SIAM J. SCI. COMPUT.
Vol. 14, No. 2, pp. 470–482, March 1993

A TRANSPOSE-FREE QUASI-MINIMAL RESIDUAL ALGORITHM FOR NON-HERMITIAN LINEAR SYSTEMS*

ROLAND W. FREUND†

Abstract. The biconjugate gradient method (BCG) for solving general non-Hermitian linear systems $Ax = b$ and its transpose-free variant, the conjugate gradients squared algorithm (CGS), both typically exhibit a rather irregular convergence behavior with wild oscillations in the residual norm. Recently, Freund and Nachtigal proposed a BCG-like approach, the quasi-minimal residual method (QMR), that remedies this problem for BCG and produces smooth convergence curves. However, like BCG, QMR requires matrix-vector multiplications with both the coefficient matrix A and its transpose A^T. In this note, it is demonstrated that the quasi-minimal residual approach can also be used to obtain a smoothly convergent CGS-like algorithm that does not involve matrix-vector multiplications with A^T. It is shown that the resulting transpose-free QMR method (TFQMR) can be implemented very easily by changing only a few lines in the standard CGS algorithm. Finally, numerical experiments are reported.

Key words. non-Hermitian linear systems, biconjugate gradients, transpose-free, conjugate gradients squared, quasi-minimal residual property

AMS(MOS) subject classification. 65F10

1. Introduction.

The classical conjugate gradient method (CG) [11] is one of the most powerful iterative schemes for solving large sparse linear systems

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

with Hermitian positive definite coefficient matrices A. The biconjugate gradient algorithm (BCG) [13], [3] is the "natural" extension of CG to linear systems (1.1) with general non-Hermitian nonsingular coefficient matrices. However, unlike CG, the BCG iterates are not characterized by a minimization property, which means that the algorithm can exhibit—and typically does—a rather irregular convergence behavior with wild oscillations in the residual norm. Furthermore, in the BCG algorithm, even breakdowns—more precisely, division by 0—may occur.

Recently, Freund and Nachtigal [7] proposed a BCG-like approach, the quasi-minimal residual method (QMR), that remedies the problems of BCG. The QMR iterates are defined by a quasi minimization of the residual norm, which leads to smooth convergence curves. Furthermore, QMR can be implemented based on a look-ahead version [14], [6] of the nonsymmetric Lanczos algorithm [12], which avoids possible breakdowns of the process, except for the very special situation of an incurable breakdown.

Except for special cases [8], such as complex symmetric matrices [4], BCG and QMR require matrix-vector multiplications with both the coefficient matrix A of (1.1) and its transpose A^T. This is a disadvantage for certain applications, such as linear systems arising in ordinary differential equation solvers [9], where A^T is not readily available. Sonneveld [15] derived a variant of BCG, the conjugate gradients squared algorithm (CGS), that does not involve A^T. However, like BCG, CGS also

* Received by the editors September 9, 1991; accepted for publication (in revised form) April 3, 1992. This research was supported by the Defense Advanced Research Projects Agency via Cooperative Agreement NCC 2-387 between the National Aeronautics and Space Administration and the Universities Space Research Association.

† Research Institute for Advanced Computer Science, Mail Stop T041-5, NASA Ames Research Center, Moffett Field, California 94035. Present address, AT&T Bell Laboratories, Room 2C-420, 600 Mountain Avenue, Murray Hill, New Jersey 07974-0636.

exhibits rather erratic convergence behavior. Van der Vorst [16] proposed the Bi-CGSTAB algorithm, which uses local steepest descent steps to obtain a more smoothly convergent CGS-like process. While Bi-CGSTAB seems to work well in many cases, it still exhibits irregular convergence behavior for some difficult problems. Also, Bi-CGSTAB can converge considerably slower than CGS.

In this note, we demonstrate that the quasi-minimal residual approach can also be used to obtain a smoothly convergent CGS-like algorithm that does not require matrix-vector multiplications with A^T. We show that the resulting transpose-free QMR method (TFQMR) can be implemented very easily, by changing only a few lines in the standard CGS algorithm. We stress that the proposed TFQMR method and the original QMR algorithm [7] are not mathematically equivalent. In particular, the sets of iterates generated by the two schemes are different.

The rest of this paper is organized as follows. In §2, we briefly review the CGS algorithm. In §3, we present the transpose-free quasi-minimal residual approach. To derive an actual implementation, we first present an auxiliary result in §4 and then present the resulting TFQMR algorithm in §5. Numerical examples are reported in §6, and in §7, we make some concluding remarks.

Throughout the paper, all vectors and matrices are allowed to have real or complex entries. As usual, M^T and M^H denote the transpose and conjugate transpose of M, respectively. The vector norm $\|x\| = \sqrt{x^H x}$ is always the Euclidean norm, and $\|M\| = \max_{\|x\|=1} \|Mx\|$ is the corresponding matrix norm. The notation

$$K_n(c, M) := \operatorname{span}\{c, Mc, \ldots, M^{n-1}c\}$$

is used for the nth Krylov subspace of $\mathbb{C}^N$ generated by $c \in \mathbb{C}^N$ and the $N \times N$ matrix M. For $q \in \mathbb{R}$, $\lfloor q \rfloor$ is the largest integer $\leq q$. The set of all complex polynomials of degree at most n is denoted by

$$\mathcal{P}_n := \{\varphi(\lambda) \equiv \gamma_0 + \gamma_1\lambda + \cdots + \gamma_n\lambda^n \mid \gamma_0, \gamma_1, \ldots, \gamma_n \in \mathbb{C}\}.$$

The coefficient matrix A of (1.1) is always assumed to be a nonsingular, in general non-Hermitian, $N \times N$ matrix, and $b \in \mathbb{C}^N$. Generally, $x_n \in \mathbb{C}^N$, $n = 0, 1, \ldots$, denote iterates for (1.1) with corresponding residual vectors $r_n := b - Ax_n$. If necessary, quantities of different algorithms will be distinguished by superscripts, e.g., x_n^{BCG} and x_n^{CGS}.

