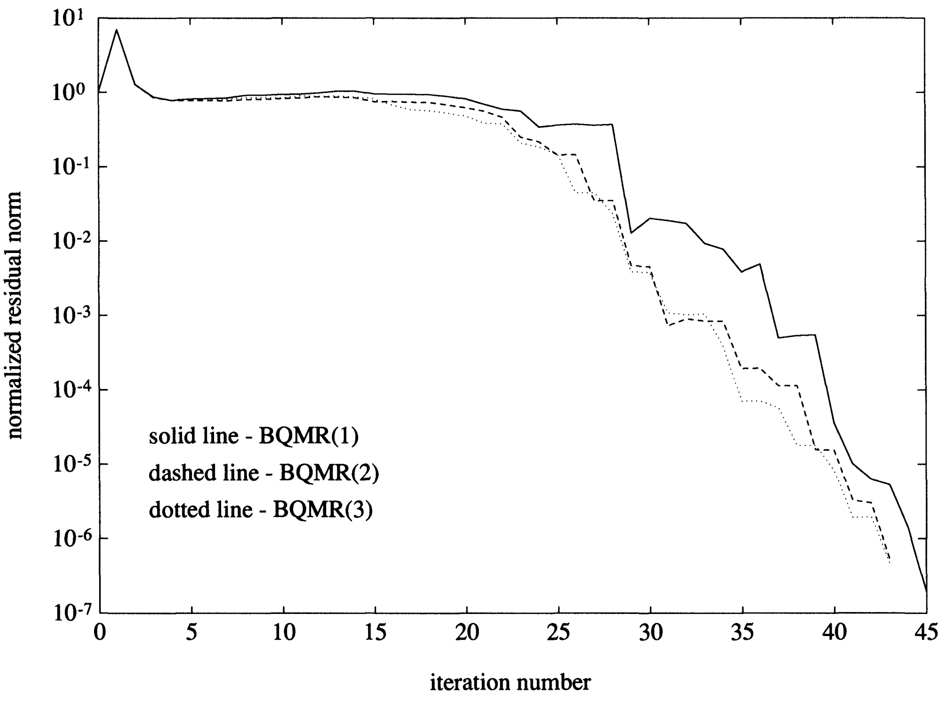

FIG. 1. *Convergence history for the* cde63 *problem* (ILUT(0) *preconditioned*).

5. Transpose-free quasi-minimal residual algorithms.

5.1. A transpose-free QMR algorithm based on BiCGSTAB. In many engineering applications the matrix vector product $A^T y$ is not easily computable. This, together with other efficiency issues, has motivated the transpose-free Lanczos-based algorithms. The first of these is CGS [16] that often gives extremely irregular convergence curves. Since Freund and Nachtigal [9] have shown that quasi minimization does help to give a smoother convergence behavior for BCG, quasi minimization has also been applied to transpose-free methods giving rise to the several transpose-free QMR methods such as Freund's transpose-free QMR method [7], the transpose-free implementation of the simplified QMR method of Chan, De Pillis, and Van der Vorst [3], and the transpose-free QMR method based on the BiCGSTAB algorithm of

Chan et al. [4] (QMRCGSTABx). Since this paper proposes variants of the QMRCGSTAB1 method, only this method will be described in detail.

QMRCGSTAB1 was developed in the same spirit as Freund developed his transpose-free QMR method from CGS. QMRCGSTAB1 was derived based on the matrix representation of Van der Vorst's BiCGSTAB algorithm [18]. In particular, the vectors generated by the BiCGSTAB process can be put into the following matrix form

$$AY_m = W_{m+1}L^e_m, \tag{5.1}$$

where $Y_m = [p_1\, s_1\, p_2\, s_2 \cdots]$, $W_m = [r_0\, s_1\, r_1\, s_2 \cdots]$, and p_i's, r_i's, and s_i's are defined in the BiCGSTAB algorithm [18]. Moreover, L^e_m is the product of an $(m+1)\times m$ bidiagonal matrix (which has unity diagonal and -1's on the subdiagonal) with an $m \times m$ diagonal matrix ($=\operatorname{diag}[\alpha_1^{-1}\omega_1^{-1}\alpha_2^{-1}\omega_1^{-1}\cdots]$). The solution can be written in the form

