Numerical Linear Algebra with Applications, Vol. 1(6), 571–581 (1994)

A Simpler GMRES

Homer F. Walker and Lu Zhou

Department of Mathematics and Statistics, Utah State University, Logan, UT 84322-3900, USA

The generalized minimal residual (GMRES) method is widely used for solving very large, nonsymmetric linear systems, particularly those that arise through discretization of continuous mathematical models in science and engineering. By shifting the Arnoldi process to begin with Ar_0 instead of r_0, we obtain simpler Gram–Schmidt and Householder implementations of the GMRES method that do not require upper Hessenberg factorization. The Gram–Schmidt implementation also maintains the residual vector at each iteration, which allows cheaper restarts of GMRES(m) and may otherwise be useful.

KEY WORDS GMRES GMRES(m) Large-scale nonsymmetric linear systems

1. Introduction

The generalized minimal residual (GMRES) method [1] is an iterative method for solving general nonsymmetric linear systems. It is particularly appropriate for very large systems such as those that arise through discretization of continuous mathematical models in science and engineering. It has enjoyed wide success on many problems of scientific and economic importance, such as aircraft dynamics modeling, semiconductor modeling, and reservoir simulation.

For a linear system

$$Ax = b, \qquad A \in \mathbf{R}^{n\times n} \text{ nonsingular} \tag{1.1}$$

GMRES begins with an initial $x_0 \in \mathbf{R}^n$ and characterizes the kth iterate as $x_k = x_0 + z_k$, where z_k solves the least-squares problem

$$\min_{z\in\mathcal{K}_k(r_0)} \|b - A(x_0 + z)\|_2 = \min_{z\in\mathcal{K}_k(r_0)} \|r_0 - Az\|_2 \tag{1.2}$$

In (1.2), $r_0 = b - Ax_0$ and, for $v \in \mathbf{R}^n$, $\mathcal{K}_k(v)$ is the Krylov subspace

$$\mathcal{K}_k(v) \equiv \text{span}\{v, Av, \ldots, A^{k-1}v\}$$

CCC 1070–5325/94/060571–11

Received January 1992
Revised July 1994

Central to the usual implementations of GMRES is the Arnoldi process, which we write as follows:

Algorithm 1.1. *Arnoldi process*
Let v_1 be given with $\|v_1\|_2 = 1$.
For $k = 1, 2, \cdots$, do:
Set $v_{k+1} \equiv \pm\Pi_k^{\perp} Av_k / \|\Pi_k^{\perp} Av_k\|_2$.

Here, $\Pi_k^{\perp}$ denotes orthogonal projection onto the orthogonal complement of $\mathcal{K}_k(v_1)$. Note that the process breaks down at step k if and only if $\Pi_k^{\perp} Av_k = 0$, equivalently, $\mathcal{K}_{k+1}(v_1)$ has dimension k.

In the usual implementations of GMRES, the process is applied with $v_1 = \pm r_0/\|r_0\|_2$. The orthonormalization at each step can be done using either the (modified) Gram–Schmidt process [1] or Householder transformations [5,6]. Each $\{v_1, \cdots, v_k\}$ is an orthonormal basis of $\mathcal{K}_k(r_0)$ that reduces the least-squares problem (1.2) to an upper Hessenberg least-squares problem, which is typically solved by QR factorization with Givens rotations.

In this note, we observe that simpler GMRES implementations result when the Arnoldi process is applied with $v_1 = \pm Ar_0/\|Ar_0\|_2$. In this case, $\{r_0, v_1, \cdots, v_{k-1}\}$ is clearly a basis of $\mathcal{K}_k(r_0)$, and it turns out that (1.2) reduces directly to an upper triangular least-squares problem. There is no need for factorization of an upper Hessenberg matrix; in addition, in our Gram–Schmidt implementation, the residual vector is maintained at each iteration, which gives cheaper re-initialization following each iteration cycle of the restarted method GMRES(m) and may otherwise be useful in some applications.

2. The implementations

Suppose $r_0 \neq 0$. Then, since A is nonsingular, $Ar_0 \neq 0$ and $v_1 \equiv \pm Ar_0/\|Ar_0\|_2$ is well-defined. Setting $\rho_{11} \equiv \pm\|Ar_0\|_2$, we have $Ar_0 = \rho_{11} v_1$.