2. The CGS algorithm.

Let $x_0 \in \mathbb{C}^N$ be any initial guess for (1.1) with residual vector $r_0 = b - Ax_0$, and let $\tilde{r}_0 \in \mathbb{C}^N$ be any vector such that $\tilde{r}_0^H r_0 \neq 0$, e.g., $\tilde{r}_0 = r_0$. The nth iterate, x_n^{BCG}, generated by the BCG process is defined by the Galerkin-type condition

$$w^H(b - Ax_n^{\mathrm{BCG}}) = 0 \quad \text{for all } w \in K_n(\tilde{r}_0, A^H), \qquad x_n^{\mathrm{BCG}} \in x_0 + K_n(r_0, A). \tag{2.1}$$

Clearly, the corresponding residual vector r_n^{BCG} is of the form

$$r_n^{\mathrm{BCG}} = \varphi_n(A)r_0, \quad \text{where} \quad \varphi_n \in \mathcal{P}_n \quad \text{and} \quad \varphi_n(0) = 1. \tag{2.2}$$

Sonneveld [15] observed that the iterates $x_n \in x_0 + K_{2n}(r_0, A)$ whose residual vectors are just the "squares" of (2.2), i.e.,

$$r_n = (\varphi_n(A))^2 r_0, \tag{2.3}$$

can be computed by means of a BCG-like process that does not involve A^H. The resulting process is as follows.

ALGORITHM 2.1 (CGS ALGORITHM [15]).

```
(1) Start:
    (a) Choose $x_0 \in \mathbb{C}^N$;
    (b) Set $p_0 = u_0 = r_0 = b - Ax_0$, $v_0 = Ap_0$;
    (c) Choose $\tilde{r}_0$ such that $\rho_0 = \tilde{r}_0^H r_0 \neq 0$.
(2) For $n = 1, 2, \dots,$ do:
    (a) Set $\sigma_{n-1} = \tilde{r}_0^H v_{n-1}$, $\alpha_{n-1} = \rho_{n-1}/\sigma_{n-1}$;
            $q_n = u_{n-1} - \alpha_{n-1} v_{n-1}$;
    (b) Set $x_n = x_{n-1} + \alpha_{n-1}(u_{n-1} + q_n)$;
            $r_n = r_{n-1} - \alpha_{n-1} A(u_{n-1} + q_n)$;
        If $x_n$ has converged: stop;
    (c) Set $\rho_n = \tilde{r}_0^H r_n$, $\beta_n = \rho_n/\rho_{n-1}$;
            $u_n = r_n + \beta_n q_n$;
            $p_n = u_n + \beta_n(q_n + \beta_n p_{n-1})$;
            $v_n = Ap_n$.
```

In exact arithmetic, Algorithm 2.1 terminates after a finite number, say n_*, of iterations. Usually, $x_{n_*} = A^{-1}b$ is the solution of (1.1). However, for general A, the possibility cannot be excluded that Algorithm 2.1 terminates prematurely with $\sigma_{n_*} = 0$ or $\rho_{n_*} = 0$, before the solution of (1.1) has been found. If this happens, a continuation of the process would lead to division by 0 in step (2a) or (2c) of the next iteration, and therefore premature termination is also referred to as a "breakdown" of the algorithm. Fortunately, breakdowns are very rare in practice and, furthermore, they can be overcome by using look-ahead strategies (cf. §7). Here, for simplicity, we will consider only the standard CGS Algorithm 2.1, without look-ahead. We note that a modified CGS algorithm that avoids exact breakdowns was recently given by Brezinski and Sadok [1].

In the sequel, we always assume that $n \in \{1, 2, \dots, n_*\}$. Note that

$$\alpha_{n-1} \neq 0 \quad \text{for all } n. \tag{2.4}$$

Moreover, we will make use of the relations

$$u_{n-1} = \varphi_{n-1}(A)\psi_{n-1}(A)r_0 \quad \text{and} \quad q_n = \varphi_n(A)\psi_{n-1}(A)r_0, \tag{2.5}$$

which are derived in [15]. Here the φ_n's are given by (2.2), and the polynomials ψ_n can be updated by means of

$$\psi_n(\lambda) \equiv \varphi_n(\lambda) + \beta_n \psi_{n-1}(\lambda), \tag{2.6}$$

where $\psi_0(\lambda) \equiv 1$. Finally, we note that

$$\varphi_n(\lambda) \equiv \varphi_{n-1}(\lambda) - \alpha_{n-1}\lambda\psi_{n-1}(\lambda). \tag{2.7}$$

3. The quasi-minimal residual approach.

By (2.2) and (2.3), the CGS iterates are defined implicitly via the Galerkin condition (2.1), but they do not satisfy a residual norm minimization property. In this section, we demonstrate that, using the same sequence of vectors u_{n-1} and q_n as generated by Algorithm 2.1, one can also define iterates with a quasi-minimization property.