$$x_m = x_0 + Y_m z \quad \text{for some } z \in \Re^m, \tag{5.2}$$

and the corresponding residual is

$$\begin{aligned} r_m &= r_0 - AY_m z \\ &= W_{m+1}(e_1^{(m+1)} - L^e_m z). \\ &= W_{m+1}\Omega^{-1}_{m+1}[\Omega_{m+1}(e_1^{(m+1)} - L^e_m z)]. \end{aligned} \tag{5.3}$$

The idea of QMRCGSTAB1 is to minimize the expression $\Omega_{m+1}(e_1^{(m+1)} - L^e_m z)$. Having the knowledge of the L^e_m matrix, quasi minimization can be carried out the same way it was done in deriving Freund's transpose-free QMR algorithm [7].

5.2. A family of transpose-free quasi-minimal residual algorithms. The generalization of the quasi-minimization step described in §2 can also be applied to the transpose-free Lanczos-based algorithms. In particular, we choose to start with the QMRCGSTAB1 algorithm [4]. (We can also start with the CGS algorithm giving rise to variants of Freund's transpose-free QMR algorithm.) In QMRCGSTAB1, Ω_{m+1} is a diagonal matrix. In general, Ω_{m+1} does not have to be a diagonal matrix. Again, if we let $\Omega^{-1}_{m+1} = B^{(k)}_{m+1}$ to be upper triangular such that $W_{m+1}B^{(k)}_{m+1}$ is orthonormal, we have a minimal residual algorithm. However, this is too expensive to compute. The strategy we propose in this paper is to determine a relatively sparse $B^{(k)}_{m+1}$ such that $W_{m+1}B^{(k)}_{m+1}$ is close to orthonormal. For example, we can orthonormalize s_1 against r_0. (These vectors are defined in [18], s_2 against r_1, and so on, so that $W_{m+1}B^{(k)}_{m+1}$ is pairwise orthonormal (having $\lceil (m+1)/2 \rceil$ subblocks where each subblock is orthonormal).) To accomplish this, $B^{(k)}_{m+1}$ is constructed to be block diagonal with block size 2 (each block is 2×2 upper triangular). We call this QMRCGSTAB(2). (To be consistent in naming the methods, QMRCGSTAB1 will subsequently be called QMRCGSTAB(1). We can further generalize this approach by generating orthonormal groups in W_{m+1} of size k. Again this can be accomplished by using a sparse $B^{(k)}_{m+1}$ weight matrix to be block diagonal with block size k. In this paper we also implement the case when $k = 4$ and we call this QMRCGSTAB(4).

6. Implementation and complexity of the transpose-free QMR algorithms.

6.1. Implementation details. Considering the QMRCGSTAB(k) algorithms, solving the least squares problem basically involves three steps.

Step 1. Update B^k_{m+1}.

Step 2. Update the relatively sparse matrix $\tilde{L}^e_m = (B^{(k)}_{m+1})^{-1}L^e_m$.

Step 3. Perform QR decomposition on $\tilde{L}^e_m$ to get R_m, and apply the same Q_m to $\rho_0 e_1^{(m+1)}$ to get $t_m = [\tau_1 \cdots \tau_m]^T$.
The solution can then be updated by

$$x_m = x_{m-1} + Y_m R_m^{-1}[0 \cdots 0\ \tau_m]^T. \tag{6.1}$$

The details of the QMRCGSTAB(k) algorithms for $k = 2, 4$ are described in the following sections.