Suppose we have applied the Arnoldi process with this v_1 to produce orthonormal $v_1, \cdots, v_{k-1}$ for some $k > 1$. If the GMRES iterate x_{k-1} is not the solution of (1.1), then $r_0 \notin A(\mathcal{K}_{k-1}(r_0))$. Since $A(\mathcal{K}_{k-1}(r_0)) = \mathcal{K}_{k-1}(v_1)$ and $\mathcal{K}_{k-1}(v_1)$ has dimension $k-1$, it follows that $\mathcal{K}_k(r_0)$ and, hence, $\mathcal{K}_k(v_1) = A(\mathcal{K}_k(r_0))$ have dimension k. Then $\Pi_{k-1}^{\perp} Av_{k-1} \neq 0$, and the Arnoldi process successfully generates v_k; furthermore, we have $Av_{k-1} = \sum_{i=1}^{k} \rho_{ik} v_i$ for unique ρ_{ik}'s, with $\rho_{kk} \neq 0$.

If we have $v_1, \cdots, v_k$ for some $k \geq 1$ and set $r_k \equiv \Pi_k^{\perp} r_0$, then we can write

$$r_0 = r_k + (v_1, \cdots, v_k) w_k = r_k + \sum_{i=1}^{k} \xi_i v_i \tag{2.1}$$

for a unique $w_k = (\xi_1, \cdots, \xi_k)^T \in \mathbf{R}^k$. Defining nonsingular, upper triangular

$$R_k \equiv \begin{pmatrix} \rho_{11} & \cdots & \rho_{1k} \\ & \ddots & \vdots \\ & & \rho_{kk} \end{pmatrix} \in \mathbf{R}^{k \times k}$$

we have for $y \in \mathbf{R}^k$,

$$\begin{cases} r_0 - Ar_0 y = r_1 + v_1 (w_1 - R_1 y), & \text{if } k = 1 \\ r_0 - A(r_0, v_1, \cdots, v_{k-1}) y = r_k + (v_1, \cdots, v_k)(w_k - R_k y), & \text{if } k > 1 \end{cases}$$

It follows that the GMRES correction z_k is given by

$$z_k = \begin{cases} r_0 y_1, & \text{if } k = 1 \\ (r_0, v_1, \cdots, v_{k-1}) y_k, & \text{if } k > 1 \end{cases} \tag{2.2}$$

where $y_k = R_k^{-1} w_k$; furthermore, r_k is the residual vector. From (2.1), we have $r_k + \xi_k v_k = r_{k-1}$, which yields the following expressions for updating $\|r_k\|_2$:

$$\begin{aligned} \|r_k\|_2 &= \sqrt{\|r_{k-1}\|_2^2 - \xi_k^2} = \|r_{k-1}\|_2 \sqrt{1 - (\xi_k / \|r_{k-1}\|_2)^2} \\ &= \|r_{k-1}\|_2 \sin\left(\cos^{-1}(\xi_k / \|r_{k-1}\|_2)\right) \end{aligned}$$

The last is most trustworthy provided sin and $\cos^{-1}$ can be evaluated accurately.

We outline two implementations of GMRES(m) based on these observations, the first using the (modified) Gram–Schmidt process and the second using Householder transformations. For the Gram–Schmidt implementation, it is convenient to note that, writing $y_k = (\eta_1, \cdots, \eta_k)^T$, we have from (2.1) and (2.2) that

$$z_k = \begin{cases} \eta_1 r_0, & \text{if } k = 1 \\ \eta_1 r_{k-1} + \sum_{i=1}^{k-1} (\eta_{i+1} + \eta_1 \xi_i) v_i, & \text{if } k > 1 \end{cases}$$

In both implementations, for numerical reasons discussed in section 3, the initial residual vector is normalized at the beginning of each iteration cycle; this requires multiplying the correction by the initial residual norm following each cycle.

Algorithm 2.1. *Simpler Gram–Schmidt GMRES(m)*

Initialize: Given x, set $r \equiv b - Ax$ and $\rho_0 \equiv \|r\|_2$. If $\rho_0 \leq \text{TOL}$, accept x and exit; otherwise, update $r \leftarrow r/\rho_0$ and set $\rho = 1$.

Iterate: For $k = 1, \cdots, m$, do:

1. Evaluate $v_k \equiv A v_{k-1}$ ($v_1 \equiv Ar$).
2. If $k > 1$, then for $i = 1, \cdots, k-1$, do:
   a. Set $\rho_{i,k} \equiv v_i^T v_k$.
   b. Update $v_k \leftarrow v_k - \rho_{i,k} v_i$.
3. Set $\rho_{k,k} \equiv \|v_k\|_2$; update $v_k \leftarrow v_k / \rho_{k,k}$.
4. Set $R_k \equiv \begin{pmatrix} & & \rho_{1,k} \\ & R_{k-1} & \vdots \\ 0 \cdots & 0 & \rho_{k,k} \end{pmatrix}$ ($R_1 \equiv (\rho_{1,1})$).
5. Set $\xi_k \equiv r^T v_k$; update $\rho \leftarrow \rho \sin\left(\cos^{-1}(\xi_k/\rho)\right)$. If $\rho \cdot \rho_0 \leq \text{TOL}$, go to Solve.
6. Update $r \leftarrow r - \xi_k v_k$.