We set

$$y_m = \begin{cases} u_{n-1} & \text{if } m = 2n-1 \text{ is odd,} \\ q_n & \text{if } m = 2n \text{ is even,} \end{cases} \tag{3.1}$$

and

$$w_m = \begin{cases} (\varphi_n(A))^2 r_0 & \text{if } m = 2n+1 \text{ is odd,} \\ \varphi_n(A)\varphi_{n-1}(A) r_0 & \text{if } m = 2n \text{ is even.} \end{cases} \tag{3.2}$$

Note that, by (2.3),

$$w_{2n+1} = r_n^{\text{CGS}}, \qquad n = 1, 2, \ldots, n_*. \tag{3.3}$$

In the sequel, it is always assumed that $m \in \{1, 2, \ldots, 2n_*\}$. Using (2.5) and (2.7), one readily verifies that the vectors (3.1) and (3.2) are related by

$$Ay_m = \frac{1}{\alpha_{\lfloor (m-1)/2 \rfloor}} (w_m - w_{m+1}) \, . \tag{3.4}$$

Note that, by (2.4), the denominator in (3.4) is guaranteed to be different from 0. Setting

$$\begin{aligned} Y_m &= [y_1 \quad y_2 \quad \cdots \quad y_m], \\ W_{m+1} &= [w_1 \quad w_2 \quad \cdots \quad w_m \quad w_{m+1}], \end{aligned}$$

we can rewrite the relations (3.4) in matrix form

$$AY_m = W_{m+1} B_m^{(e)}, \tag{3.5}$$

where

$$B_m^{(e)} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ -1 & 1 & \ddots & \vdots \\ 0 & \ddots & \ddots & 0 \\ \vdots & \ddots & -1 & 1 \\ 0 & \cdots & 0 & -1 \end{bmatrix} \cdot \left(\operatorname{diag}\left(\alpha_0, \alpha_0, \alpha_1, \ldots, \alpha_{\lfloor (m-1)/2 \rfloor}\right)\right)^{-1} \tag{3.6}$$

is an $(m+1) \times m$ lower bidiagonal matrix. Finally, note that in view of (2.6), (2.7), and (2.4), the polynomials φ_n and ψ_n are both of full degree n, and with (2.5) and (3.1), it follows that

$$K_m(r_0, A) = \operatorname{span}\{y_1, y_2, \ldots, y_m\} = \{Y_m z \mid z \in \mathbb{C}^m\}. \tag{3.7}$$

After these preliminaries, we now begin our derivation of the quasi-minimal residual approach. By (3.7), any possible iterate $x_m \in x_0 + K_m(r_0, A)$ can be written in the form

$$x_m = x_0 + Y_m z \quad \text{for some } z \in \mathbb{C}^m. \tag{3.8}$$

By (3.5) and since $w_1 = r_0$ (see (3.3)), the corresponding residual vector satisfies

$$r_m = r_0 - AY_m z = W_{m+1}\left(e_1^{(m+1)} - B_m^{(e)} z\right), \tag{3.9}$$

where $e_1^{(m+1)} = [1 \quad 0 \quad \cdots \quad 0]^T \in \mathbb{R}^{(m+1)}$. Ideally, we would like to choose the free parameter vector z in (3.8) such that the norm $\|r_m\|$ of (3.9) is minimized. However, since, in general, the matrix W_{m+1} is dense and its columns are not orthogonal to each other, this would require the solution of a dense $N \times (m+1)$ least-squares problem. Instead, we minimize only the length of the coefficient vector in the representation (3.9) of r_m. More precisely, let

$$\Omega_{m+1} = \operatorname{diag}(\omega_1, \omega_2, \ldots, \omega_{m+1}), \quad \omega_k > 0, \quad k = 1, 2, \ldots, m+1, \tag{3.10}$$

be any scaling matrix, and rewrite (3.9) as

$$r_m = W_{m+1}\Omega_{m+1}^{-1}\left(f_{m+1} - H_m^{(e)} z\right), \tag{3.11}$$

where

$$f_{m+1} = \omega_1 e_1^{(m+1)} \quad \text{and} \quad H_m^{(e)} = \Omega_{m+1} B_m^{(e)}. \tag{3.12}$$

We then define the mth iterate x_m of TFQMR by

$$x_m = x_0 + Y_m z_m, \tag{3.13}$$

where $z_m \in \mathbb{C}^m$ is the solution of the least-squares problem

$$\tau_m := \left\|f_{m+1} - H_m^{(e)} z_m\right\| = \min_{z \in \mathbb{C}^m} \left\|f_{m+1} - H_m^{(e)} z\right\|. \tag{3.14}$$

Note that by (3.6), (3.10), and (3.12), the matrix $H_m^{(e)}$ has full column rank m, and thus z_m is uniquely defined by (3.14). Furthermore, as we will show in the next two sections, the solution of (3.14) and hence x_m can be computed by means of simple recurrences.

Clearly, the iterates x_m still depend on the choice of the weights ω_k in (3.10). The standard strategy is to set

$$\omega_k = \|w_k\|, \qquad k = 1, 2, \ldots, m+1, \tag{3.15}$$

which means that all columns of $W_{m+1}\Omega_{m+1}^{-1}$ in the representation (3.11) are treated equally and scaled to be unit vectors. However, we will also consider a slightly less expensive choice of the weights in §5.