6.1.1. Implementation of QMRCGSTAB(2) algorithms. The pseudocode for the QMRCGSTAB(2) algorithm is as follows.

```
ALGORITHM QMRCGSTAB(2)
(1) Initialization
    (a) $r_0 = b - Ax_0$; $\tilde{r}_1$ arbitrary, $\tilde{r}_0 r_0 \neq 0$
    (b) $p_0 = v_0 = d_0 = 0$; $\rho_0 = \alpha_0 = \omega_0 = 1$; $\tilde{\tau} = \tau = \| r_0 \|$
    (c) $s_{-1} = s_0 = 0$; $c_{-1} = c_0 = -1$; $\epsilon_{11} = 1/ \| r_0 \|$; $w_0 = \epsilon_{11} r_0$
(2) For $n = 1, 2, \ldots$ do
    (a) $\rho_n = \tilde{r}_0^T r_{n-1}$; $\beta_n = (\rho_n \alpha_{n-1})/(\rho_{n-1}\omega_{n-1})$
    (b) $p_n = r_{n-1} + \beta_n(p_{n-1} - \omega_{n-1} v_{n-1})$
    (c) $v_n = Ap_n$; $\alpha_n = \rho_n/(\tilde{r}_0^T v_n)$; $y_n = r_n - \alpha_n v_n$
(3) U-solve to get the modified B matrix
    $m = 2n - 1$; $\tilde{w}_m = y_n - (y_n^T w_{m-1})w_{m-1}$; $\epsilon_{22} = 1/ \| \tilde{w}_m \|$; $w_m = \epsilon_{22}\tilde{w}_m$
    $\epsilon_{12} = -(y_n^T w_{m-1})\epsilon_{22}\epsilon_{11}$; $\mu_1 = -1/(\alpha_n \epsilon_{22})$; $\mu_2 = 1/[(\alpha_n - \epsilon_{12}\mu_1)\epsilon_{11}]$
    $\nu_2 = s_{m-1}\mu_2$; $\delta_1 = -c_{m-1}\mu_2$
    $\tilde{d}_m = p_n - \nu_2 d_{m-1}$
(4) Continue quasi minimization
    if $(|\mu_1| > |\delta_1|)$ $\xi = -\delta_1/\mu_1$; $s_m = 1/\sqrt{1+\xi^2}$; $c_m = s_m\xi$
    else $\xi = -\mu_1/\delta_1$; $c_m = 1/\sqrt{1+\xi^2}$; $s_m = c_m\xi$
    $\nu_1 = -c_m\delta_1 + s_m\mu_1$; $\tau = -c_m\tilde{\tau}$; $\tilde{\tau} = -s_m\tilde{\tau}$; $d_m = \tilde{d}_m/\nu_1$
    $x_m = x_{m-1} + \tau d_m$
(5) Continue with BiCGSTAB
    $t_n = Ay_n$
    $\omega_n = (y_n^T t_n)/(t_n^T t_n)$
    $r_n = y_n - \omega_n t_n$
(6) second U-solve to get the modified B matrix
    $m = 2n$; $\epsilon_{33} = 1/ \| r_n \|$; $w_m = \epsilon_{33} r_n$
    $\mu_1 = -1/(\omega_n\epsilon_{33})$; $\mu_2 = 1/(\omega_n\epsilon_{22})$
    $\mu_3 = -(\epsilon_{12}\mu_2)/\epsilon_{11}$; $\nu_3 = s_{m-2}\mu_3$; $\delta_2 = -c_{m-2}\mu_3$; $\nu_2 = -c_{m-1}\delta_2 + s_{m-1}\mu_2$
    $\epsilon_{11} = \epsilon_{33}$; $\delta_1 = -s_{m-1}\delta_2 - c_{m-1}\mu_2$, $\tilde{d}_m = y_n - \nu_3 d_{m-2} - \nu_2 d_{m-1}$
(7) Continue quasi minimization
    if $(|\mu_1| > |\delta_1|)$ $\xi = -\delta_1/\mu_1$; $s_m = 1/\sqrt{1+\xi^2}$; $c_m = s_m\xi$
    else $\xi = -\mu_1/\delta_1$; $c_m = 1/\sqrt{1+\xi^2}$; $s_m = c_m\xi$
    $\nu_1 = -c_m\delta_1 + s_m\mu_1$; $\tau = -c_m\tilde{\tau}$; $\tilde{\tau} = -s_m\tilde{\tau}$; $d_m = \tilde{d}_m/\nu_1$
    $x_m = x_{m-1} + \tau d_m$
(8) If $x_m$ is accurate enough, stop
(9) End for
```