Solve: Let k be the final iteration number from Iterate.

1. Solve $R_k y = (\xi_1, \cdots, \xi_k)^T$ for $y = (\eta_1, \cdots, \eta_k)^T$.
2. Form $z = \begin{cases} \eta_1 r, & \text{if } k = 1, \\ \eta_1 r + \sum_{i=1}^{k-1} (\eta_{i+1} + \eta_1 \xi_i) v_i, & \text{if } k > 1. \end{cases}$

3. Update $x \leftarrow x + \rho_0 z$.
4. If $\rho \cdot \rho_0 \leq \text{TOL}$, accept x and exit; otherwise, update
   $r \leftarrow (r - \xi_k v_k)/\rho$, $\rho_0 \leftarrow \rho \cdot \rho_0$, $\rho \leftarrow 1$ and return to Iterate.

Remark The update $r \leftarrow (r - \xi_k v_k)/\rho$ updates r from the previous to the current (normalized) residual vector. It is placed above to avoid one 'saxpy' in step 2 of SOLVE and an unnecessary residual update on termination. If the current residual is desired on termination, then an update $r \leftarrow \rho_0(r - \xi_k v_k)$ should be done before exiting. An analogous remark holds for Algorithm 2.2 below.

In outlining the Householder implementation, our conventions are that, for $i = 1, \cdots, n$, $e_i \in \mathbf{R}^n$ denotes the ith column of $I_n \in \mathbf{R}^{n \times n}$ and P_i denotes a Householder transformation that 'transforms' only the ith through nth coordinates of vectors on which it acts. We note the following: Suppose $P_1, \cdots, P_k$ are such that

$$\begin{pmatrix} R_k \\ 0 \end{pmatrix} = \begin{cases} P_1 A r_0, & \text{if } k = 1 \\ P_k \cdots P_1 A(r_0, v_1, \cdots, v_{k-1}), & \text{if } k > 1 \end{cases}$$

Since R_k is invertible and

$$\begin{cases} A r_0 = v_1 R_1, & \text{if } k = 1 \\ A(r_0, v_1, \cdots, v_{k-1}) = (v_1, \cdots, v_k) R_k, & \text{if } k > 1 \end{cases}$$

it follows that

$$P_k \cdots P_1 (v_1, \cdots, v_k) = \begin{pmatrix} I_k \\ 0 \end{pmatrix}, \qquad I_k \in \mathbf{R}^{k \times k} \tag{2.3}$$

and that $v_j = P_1 \cdots P_k e_j = P_1 \cdots P_j e_j$ for $j = 1, \cdots, k$. Note that, with $y_k = (\eta_1, \cdots, \eta_k)^T$, (2.2) and (2.3) give

$$z_k = \begin{cases} P_1(\eta_1 P_1 r_0), & \text{if } k = 1 \\ P_1 \cdots P_k \left[\eta_1 P_k \cdots P_1 r_0 + (\eta_2, \cdots, \eta_k, 0, \cdots)^T\right], & \text{if } k > 1 \end{cases}$$

Also, since $(P_k \cdots P_1 r_k)^T e_j = (P_k \cdots P_1 r_k)^T (P_k \cdots P_1 v_j) = r_k^T v_j = 0$ for $j = 1, \cdots, k$, it follows from (2.1) and (2.3) that $\xi_k = e_k^T P_k \cdots P_1 r_0$ and

$$r_k = P_1 \cdots P_k \begin{pmatrix} 0 & \\ & I_{n-k} \end{pmatrix} P_k \cdots P_1 r_0, \qquad I_{n-k} \in \mathbf{R}^{(n-k)\times(n-k)}$$

Algorithm 2.2. *Simpler Householder GMRES(m)*

Initialize: Given x, set $r \equiv b - Ax$ and $\rho_0 \equiv \|r\|_2$. If $\rho_0 \leq \text{TOL}$, accept x and exit; otherwise, update $r \leftarrow r/\rho_0$ and set $\rho = 1$.

Iterate: For $k = 1, \cdots, m$, do:

1. Evaluate $\tilde{v}_k \equiv P_{k-1} \cdots P_1 A P_1 \cdots P_{k-1} e_{k-1}$ $(\tilde{v}_1 \equiv Ar)$.
2. Determine P_k such that $P_k \tilde{v}_k = (\rho_{1,k}, \cdots, \rho_{k,k}, 0, \cdots)^T$.
3. Set $R_k \equiv \begin{pmatrix} & & \rho_{1,k} \\ & R_{k-1} & \vdots \\ 0 \cdots 0 & & \rho_{k,k} \end{pmatrix}$ $(R_1 \equiv (\rho_{1,1}))$.
4. Update $r \leftarrow P_k r$, set $\xi_k \equiv e_k^T r$, and update $\rho \leftarrow \rho \sin\left(\cos^{-1}(\xi_k/\rho)\right)$. If $\rho \cdot \rho_0 \leq$ TOL, go to Solve.