For the derivation of an actual implementation of the TFQMR method, the auxiliary iterates

$$\tilde{x}_m = x_0 + Y_m \tilde{z}_m, \quad \text{where} \quad \tilde{z}_m = H_m^{-1} f_m, \tag{3.16}$$

will be used. Here H_m denotes the $m \times m$ matrix obtained by deleting the last row of $H_m^{(e)}$. Using (3.6), (3.10), and (3.12), one readily verifies that H_m is indeed nonsingular and that

$$\tilde{z}_m = [\alpha_0 \quad \alpha_0 \quad \alpha_1 \quad \cdots \quad \alpha_{\lfloor (m-1)/2 \rfloor}]^T, \tag{3.17}$$

$$\omega_{m+1} = \left\| f_{m+1} - H_m^{(e)} \tilde{z}_m \right\|. \tag{3.18}$$

Finally, by comparing (3.16) and (3.17) with the update formula for the iterates x_n^{CGS} in step (2b) of Algorithm 2.1, we conclude that

$$\tilde{x}_{2n} = x_n^{\mathrm{CGS}}. \tag{3.19}$$

4. A lemma.

In this section, we show that the solutions of least-squares problems of the type (3.14) can be updated easily from step to step by using the auxiliary quantity $\tilde{z}_m$ defined in (3.16). This is true for more general upper Hessenberg matrices $H_m^{(e)}$, not only for the particular matrices given by (3.12) and (3.6). Therefore, we prove the following more general result, which might also be useful in other situations.

LEMMA 4.1. *Let $\omega_1 > 0$, $m \geq 1$, and*

$$H_m^{(e)} = \begin{bmatrix} H_m \\ h_{m+1,m} e_m^T \end{bmatrix} = \begin{bmatrix} H_{m-1}^{(e)} & * \\ 0 & h_{m+1,m} \end{bmatrix}, \tag{4.1}$$

where $e_m^T = [0 \ \cdots \ 0 \ 1]$, be an $(m+1) \times m$ upper Hessenberg matrix of full column rank m. For $k = m-1$, m, let $z_k \in \mathbb{C}^k$ denote the solution of the least-squares problem

$$\tau_k := \min_{z \in \mathbb{C}^k} \left\| f_{k+1} - H_k^{(e)} z \right\|, \quad \text{where } f_{k+1} = \omega_1 e_1^{(k+1)} \in \mathbb{R}^{k+1}. \tag{4.2}$$

Moreover, assume that the $m \times m$ matrix H_m in (4.1) is nonsingular, and set $\tilde{z}_m := H_m^{-1} f_m$. Then

$$z_m = (1 - c_m^2) \begin{bmatrix} z_{m-1} \\ 0 \end{bmatrix} + c_m^2 \tilde{z}_m, \tag{4.3}$$

$$\tau_m = \tau_{m-1} \vartheta_m c_m, \tag{4.4}$$

where

$$\vartheta_m = \frac{1}{\tau_{m-1}} \left\| f_{m+1} - H_m^{(e)} \tilde{z}_m \right\|, \qquad c_m = \frac{1}{\sqrt{1 + \vartheta_m^2}}. \tag{4.5}$$

Proof. Let $k = m-1$ or $k = m$. The standard approach (see, e.g., [10, Chap. 6]) for solving the least-squares problem (4.2) is based on a QR factorization of $H_k^{(e)}$,

$$Q_{k+1} H_k^{(e)} = \begin{bmatrix} R_k \\ 0 \end{bmatrix}, \tag{4.6}$$

where Q_{k+1} is unitary and R_k is a nonsingular upper triangular matrix. The solution of (4.2) is then given by

$$z_k = R_k^{-1} g_k, \quad \text{where} \begin{bmatrix} g_k \\ \tilde{\gamma}_{k+1} \end{bmatrix} = Q_{k+1} f_{k+1}, \quad \tilde{\gamma}_{k+1} \in \mathbb{C}, \tag{4.7}$$

and

$$\tau_k = |\tilde{\gamma}_{k+1}|. \tag{4.8}$$

Furthermore, since $H_k^{(e)}$ is an upper Hessenberg matrix, we can choose the matrix Q_{k+1} as a product of k Givens rotations. In particular, the factorization (4.6) for $k = m$ can then be easily updated from the one for $k = m - 1$, by setting

$$Q_{m+1} = \begin{bmatrix} I_{m-1} & 0 & 0 \\ 0 & c_m & -\overline{s_m} \\ 0 & s_m & c_m \end{bmatrix} \cdot \begin{bmatrix} Q_m & 0 \\ 0 & 1 \end{bmatrix}, \tag{4.9}$$

where $c_m \geq 0$ and $s_m \in \mathbb{C}$ are suitably chosen with $c_m^2 + |s_m|^2 = 1$. Also, note that

$$R_m = \begin{bmatrix} R_{m-1} & * \\ 0 & * \end{bmatrix}. \tag{4.10}$$

Using (4.9) and (4.1), we deduce from (4.6) (with $k = m$) that

$$Q_m H_m = \operatorname{diag}(1, \ldots, 1, c_m) R_m, \tag{4.11}$$

which, in particular, implies $c_m > 0$. Moreover, with (4.9), one easily verifies that

$$g_m = \begin{bmatrix} g_{m-1} \\ \gamma_m \end{bmatrix}, \quad \gamma_m = c_m \tilde{\gamma}_m, \quad \text{and} \quad \tilde{\gamma}_{m+1} = s_m \tilde{\gamma}_m. \tag{4.12}$$

From (4.11), (4.12), and (4.7), it follows that

$$\tilde{z}_m = (Q_m H_m)^{-1} \begin{bmatrix} g_{m-1} \\ \tilde{\gamma}_m \end{bmatrix} = R_m^{-1} \begin{bmatrix} g_{m-1} \\ \gamma_m / c_m^2 \end{bmatrix} = R_m^{-1} \begin{bmatrix} R_{m-1} z_{m-1} \\ \gamma_m / c_m^2 \end{bmatrix}. \tag{4.13}$$

Using (4.7), (4.12), and (4.13), we obtain

$$z_m = (1 - c_m^2) R_m^{-1} \begin{bmatrix} R_{m-1} z_{m-1} \\ 0 \end{bmatrix} + c_m^2 \tilde{z}_m,$$

which, in view of (4.10), is just (4.3).