6.1.2. Implementation of QMRCGSTAB(4) algorithm. The pseudocode for the QMRCGSTAB(4) algorithm is as follows.

```
ALGORITHM QMRCGSTAB(4)
(1) Initialization
    (a) $r_0 = b - Ax_0$; $\tilde{r}_1$ arbitrary, $\tilde{r}_0 r_0 \neq 0$
    (b) $p_0 = v_0 = d_0 = 0$; $\rho_0 = \alpha_0 = \omega_0 = 1$; $\tilde{\tau} = \tau = \| r_0 \|$
    (c) $s_{-3} = s_{-2} = s_{-1} = s_0 = 0$; $c_{-3} = c_{-2} = c_{-1} = c_0 = -1$
    (d) $\epsilon_{11} = 1/ \| r_0 \|$; $w_0 = \epsilon_{11} r_0$
(2) For $n = 1, 2, \ldots$ do
    same as step (2a)–(2c) in Algorithm QMRCGSTAB(2)
(3) U-solve to get the modified B matrix
    (a) if mod(n,2) = 1 $(m = 2n - 1)$
        $\tilde{w}_m = y_n - (y_n^T w_{m-1}) w_{m-1}$; $\epsilon_{22} = 1/ \| \tilde{w}_m \|$; $w_m = \epsilon_{22} \tilde{w}_m$
        $\epsilon_{12} = -(y_n^T w_{m-1}) \epsilon_{22} \epsilon_{11}$; $\mu_1 = -1/(\alpha_n \epsilon_{22})$; $\mu_2 = 1/[(\alpha_n - \epsilon_{12}\mu_1)\epsilon_{11}]$
        $\nu_2 = s_{m-1}\mu_2$; $\delta_1 = -c_{m-1}\mu_2$
        $\tilde{d}_m = p_n - \nu_2 d_{m-1}$
    (b) else if mod(n,2) = 0
        $\tilde{w}_m = y_n - (y_n^T w_{m-1}) w_{m-1} - (y_n^T w_{m-2}) w_{m-2} - (y_n^T w_{m-3}) w_{m-3}$
        $\epsilon_{44} = 1/ \| \tilde{w}_m \|$; $w_m = \epsilon_{44} \tilde{w}_m$
        $\epsilon_{14} = -\epsilon_{44}[(y_n^T w_{m-1})\epsilon_{11} + (y_n^T w_{m-2})\epsilon_{12} + (y_n^T w_{m-3})\epsilon_{13}]$
        $\epsilon_{24} = -\epsilon_{44}[(y_n^T w_{m-2})\epsilon_{22} + (y_n^T w_{m-3})\epsilon_{33}]$
        $\epsilon_{34} = -\epsilon_{44}(y_n^T w_{m-3})\epsilon_{33}$; $\mu_1 = -1/(\alpha_n \epsilon_{44})$; $\mu_2 = (1/\alpha_n - \epsilon_{34}\mu_1)/\epsilon_{33}$
        $\mu_3 = -(\epsilon_{24}\mu_1 + \epsilon_{23}\mu_2)/\epsilon_{22}$; $\mu_4 = -(\epsilon_{14}\mu_1 + \epsilon_{13}\mu_2 + \epsilon_{12}\mu_3)/\epsilon_{11}$
        $\nu_4 = -c_{m-3}\mu_4$; $\delta_3 = -c_{m-3}\mu_4$; $\nu_3 = -c_{m-2}\delta_3 + s_{m-2}\mu_3$
        $\delta_2 = -s_{m-2}\delta_3 - c_{m-2}\mu_3$
        $\nu_2 = -c_{m-1}\delta_2 + s_{m-1}\mu_2$; $\delta_1 = -s_{m-1}\delta_2 - c_{m-1}\mu_2$
        $\tilde{d}_m = p_n - \nu_4 d_{m-3} - \nu_3 d_{m-2} - \nu_2 d_{m-1}$
(4) Continue quasi minimization–same as step (4) in QMRCGSTAB(2)