Solve: Let k be the final iteration number from Iterate.

1. Solve $R_k y = (\xi_1, \cdots, \xi_k)^T$ for $y = (\eta_1, \cdots, \eta_k)^T$.
2. Form $z = \begin{cases} P_1(\eta_1 r), & \text{if } k = 1, \\ P_1 \cdots P_k \left[\eta_1 r + (\eta_2, \cdots, \eta_k, 0, \cdots)^T\right], & \text{if } k > 1. \end{cases}$
3. Update $x \leftarrow x + \rho_0 z$.
4. If $\rho \cdot \rho_0 \leq \text{TOL}$, accept x and exit; otherwise, update

$$r \leftarrow P_1 \cdots P_k \begin{pmatrix} 0 & \\ & I_{n-k} \end{pmatrix} r/\rho,\ \rho_0 \leftarrow \rho \cdot \rho_0,\ \rho \leftarrow 1$$ and return to Iterate.

Remark In [5,6], a Householder GMRES(m) implementation is given in which products of Householder transformations are expanded as sums. Such expansion gives more 'BLAS-2' matrix–vector operations and therefore may offer advantages in some computing environments. It would be straightforward to give such a variation of Algorithm 2.2.

Remark In both Algorithms 2.1 and 2.2, the final updating of r and ρ_0 in step 4 of SOLVE can be replaced by re-evaluating $r \leftarrow (b - Ax)/\|b - Ax\|_2$ and $\rho_0 \leftarrow \|b - Ax\|_2$. This might be cheaper in some applications, especially for Algorithm 2.2. Also, we have seen in experiments that it can give somewhat more accurate values when the residual has been considerably reduced, resulting in more reliable performance near the limits of residual reduction. Thus it may be advisable if high accuracy is desired; see [3].

3. Discussion

Because upper-Hessenberg QR factorization is unnecessary, Algorithms 2.1 and 2.2 are simpler to program and require $O(m^2)$ fewer arithmetic operations over each iteration cycle of m steps than the usual Gram–Schmidt and Householder implementations. In addition, in Algorithm 2.1, the residual vector is always available or easily obtained. In the usual Gram–Schmidt implementation, producing the residual vector without a re-evaluation $r \leftarrow b - Ax$ costs $O(kn)$ arithmetic operations after k steps and, in particular, $O(mn)$ arithmetic operations after a cycle of m steps. Having the residual vector allows cheaper restarting after each iteration cycle and may offer other advantages in some situations. However, the overall cost per cycle of each of these implementations is $O(m^2 n)$ arithmetic operations plus m products of A with vectors, and so these arithmetic advantages may be relatively slight in practice.

To discuss numerical accuracy, we first note that, in all GMRES implementations, solving the least-squares problem (1.2) involves determining some $B_k \in \mathbf{R}^{n \times k}$, the columns of which constitute a basis of $\mathcal{K}_k(r_0)$. Then (1.2) becomes

$$\min_{y \in \mathbf{R}^k} \|r_0 - AB_k y\|_2$$

The accuracy with which the solution y_k is computed depends on the conditioning of AB_k, and the accuracy with which $z_k = B_k y_k$ is formed from the computed y_k depends on the conditioning of B_k. Thus the conditioning of B_k is a matter of concern, even though its precise relationship to that of AB_k may not be clear in general.

In the usual GMRES implementation, the columns of B_k are the orthonormal vectors $v_1, \cdots, v_k$ produced by the Arnoldi process with $v_1 = \pm r_0/\|r_0\|_2$; hence, this B_k is ideally

conditioned. In the implementations here, we have

$$B_k = (r_0/\|r_0\|_2, v_1, \cdots, v_{k-1}) \tag{3.1}$$

where $v_1, \cdots, v_{k-1}$ are produced by the Arnoldi process with $v_1 = \pm Ar_0/\|Ar_0\|_2$. This may become ill-conditioned, but Lemma 3.1 below shows that it does only if commensurate residual reduction was achieved at the previous step. In the following, the condition number $\kappa_2(M)$ of a matrix M is the quotient of the largest and smallest singular values of M.