Since Q_{m+1} is unitary, the following holds:

$$\left\| f_{m+1} - H_m^{(e)} \tilde{z}_m \right\| = \left\| Q_{m+1} f_{m+1} - Q_{m+1} H_m^{(e)} \tilde{z}_m \right\|,$$

and with (4.6), (4.7), (4.13), and (4.12), we arrive at

$$\begin{aligned} \left\| f_{m+1} - H_m^{(e)} \tilde{z}_m \right\| &= \left\| \begin{bmatrix} g_{m-1} \\ \gamma_m \\ \tilde{\gamma}_{m+1} \end{bmatrix} - \begin{bmatrix} g_{m-1} \\ \gamma_m / c_m^2 \\ 0 \end{bmatrix} \right\| \\ &= \left\| \begin{bmatrix} \gamma_m |s_m|^2 / c_m^2 \\ \tilde{\gamma}_{m+1} \end{bmatrix} \right\| = |\tilde{\gamma}_m| \frac{|s_m|}{c_m}. \end{aligned} \tag{4.14}$$

Finally, setting $\vartheta_m = |s_m| / c_m$ and using (4.8), the relations (4.5) and (4.4) follow from (4.14) and the identity on the right-hand side of (4.12). □

5. The TFQMR algorithm.

We now apply Lemma 4.1 to the particular least-squares problem (3.14). From (4.3), it follows that the TFQMR iterates (3.13) and the auxiliary iterates (3.16) are connected by

$$x_m = (1 - c_m^2)x_{m-1} + c_m^2 \tilde{x}_m. \tag{5.1}$$

Moreover, by (4.5), (4.4), and (3.18), we have

$$\vartheta_m = \frac{\omega_{m+1}}{\tau_{m-1}}, \quad c_m = \frac{1}{\sqrt{1+\vartheta_m^2}}, \quad \text{and} \quad \tau_m = \tau_{m-1}\vartheta_m c_m. \tag{5.2}$$

Setting

$$d_m = \frac{1}{\alpha_{\lfloor (m-1)/2 \rfloor}} \left(\tilde{x}_m - x_{m-1}\right), \tag{5.3}$$

we can rewrite (5.1) in the form

$$x_m = x_{m-1} + \eta_m d_m, \quad \text{where} \quad \eta_m = c_m^2 \alpha_{\lfloor (m-1)/2 \rfloor}. \tag{5.4}$$

Next note that by (3.16) and (3.17), we have

$$\tilde{x}_m = \tilde{x}_{m-1} + \alpha_{\lfloor (m-1)/2 \rfloor} y_m,$$

and together with (5.3) and (5.4) (both with m replaced by $m-1$), it follows that

$$d_m = y_m + \frac{\vartheta_{m-1}^2 \eta_{m-1}}{\alpha_{\lfloor (m-1)/2 \rfloor}} d_{m-1}, \quad \text{where} \quad \vartheta_{m-1}^2 := \frac{1 - c_{m-1}^2}{c_{m-1}^2}. \tag{5.5}$$

Next we note that with (3.1) and (3.3), the recursions for q_n and u_n in step (2a) and (2c) of Algorithm 2.1 can be rewritten as follows:

$$y_{2n} = y_{2n-1} - \alpha_{n-1} v_{n-1} \quad \text{and} \quad y_{2n+1} = w_{2n+1} + \beta_n y_{2n}. \tag{5.6}$$

Also, by multiplying the update formula for p_n in step (2c) of Algorithm 2.1 by A, we obtain the recursion

$$v_n = A y_{2n+1} + \beta_n (A y_{2n} + \beta_n v_{n-1}) \tag{5.7}$$

for $v_n = Ap_n$. By (3.4), the w_m's can be generated via

$$w_{m+1} = w_m - \alpha_{\lfloor (m-1)/2 \rfloor} A y_m. \tag{5.8}$$

Finally, by combining the recurrences (5.2), (5.4)–(5.8), we obtain an actual implementation for computing the iterates of the TFQMR method. First, we state the resulting algorithm for the standard weighting strategy (3.15).