(5) Continue with BiCGSTAB–same as step (5) in QMRCGSTAB(2)
(6) second U-solve to get the modified B matrix
    (a) if mod(n,2) = 1 $(m = 2n - 1)$
        $\tilde{w}_m = r_n - (r_n^T w_{m-1}) w_{m-1} - (r_n^T w_{m-2}) w_{m-2}$
        $\epsilon_{33} = 1/ \| \tilde{w}_m \|$; $w_m = \epsilon_{33} \tilde{w}_m$
        $\epsilon_{13} = -\epsilon_{33}[(r_n^T w_{m-1})\epsilon_{11} + (r_n^T w_{m-2})\epsilon_{12}]$; $\epsilon_{23} = -(r_n^T w_{m-2})\epsilon_{22}\epsilon_{33}$
        $\mu_1 = -1/(\omega_n \epsilon_{33})$; $\mu_2 = (1/\omega_n - \epsilon_{23}\mu_1)/\epsilon_{22}$
        $\mu_3 = -(\epsilon_{13}\mu_1 + \epsilon_{12}\mu_2)/\epsilon_{11}$; $\nu_3 = s_{m-2}\mu_3$; $\delta_2 = -c_{m-2}\mu_3$
        $\nu_2 = -c_{m-1}\delta_2 + s_{m-1}\mu_2$; $\delta_1 = -s_{m-1}\delta_2 - c_{m-1}\mu_2$
        $\tilde{d}_m = y_n - \nu_3 d_{m-2} - \nu_2 d_{m-1}$
    (b) else if mod(n,2) = 0 $(m = 2n)$
        $\epsilon_{55} = 1/ \| r_n \|$; $w_m = \epsilon_{55} r_n$; $\mu_1 = -1/(\omega_n \epsilon_{55})$; $\mu_2 = 1/(\omega_n \epsilon_{44})$
        $\mu_3 = -(\epsilon_{34}\mu_2)/\epsilon_{33}$; $\mu_4 = -(\epsilon_{24}\mu_2 + \epsilon_{23}\mu_3)/\epsilon_{22}$
        $\mu_5 = -(\epsilon_{14}\mu_2 + \epsilon_{13}\mu_3 + \epsilon_{12}\mu_4)/\epsilon_{11}$; $\epsilon_{11} = \epsilon_{55}$
        $\nu_5 = -c_{m-4}\mu_5$; $\delta_4 = -c_{m-4}\mu_5$
        $\nu_4 = -c_{m-3}\delta_4 + s_{m-3}\mu_4$; $\delta_3 = -s_{m-3}\delta_4 - c_{m-3}\mu_4$
        $\nu_3 = -c_{m-2}\delta_3 + s_{m-2}\mu_3$; $\delta_2 = -s_{m-2}\delta_3 - c_{m-2}\mu_3$
        $\nu_2 = -c_{m-1}\delta_2 + s_{m-1}\mu_2$; $\delta_1 = -s_{m-1}\delta_2 - c_{m-1}\mu_2$
        $\tilde{d}_m = y_n - \nu_5 d_{m-4} - \nu_4 d_{m-3} - \nu_3 d_{m-2} - \nu_2 d_{m-1}$
(7) Continue quasi minimization
    if $(|\mu_1| > |\delta_1|)$ $\xi = -\delta_1/\mu_1$; $s_m = 1/\sqrt{1+\xi^2}$; $c_m = s_m \xi$
    else $\xi = -\mu_1/\delta_1$; $c_m = 1/\sqrt{1+\xi^2}$; $s_m = c_m \xi$
```

```
        $\nu_1 = -c_m\delta_1 + s_m\mu_1$; $\tau = -c_m\tilde{\tau}$; $\tilde{\tau} = -s_m\tilde{\tau}$; $d_m = \tilde{d}_m/\nu_1$
        $x_m = x_{m-1} + \tau d_m$
    (8) If $x_m$ is accurate enough, stop
    (9) **End for**
```