Lemma 3.1. *For B_k given by (3.1), we have*

$$\kappa_2(B_k) \le 2\frac{\|r_0\|_2}{\|r_{k-1}\|_2} \tag{3.2}$$

Proof From (3.1) and (2.1), we have

$$B_k = (r_{k-1}/\|r_{k-1}\|_2, v_1, \cdots, v_{k-1})M_k$$

where

$$M_k \equiv \begin{pmatrix} \|r_{k-1}\|_2/\|r_0\|_2 & & \\ \xi_1/\|r_0\|_2 & 1 & \\ \vdots & & \ddots \\ \xi_{k-1}/\|r_0\|_2 & & 1 \end{pmatrix} \in \mathbf{R}^{k\times k}$$

Then $\kappa_2(B_k) = \kappa_2(M_k)$, and (3.2) follows from Lemma 3.2 below. ∎

Lemma 3.2. *Suppose $M \in \mathbf{R}^{k\times k}$ has the form $M = (a, e_2, \cdots, e_k)$, where $a = (\alpha_1, \cdots, \alpha_k)^T$ with $\alpha_1 \neq 0$. Then $\kappa_2(M) \le (\|a\|_2^2 + 1)/|\alpha_1|$.*

Proof In this case, $\kappa_2(M) = \|M\|_2\|M^{-1}\|_2$. If $u = (v_1, \cdots, v_k)^T$ is a unit vector, then so is $\hat{u} \equiv (1 - v_1^2)^{-1/2}(0, v_2, \cdots, v_k)^T$, and we have

$$\begin{aligned} \|Mu\|_2 &= \|v_1 a + (1 - v_1^2)^{1/2}\hat{u}\|_2 \le |v_1|\,\|a\|_2 + (1 - v_1^2)^{1/2}\|\hat{u}\|_2 \\ &\le \left(\|a\|_2^2 + \|\hat{u}\|_2^2\right)^{1/2} = \left(\|a\|_2^2 + 1\right)^{1/2} \end{aligned}$$

It follows that $\|M\|_2 \le \left(\|a\|_2^2 + 1\right)^{1/2}$. We also have $M^{-1} = (\hat{a}, e_2, \cdots, e_k)$, where $\hat{a} = (1/\alpha_1, -\alpha_2/\alpha_1, \cdots, -\alpha_k/\alpha_1)^T$. The same reasoning gives

$$\|M^{-1}\|_2 \le \left(\|\hat{a}\|_2^2 + 1\right)^{1/2} = \frac{\left(\|a\|_2^2 + 1\right)^{1/2}}{|\alpha_1|}$$

and the lemma follows. ∎

The bound (3.2) shows that $\kappa_2(B_k)$ is unlikely to become undesirably large in practice. In GMRES(m), $\|r_0\|_2$ in (3.2) is the residual norm at outset of the current iteration cycle. In typical applications, the residual norm reduction over each cycle is usually sufficiently limited that $\|r_0\|_2/\|r_{k-1}\|_2$ does not become undesirably large. If it were to become so, then a smaller choice of m would probably yield a more economical method. At any rate, if the stopping tolerance is not especially small, then the method will certainly terminate at step $k - 1$ of the current cycle if $\|r_0\|_2/\|r_{k-1}\|_2$ becomes undesirably large.

Our reason for normalizing initial residuals in Algorithms 2.1 and 2.2 can be explained as follows. Without normalization, we would have $B_k = (r_0, v_1, \cdots, v_{k-1})$ instead of (3.1) and

$$\kappa_2(B_k) \leq \frac{\|r_0\|_2^2 + 1}{\|r_{k-1}\|_2}$$

instead of (3.2). This is unsatisfactory since the right-hand side grows without bound as $\|r_{k-1}\|_2$ becomes small, regardless of the size of $\|r_0\|_2$. Also, we have seen in experiments that for this B_k, AB_k often becomes undesirably ill-conditioned when the residual norm becomes small.

The GMRES implementations here have some features in common with other methods. The scheme used here for maintaining the residual vector is also used in ORTHODIR [7] and GMRESR [4]. There is also a closer relationship with (restarted) ORTHODIR, which is mathematically equivalent to (restarted) GMRES: When implemented so that only one product of A with a vector is required at each iteration, ORTHODIR explicitly maintains at the kth step two arrays $q_1, \cdots, q_k$ and $u_1, \cdots, u_k$ such that $A(q_1, \cdots, q_k) = (u_1, \cdots, u_k)$, with $q_1 = \lambda r_0$ for a λ determined by the scaling (if any) used in a particular implementation. The first array is used only to update the approximate solution at each iteration by $x_k = x_{k-1} + (u_k^T r_{k-1}/\|u_k\|_2^2)q_k$. The u_is are constructed to be mutually orthogonal, and if the method is scaled so that they are normalized as well, then one can easily show that $(u_1, \cdots, u_k) = (v_1, \cdots, v_k)$, $(q_1, \cdots, q_k) = (r_0, v_1, \cdots, v_{k-1})R_k^{-1}$, and the solution update is $x_k = x_{k-1} + \xi_k q_k$, where $(v_1, \cdots, v_k)$, R_k, and ξ_k are as in section 2 above. Thus ORTHODIR *with this scaling* uses essentially the same artifacts as our GMRES implementations but differs by explicitly forming the last column of $(q_1, \cdots, q_k) = (r_0, v_1, \cdots, v_{k-1})R_k^{-1}$ and using it to update the approximate solution at each step, rather than maintaining R_k and $w_k = (\xi_1, \cdots, \xi_k)^T$ and, for the final k, solving $R_k y_k = w_k$ and forming $x_k = x_0 + (r_0, v_1, \cdots, v_{k-1})y_k$. In addition to requiring approximately twice the arithmetic and storage, this incremental updating of the approximate solution is numerically less sound than accumulating corrections over a number of steps when corrections are small relative to the approximate solution. In general, ORTHODIR is considered to be less numerically sound than GMRES, but this has been attributed to poor scaling [1, p. 857].