Algorithm 5.1 (TFQMR Algorithm with Weights (3.15)).

```
(1) Start:
    (a) Choose $x_0 \in \mathbb{C}^N$;
    (b) Set $w_1 = y_1 = r_0 = b - Ax_0$, $v_0 = Ay_1$, $d_0 = 0$;
            $\tau_0 = \|r_0\|$, $\vartheta_0 = 0$, $\eta_0 = 0$;
    (c) Choose $\tilde{r}_0$ such that $\rho_0 = \tilde{r}_0^H r_0 \neq 0$.
(2) For $n = 1, 2, \dots,$ do:
```

```
(a) Set $\sigma_{n-1} = \tilde{r}_0^H v_{n-1}$, $\alpha_{n-1} = \rho_{n-1}/\sigma_{n-1}$;
    $y_{2n} = y_{2n-1} - \alpha_{n-1} v_{n-1}$;
(b) For $m = 2n-1, 2n$ do:
    • Set $w_{m+1} = w_m - \alpha_{n-1} A y_m$;
    •     $\vartheta_m = \|w_{m+1}\|/\tau_{m-1}$, $c_m = 1/\sqrt{1+\vartheta_m^2}$;
    •     $\tau_m = \tau_{m-1}\vartheta_m c_m$, $\eta_m = c_m^2 \alpha_{n-1}$;
    •     $d_m = y_m + (\vartheta_{m-1}^2 \eta_{m-1}/\alpha_{n-1}) d_{m-1}$;
    •     $x_m = x_{m-1} + \eta_m d_m$;
    • If $x_m$ has converged: stop;
(c) Set $\rho_n = \tilde{r}_0^H w_{2n+1}$, $\beta_n = \rho_n/\rho_{n-1}$;
    $y_{2n+1} = w_{2n+1} + \beta_n y_{2n}$;
    $v_n = A y_{2n+1} + \beta_n (A y_{2n} + \beta_n v_{n-1})$.
```

The convergence check in step (2b) is usually based on the norm $\|r_m\|$ of the residual vector r_m corresponding to x_m. These vectors are not generated explicitly in Algorithm 5.1. However, we have the following upper bound available at no extra cost:

$$\|r_m\| \le \left\|W_{m+1}\Omega_{m+1}^{-1}\right\| \cdot \|f_{m+1} - H_m^{(e)} z_m\| \le \sqrt{m+1}\,\tau_m. \tag{5.9}$$

The inequalities in (5.9) follow from (3.11) (with $z = z_m$), (3.14), and

$$\left\|W_{m+1}\Omega_{m+1}^{-1}\right\| \le \sqrt{m+1}.$$

The latter estimate holds since, by (3.15), the columns of the $N \times (m+1)$ matrix $W_{m+1}\Omega_{m+1}$ all have unit length. Therefore, the convergence criterion in Algorithm 5.1 can be checked for the upper bound on the right-hand side of (5.9), and the actual norm $\|r_m\|$ only needs to be computed in the final stages of the iteration process.

Notice that the vectors w_{2n} in Algorithm 5.1 are not used directly; only their norms $\|w_{2n}\|$ are needed. We can eliminate the w_{2n}'s, and thus save one inner product per iteration, by approximating the weights ω_{2n} in (3.15) as follows. First, recall that by (3.3) and (2.3),

$$\|w_{2n+1}\| = \|r_n^{\mathrm{CGS}}\| = \left\|(\varphi_n(A))^2 r_0\right\|. \tag{5.10}$$

Since by (3.2), $w_{2n} = \varphi_n(A)\varphi_{n-1}(A) r_0$, the relation (5.10) suggests the approximation

$$\|w_{2n}\| \approx \sqrt{\|r_{n-1}^{\mathrm{CGS}}\| \cdot \|r_n^{\mathrm{CGS}}\|}.$$

This leads to the weighting strategy

$$\omega_m = \begin{cases} \|r_n^{\mathrm{CGS}}\| & \text{if } m = 2n+1 \text{ is odd,} \\ \sqrt{\|r_{n-1}^{\mathrm{CGS}}\| \cdot \|r_n^{\mathrm{CGS}}\|} & \text{if } m = 2n \text{ is even.} \end{cases} \tag{5.11}$$

In the case of (5.11) and, more generally, for any choice of weights $\omega_k > 0$ in (3.10) that does not require $\|w_{2n}\|$, we can eliminate w_{2n} in Algorithm 5.1. The resulting procedure is, except for the update of the different iterates in step (2b), identical with the CGS Algorithm 2.1, and it can be stated as follows.

ALGORITHM 5.2 (TFQMR ALGORITHM WITH GENERAL WEIGHTS $\omega_k > 0$).

```
(1) Start:
    (a) Choose $x_0 \in \mathbb{C}^N$;
    (b) Set $p_0 = u_0 = r_0^{\mathrm{CGS}} = r_0 = b - Ax_0$, $v_0 = Ap_0$, $d_0 = 0$;
            $\tau_0 = \omega_1$, $\vartheta_0 = 0$, $\eta_0 = 0$;
    (c) Choose $\tilde{r}_0$ such that $\rho_0 = \tilde{r}_0^H r_0 \neq 0$.
(2) For $n = 1, 2, \ldots,$ do:
    (a) Set $\sigma_{n-1} = \tilde{r}_0^H v_{n-1}$, $\alpha_{n-1} = \rho_{n-1}/\sigma_{n-1}$;
            $q_n = u_{n-1} - \alpha_{n-1} v_{n-1}$;
            $r_n^{\mathrm{CGS}} = r_{n-1}^{\mathrm{CGS}} - \alpha_{n-1} A(u_{n-1} + q_n)$;
    (b) For $m = 2n-1, 2n$ do:
        • Set $\vartheta_m = \omega_{m+1}/\tau_{m-1}$, $c_m = 1/\sqrt{1+\vartheta_m^2}$;
        •     $\tau_m = \tau_{m-1}\vartheta_m c_m$, $\eta_m = c_m^2 \alpha_{n-1}$;
        •     $d_m = y_m + (\vartheta_{m-1}^2 \eta_{m-1}/\alpha_{n-1}) d_{m-1}$,
        •         where $y_m = \begin{cases} u_{n-1} & \text{if } m = 2n-1, \\ q_n & \text{if } m = 2n; \end{cases}$
        •     $x_m = x_{m-1} + \eta_m d_m$;
        • If $x_m$ has converged: stop;
    (c) Set $\rho_n = \tilde{r}_0^H r_n^{\mathrm{CGS}}$, $\beta_n = \rho_n/\rho_{n-1}$;
            $u_n = r_n^{\mathrm{CGS}} + \beta_n q_n$;
            $p_n = u_n + \beta_n(q_n + \beta_n p_{n-1})$;
            $v_n = Ap_n$.
```