6.2. Complexity and storage analysis. Table 6.1 shows the operation counts for the QMRCGSTAB(1), QMRCGSTAB(2), and QMRCGSTAB(4) algorithms. Since the operation counts for the QMRCGSTAB(2) algorithm are different between odd and even iterations, only their average is listed. The same is true for QMRCGSTAB(4).

TABLE 6.1
Complexity and storage statistics per iteration.

Method	(1)	(2)	(3)	(4)	Total op. count	Storage
QMRCGSTAB(1)	2	6	8	0	$28N$+2 (MVP + P)	$8N$+A
QMRCGSTAB(2)	2	7	10	4	$38N$+2 (MVP + P)	$9N$+A
QMRCGSTAB(4)	2	9	14	4	$50N$+2 (MVP + P)	$11N$+A

(1) Number of matrix vector multiplications
(2) Number of inner product calculations
(3) Number of daxpy operations
(4) Number of other (e.g., vector add) operations
A = storage for the A matrix
P = preconditioning
MVP = matrix vector products.

7. Numerical experiments for the transpose-free QMR algorithms. The set of numerical examples used here is the same as in §4. Table 7.1 shows the iteration counts for the different algorithms. The initial solution x_0 is a random vector, $\tilde{r}_0 = r_0$, and the stopping criterion is $\| r_k \| / \| b \| < 10^{-8}$.

It can be observed from Table 7.1 that the number of iterations needed for the QMRCGSTAB (k) algorithms in most cases is about half of those of GMRES when preconditioning is used. Again, the number of iterations for QMRCGSTAB(k) is in most cases slightly lower when larger k is used. Again, due to unequal costs per iteration between the methods, iteration counts alone do not reflect the actual efficiency, which is affected by many other factors such as total operation counts and the target computer architecture on computers where matrix vector multiplication is much more expensive than vector operations (as on some supercomputers where efficient gather/scatter hardware is not available). Hence again, in addition to giving smoother convergence curves, the QMRCGSTAB(k) algorithms are potentially more efficient.

In addition to the iteration counts, the convergence history for a convection diffusion equation cde63 is also plotted in Fig. 2.

8. Concluding remarks. In this paper, we explored the use of different weight matrices in the quasi-minimal residual algorithms, and we derived several QMR variants such as BQMR(2) and BQMR(3) algorithms and transpose-free QMR variants such as QMR-CGSTAB(2) and QMRCGSTAB(4) algorithms. These variants were shown to represent a few of the many possibilities lying in the spectrum of quasi-minimal residual (and transpose-free

TABLE 7.1
Iteration counts for different problems.

Problem	A		B		C		GMRES
	(U)	(P)	(U)	(P)	(U)	(P)	(P)
cde31	65	16	65	16	64	16	24
cde63	119	26	119	26	118	25	39
orsreg1	308	16	305	15	303	15	24
osrirr1	1437	12	1383	12	1358	11	19
orsirr2	882	12	824	12	804	12	19
sherman1	400	18	369	18	362	18	28
sherman3	(a)	65	(a)	65	(a)	64	76
sherman4	110	25	106	23	105	23	33
sherman5	1848	15	1843	14	1812	14	22
saylr1	(a)	9	(a)	9	(a)	8	11
saylr3	345	18	353	18	348	18	28
saylr4	(a)	33	(a)	32	(a)	31	44