4. Numerical experiments

We address the issue of whether the GMRES implementations given here are as numerically sound as the usual implementations, focusing on Algorithm 2.1 because it is more likely to be used in practice than Algorithm 2.2. We report on experiments in which we applied Algorithm 2.1 and the usual Gram–Schmidt GMRES(m) implementation to a test problem and noted both the comparative performance and the accuracy of the two methods. In assessing the comparative performance, we monitored the *true* residual norms $\|b - Ax_k\|_2$ generated by the two methods. In assessing accuracy, we compared the true residual norms with the residual norm values that are maintained recursively by each method.

The test problem is a discretization of

$$\begin{aligned} \Delta u + cu + d\frac{\partial u}{\partial x} &= f \qquad \text{in } D \\ u &= 0 \qquad \text{on } \partial D \end{aligned}$$

where $D = [0, 1] \times [0, 1]$ and c and d are constants. In the experiments here, we took $f \equiv 1$ and used a uniform 100×100 mesh so that $n = 10\,000$. Discretization was by the usual second-order centered differences. Preconditioning, when used, was with a fast Poisson solver from FISHPACK [2]; this is very effective for small values of c and d but becomes less so as c and d increase. In both experiments reported below, we took $m = 10$ in Algorithm 2.1 and in the usual Gram–Schmidt GMRES(m) implementation. This m is effective for this problem; in unreported experiments, other values were used with qualitatively similar results. All computing was done in double precision on Sun Microsystems workstations.

In the first experiment, we took $c = d = 100$ and used no preconditioning. We ran Algorithm 2.1 and the usual Gram–Schmidt GMRES(m) implementation for 600 iterations, which was sufficient to reach the limits of residual norm reduction. Figure 1 shows the true residual norms of the iterates. It is evident that there are some differences in the iterates produced by the two methods, although both methods produced the same ultimate residual norm reduction. We regard these differences as insignificant; when the experiment was repeated with both methods re-evaluating the residual at the beginning of each iteration cycle, Algorithm 2.1 reduced the residual norm slightly faster than the usual implementation in the later iterations, rather than slightly slower as in Figure 1. Figure 2 shows the true and recursive residual norms for Algorithm 2.1. One sees that the algorithm maintains very satisfactory accuracy until very near the limit of residual norm reduction, after which the recursive residual norm is further (and erroneously) reduced while the true residual norm stagnates (as one would expect). The analogous figure for the usual Gram–Schmidt GMRES(m) implementation is not shown but is qualitatively very similar.

In our second experiment, we took $c = d = 10$ and used fast Poisson preconditioning. We ran Algorithm 2.1 and the usual Gram–Schmidt GMRES(m) implementation for 35 iterations, which gave the limits of residual norm reduction. Figure 3 shows the true residual norms. One sees that although the two methods agreed until near the end, the usual Gram–Schmidt implementation ultimately gave somewhat greater residual norm reduction. When the experiment was repeated with both methods re-evaluating the residual at the beginning of each iteration cycle, the residual norm plots for the two methods were visually indistinguishable. Figure 4 shows the true and recursive residual norms for Algorithm 2.1. As in the previous experiment, the algorithm maintains very satisfactory accuracy until very near the limit of residual norm reduction.

Overall, these experiments indicate that Algorithm 2.1 can be trusted to maintain accuracy until very near the limits of residual norm reduction. The performance is comparable to but may differ somewhat from that of the usual Gram–Schmidt GMRES(m) implementation. Most significantly, the latter gave somewhat greater residual norm reduction in some experiments (with preconditioning); these differences were eliminated when both methods were run with re-evaluation of the residual at the beginning of each iteration cycle. In fact, such residual re-evaluation gave slightly greater ultimate residual norm reduction for both methods in all cases.