Finally, we note that x_n^{CGS} can be generated easily during the course of the TFQMR Algorithms 5.1 or 5.2, whenever a CGS iterate is desired. Indeed, in view of (3.19), (5.1), and (5.3), the CGS and TFQMR iterates are connected by

$$x_n^{\mathrm{CGS}} = x_{2n-1} + \alpha_{n-1} d_{2n}.$$

6. Numerical examples.

6. Numerical examples. We have performed numerical experiments with the CGS method, the Bi-CGSTAB algorithm, and the TFQMR algorithm with both weighting strategies (3.15) and (5.11). In this section, we present the results for two linear systems that are quite difficult to solve by iterative methods.

In both examples we used $x_0 = 0$ as starting vector, and $\tilde{r}_0$ was chosen as a vector with random entries from a normal distribution with mean 0.0 and variance 1.0. As stopping criterion, we used

$$\frac{\|r_n\|}{\|r_0\|} \leq 10^{-6} \qquad (6.1)$$

for CGS and Bi-CGSTAB, and

$$\frac{\|r_m\|}{\|r_0\|} \leq 10^{-6} \qquad (6.2)$$

for TFQMR. Note that CGS and Bi-CGSTAB both produce only one iterate x_n per iteration, while QMR generates two iterates x_m, with indices $m = 2n-1$ and $m = 2n$, in the nth step of the iteration. However, work and storage per iteration are roughly the same for all three methods. For both examples, we show the relative residual norm (6.1), respectively (6.2), plotted versus the iteration number n. The solid line is the convergence curve for the TFQMR Algorithm 5.1 (with weighting strategy (3.15)), the dotted line shows the behavior of CGS, and the dashed line is the Bi-CGSTAB convergence curve.

Example 6.1. We consider the partial differential equation

$$-\frac{\partial^2 u}{\partial x^2}-\frac{\partial^2 u}{\partial y^2}+\gamma\left(x\frac{\partial u}{\partial x}+y\frac{\partial u}{\partial y}\right)+\beta u=f \quad \text{on} \quad (0,1)\times(0,1), \tag{6.3}$$

with Dirichlet boundary conditions. We discretize (6.3) using centered differences on a uniform 63×63 grid with mesh size $h = 1/64$. The resulting linear system has a sparse coefficient matrix A of order $N = 3969$. The parameters in (6.3) were chosen to be $\beta = -200$ and $\gamma = 100$. Note that this choice guarantees that the cell Reynolds number is smaller than one, and hence centered differences yield a stable discretization of (6.3). The right-hand side was chosen such that the vector of all ones is the exact solution of the linear system. The convergence behavior is shown in Fig. 6.1. For this example, we have also plotted (dashed-dotted line) the convergence curve for the TFQMR Algorithm 5.2 with the slightly less expensive weighting strategy (5.11). Note that the TFQMR curves for both strategies (3.15) and (5.11) are very close. This behavior is typical, and it was also observed in other numerical tests.

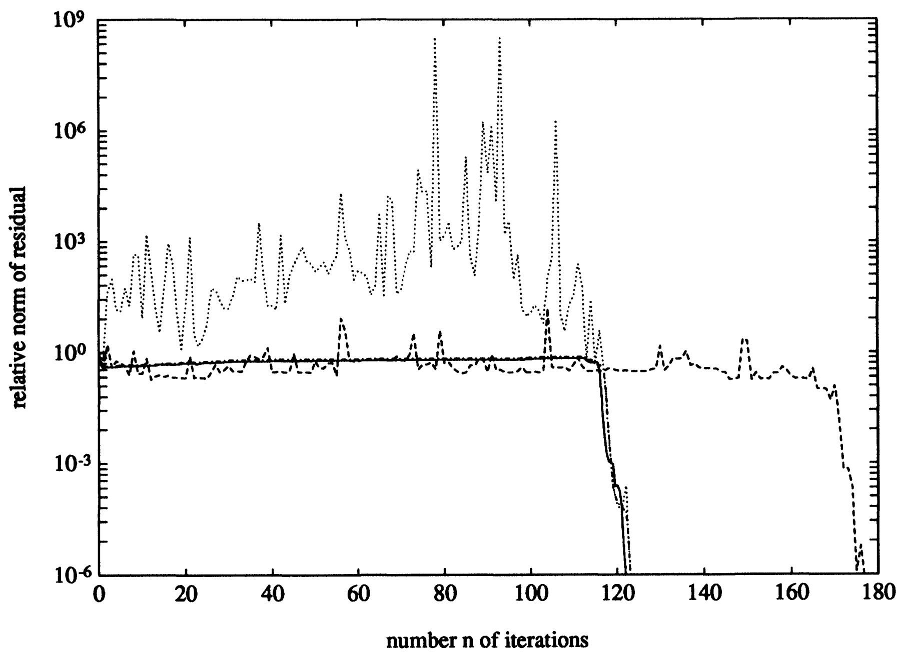

FIG. 6.1.