(A) QMRCGSTAB(1)
(B) QMRCGSTAB(2)
(C) QMRCGSTAB(4)
(U) No preconditioning
(P) ILUT(0) preconditioning used
(a) > 2000 iterations

QMR) algorithms where on the two ends are nonlook-ahead QMR and a GMRES-like minimal residual algorithm, respectively. We showed that these variants are only slightly more expensive than BQMR(1) and QMRCGSTAB(1) per iteration but they generally give smoother convergence curves. Moreover, in general these variants also give a slightly faster convergence rate. Thus these variants can potentially be more efficient for the following situations:

- when the reduction in iteration count is such that the total operation count is lower (e.g., orsreg1 for BQMR(2));
- when the percentage of nonzeros in the problem matrix is relatively high so that matrix vector multiply is relatively more expensive than the overhead for doing quasi minimization;
- when more powerful preconditioners are used (e.g., ILUT(k) for large k) so that the overhead for doing quasi minimization is relatively inexpensive;
- on certain computer platform (e.g., on some parallel computers or supercomputers with no gather/scatter hardware) where matrix vector multiply is much more expensive than daxpy and dot products.

Acknowledgments. The author would like to thank Roland Freund and Noel Nachtigal for their permission to use their ILUT subroutine and some of their support subroutines. Their comments on this work are also well taken. Most matrix and vector operations are performed through the use of the subroutines available in SPARSKIT. The author would also like to thank

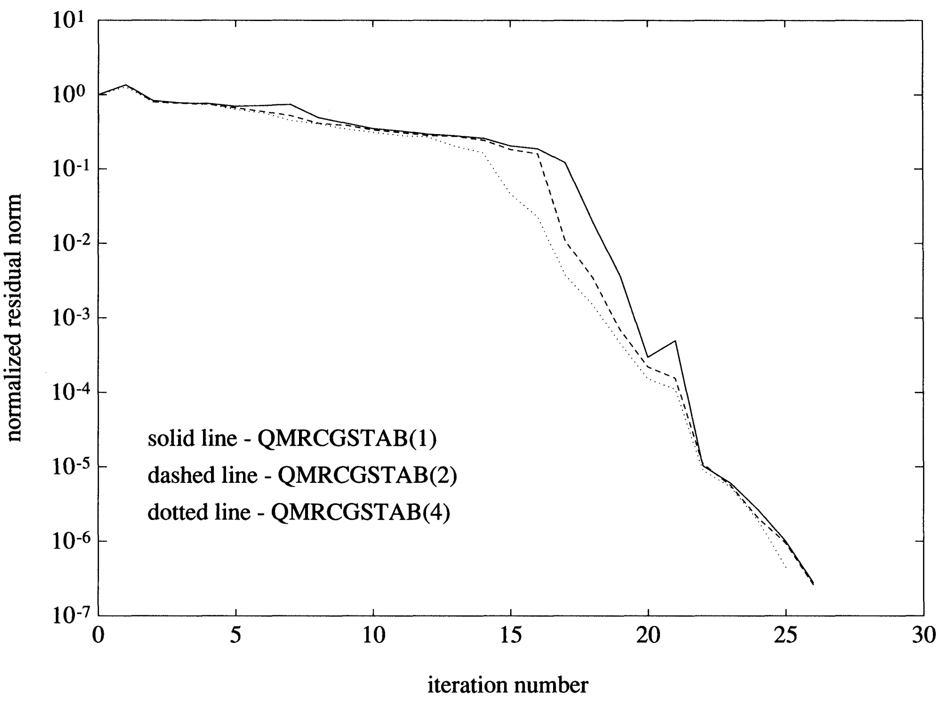

FIG. 2. *Convergence history for the* cde63 *problem* (ILUT(0) *preconditioned*).