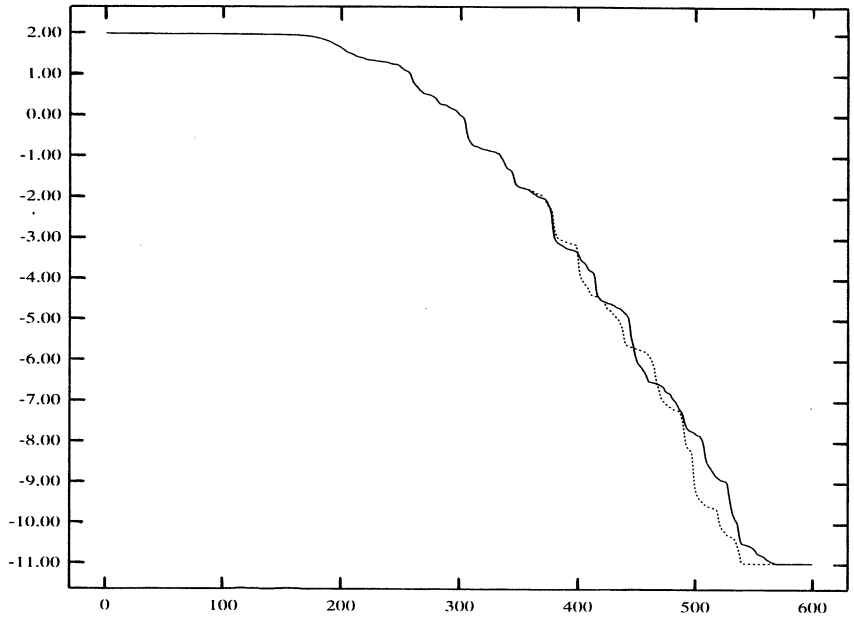

Figure 1. Log_{10} of the true residual norms versus the number of iterations; $c = d = 100$, no preconditioning. Solid curve: Algorithm 2.1; dotted curve: the usual Gram–Schmidt GMRES(m) implementation

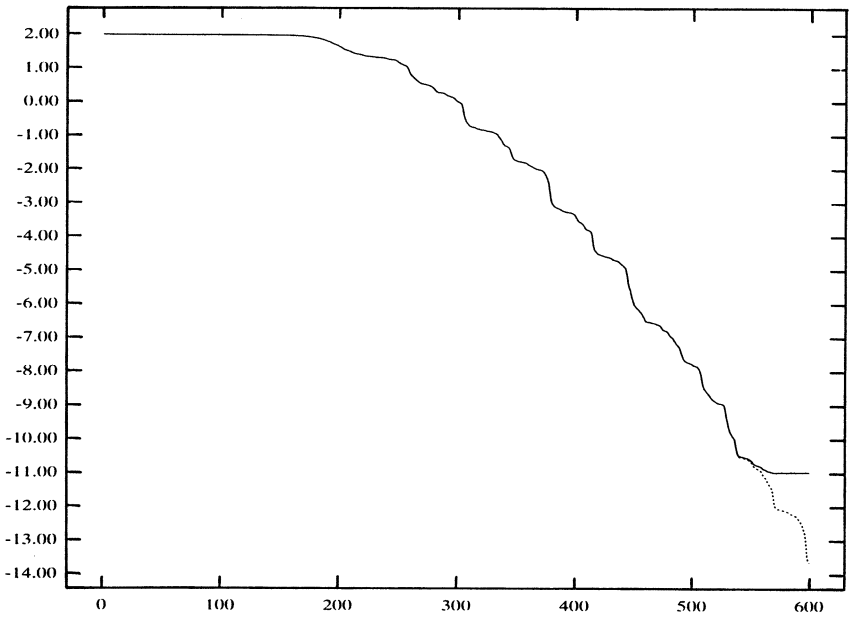

Figure 2. Log_{10} of the true and recursive residual norms versus the number of iterations for Algorithm 2.1; $c = d = 100$, no preconditioning. Solid curve: true residual norms; dotted curve: recursive residual norms