Example 6.2. This example was taken from the Harwell–Boeing set of sparse test matrices [2]. It is the fifth matrix from the SHERMAN collection, called SHERMAN5. It comes from a fully implicit black oil simulator on a $16 \times 23 \times 3$ grid, with three unknowns per grid point. The order of the matrix is 3312, and it has 20793 nonzero elements. The right-hand side b was also chosen as a vector with random entries. The convergence curves are shown in Fig. 6.2.

Both examples clearly demonstrate that the TFQMR method is certainly a remedy for the erratic convergence behavior of CGS. They also show that, in general, the residual norms for Bi-CGSTAB can still oscillate considerably, and that for difficult problems, convergence can be significantly slower than for CGS and TFQMR.

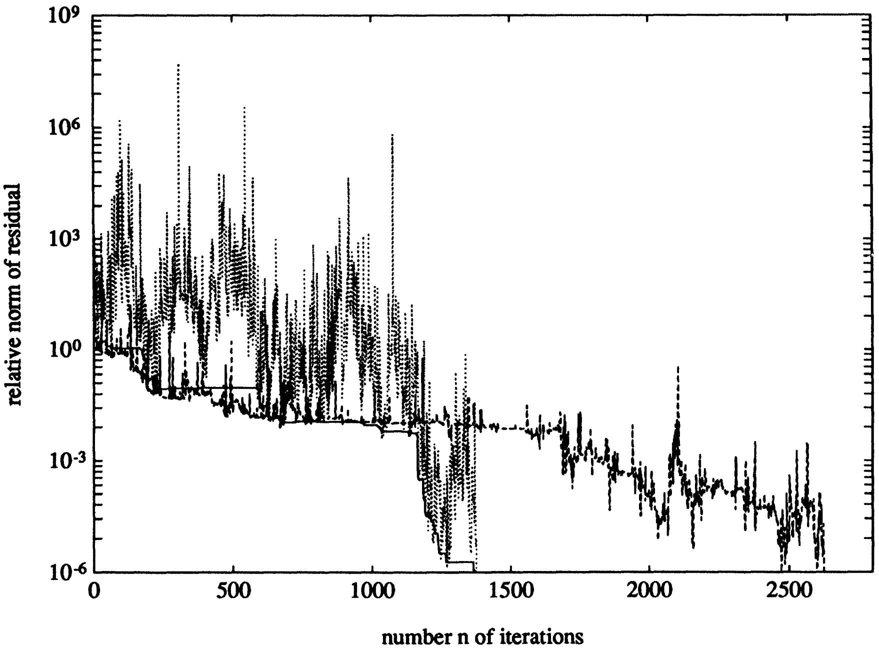

FIG. 6.2.

7. Conclusion. In this note, we proposed the TFQMR algorithm for solving general nonsingular non-Hermitian linear systems. The TFQMR method is closely related to the CGS algorithm, and it can be implemented by changing only a few lines in standard CGS. However, unlike CGS, the iterates of the TFQMR algorithm are characterized by a quasi minimization of the residual norm. This leads to smooth convergence curves for the TFQMR method, while CGS typically exhibits a rather irregular convergence behavior with wild oscillations in the residual norm.

Like standard CGS, the TFQMR algorithm described here can break down prematurely. Although these breakdowns are very rare in practice, for a robust implementation of the method that can be used as a black-box solver, it is crucial to incorporate look-ahead steps that allow to leap over those iterations in which exact breakdowns or near-breakdowns would occur in the standard process. The details of such a TFQMR algorithm with look-ahead will be presented elsewhere.

Finally, we remark that, based on the quasi-minimal residual property, one can derive upper bounds for the residual norms of the TFQMR iterates. Such a convergence result for the TFQMR method is given in [5, Thm. 6].

REFERENCES

[1] C. BREZINSKI AND H. SADOK, *Avoiding breakdown in the* CGS *algorithm*, Numer. Algorithms, 1 (1991), pp. 199–206.

[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] 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.

[4] R. W. Freund, *Conjugate gradient-type methods for linear systems with complex symmetric coefficient matrices*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 425–448.

[5] ———, *Quasi-kernel polynomials and convergence results for quasi-minimal residual iterations*, in Numerical Methods of Approximation Theory, D. Braess and L. L. Schumaker, eds., Birkhäuser, Basel, 1992, to appear.

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

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

[8] R. W. Freund and H. Zha, *Simplifications of the nonsymmetric Lanczos process and a new algorithm for solving symmetric indefinite linear systems*, Numerical Analysis Manuscript, AT&T Bell Laboratories, Murray Hill, NJ, 1992, in preparation.

[9] C. W. Gear and Y. Saad, *Iterative solution of linear equations in* ODE *codes*, SIAM J. Sci. Statist. Comput., 4 (1983), pp. 583–601.

[10] G. H. Golub and C. F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 1983.

[11] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Standards, 49 (1952), pp. 409–436.

[12] C. Lanczos, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*, J. Res. Nat. Bur. Standards, 45 (1950), pp. 255–282.

[13] ———, *Solution of systems of linear equations by minimized iterations*, J. Res. Nat. Bur. Standards, 49 (1952), pp. 33–53.

[14] B. N. Parlett, D. R. Taylor, and Z. A. Liu, *A look-ahead Lanczos algorithm for unsymmetric matrices*, Math. Comp., 44 (1985), pp. 105–124.

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

[16] H. A. van der Vorst, BI-CGSTAB: *A fast and smoothly converging variant of* BI-CG *for the solution of nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 631–644.