Tony Chan for inspiring this work and Raymond Tuminaro for many helpful discussions. Finally, the constructive comments from Martin Gutknecht are greatly appreciated.

REFERENCES

[1] A. T. CHRONOPOULOS AND S. MA, *On Squaring Krylov Subspace Iterative Methods for Nonsymmetric Linear Systems*, Tech. Report 89-67, Computer Science Department, Univ. of Minnesota at Minneapolis, 1989.

[2] I. S. DUFF, R. G. GRIMES, AND J. G. LEWIS, *Sparse matrix test problems*, ACM Trans. Math. Software, 15 (1989), pp. 1–14.

[3] T. F. CHAN, L. DE PILLIS, AND H. VAN DER VORST, *A Transpose-Free Squared Lanczos Algorithm and Application to Solving Nonsymmetric Linear Systems*, CAM Report 91-17, Dept. Mathematics, UCLA, Oct. 1991.

[4] T. F. CHAN, E. GALLOPOULOS, V. SIMONCINI, T. SZETO, AND C. H. TONG, *QMRCGSTAB: A Quasi-minimal Residual Variant of the BiCGSTAB Algorithm for Nonsymmetric Systems*, Center for Supercomputing Research and Development Report 1231, University of Illinois at Urbana-Champaign, May 1992.

[5] V. FABER AND T. MANTEUFFEL, *Necessary and sufficient conditions for the existence of a conjugate gradient method*, SIAM J. Numer. Anal., 21 (1984), pp. 352–362.

[6] R. FLETCHER, *Conjugate Gradient Methods for Indefinite Systems*, in Proc. Dundee Conference on Numerical Analysis, 1975, Lecture Notes in Mathematics 506, G. A. Watson, ed., Springer-Verlag, Berlin, 1976, pp. 73–89.

[7] R. W. FREUND, *A Transpose-Free Quasi-Minimal Residual Algorithm for Non-Hermitian Linear Systems*, Research Institute for Advanced Computer Science Tech. Report 91.18, NASA Ames Research Center, Moffett Field, CA, Sept. 1991.

[8] R. W. FREUND, M. H. GUTKNECHT, AND N. M. NACHTIGAL, *An Implementation of the Look-ahead Lanczos Algorithm for Non-Hermitian Matrices*, Tech. Report 91-09, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffett Field, CA, 1991.

[9] R. W. FREUND AND N. M. NACHTIGAL, *QMR: a quasi-minimal residual methodfor non-Hermitian linear systems*, Numer. Math., 60 (1991), pp. 315–339.

[10] M. R. HESTENES AND E. STIEFEL, *Methods of conjugate gradients for solving linear systems*, J. Res. Natl. Bur. Stand., 49 (1952), pp. 409–436.

[11] C. LANCZOS, *Solution of systems of linear equations by minimized iterations*, J. Res. Natl. Bur. Stand., 49 (1952), pp. 33–53.

[12] Y. SAAD, *The Lanczos biorthogonalization algorithm and other oblique projection methods for solving large unsymmetric systems*, SIAM J. Numer. Anal., 19 (1982), pp. 485–506.

[13] Y. SAAD, *Preconditioning techniques for nonsymmetric and indefinite linear systems*, J. Comput. Appl. Math., 24 (1988), pp. 89–105.

[14] Y. SAAD AND M. H. SCHULTZ, *Conjugate gradient-like algorithms for solving nonsymmetric linear systems*, Math. Comput., 44 (1985), pp. 417–424.

[15] Y. SAAD AND M. H. SCHULTZ, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869.

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

[17] C. H. TONG, *A Comparative Study of Preconditioned Lanczos Methods for Nonsymmetric Linear Systems*, Sandia Report, SAND91-8240B Sandia National Laboratries, Livermore, CA, 1992.

[18] H. A. VAN DER VORST, *Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems*, Preprint, Univ. of Utrecht, the Netherlands, Sept. 1990.