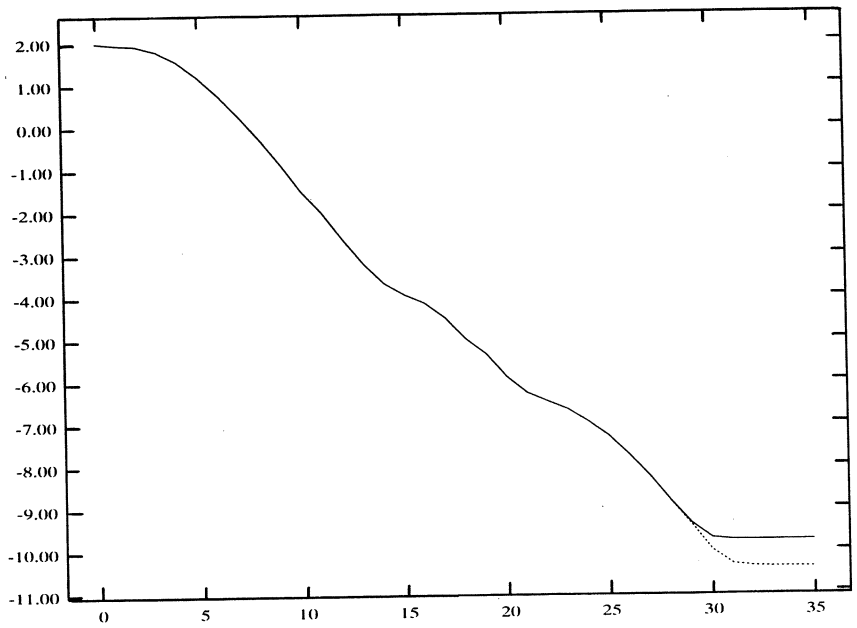

Figure 3. Log_{10} of the true residual norms versus the number of iterations; $c = d = 10$, fast Poisson preconditioning. Solid curve: Algorithm 2.1; dotted curve: the usual Gram–Schmidt GMRES(m) implementation

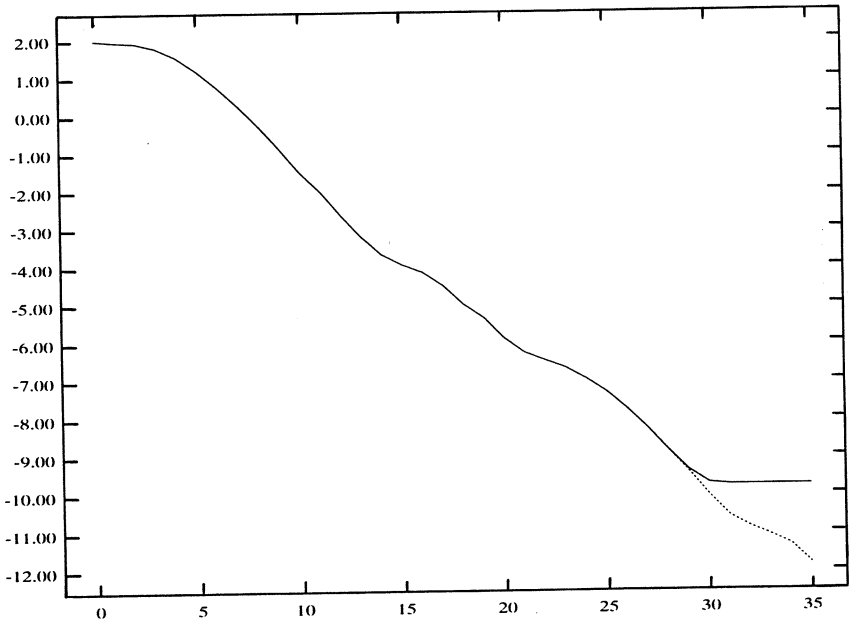

Figure 4. Log_{10} of the true and recursive residual norms versus the number of iterations for Algorithm 2.1; $c = d = 10$, fast Poisson preconditioning. Solid curve: true residual norms; dotted curve: recursive residual norms

Acknowledgements

The authors wish to express appreciation to the referees, whose comments instigated major additions to the original work. We also thank Steve Ashby and Wayne Joubert for helpful discussions.

This work was supported in part by United States Air Force Office of Scientific Research grant AFOSR-91-0294, United States Department of Energy grants DE-FG02-92ER25136 and DE-FG03-94ER25221, and National Science Foundation grant DMS-9400217, all with Utah State University.

REFERENCES

1. Y. Saad and M. H. Schultz. GMRES: a generalized minimal residual method for solving non-symmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7, 856–869, 1986.
2. P. N. Swarztrauber and R. A. Sweet. Efficient FORTRAN subprograms for the solution of elliptic partial differential equations. *ACM Trans. Math. Soft.*, 5, 352–364, 1979.
3. K. Turner and H. F. Walker. Efficient high accuracy solutions with GMRES(m). *SIAM J. Sci. Stat. Comput.*, 13, 815–825, 1992.
4. H. A. Van der Vorst and C. Vuik. GMRESR: a family of nested GMRES methods. Technical Report 91–80, Faculty of Technical Mathematics and Informatics, Delft University of Technology, 1991.
5. H. F. Walker. Implementation of the GMRES method using Householder transformations. *SIAM J. Sci. Stat. Comput.*, 9, 52–163, 1988.
6. H. F. Walker. Implementations of the GMRES method. *Computer Physics Communication*, 53, 311–320, 1989.
7. D. M. Young and K. C. Jea. Generalized conjugate-gradient acceleration of nonsymmetrizable iterative methods. *J. Lin. Alg. Appl.*, 34, 159–194, 1980.