Computer Physics Communications 53 (1989) 311–320
North-Holland, Amsterdam

IMPLEMENTATIONS OF THE GMRES METHOD

Homer F. WALKER [1]

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

Received 23 August 1988

We discuss general and specific implementations of the GMRES method for solving large nonsymmetric linear systems. Particular attention is given to issues of numerical reliability and efficiency. Computational results are reviewed.

1. Introduction

The generalized minimal residual (GMRES) method is an iterative method introduced by Saad and Schultz [10] for solving large linear systems of equations

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

in which A is assumed to be $n \times n$ and nonsingular but to have no special properties such as symmetry. The GMRES method begins with an initial approximate solution x_0 and initial residual $r_0 = b - Ax_0$, and at the mth iteration, a correction z_m is determined in the *Krylov subspace*

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

with $v = r_0$ which solves the least-squares problem

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

where $\| \cdot \|_2$ denotes the Euclidean norm. The mth iterate is then $x_m = x_0 + z_m$. The residual norm is clearly nonincreasing from one iteration to the next, and if exact arithmetic were used, then the solution would be reached in no more than n iterations (ref. [10], corollary 3, p. 865). However, the residual norm need not be strictly decreasing and may decrease only very slowly; see the example of Brown [2].

A natural approach to determining z_m is to determine a basis of $\mathcal{K}_m(r_0)$ and then to solve an m-dimensional least-squares problem for the coefficients of the linear combination of basis elements that solves (1.2). The manner in which this is carried out is critical to the efficiency and reliability of the method, and the object of this paper is to discuss various ways of doing this and their merits.

This paper is predominantly expository in nature. Although there is some new material, the novelty lies mainly in the presentation. We have drawn heavily on other papers, especially refs. [10,13]; however, this is not intended to be a thorough survey.

In section 2, we first formulate a general GMRES implementation and then discuss how specific implementations can be derived from it. Our feeling is that most discussions of the GMRES method are so implementation-specific that the basic workings of the method are obscured, and we hope to make these clear. In section 3, we give detailed outlines of several specific implementations, discuss their storage and arithmetic requirements, and give the results of computational experiments which reflect their numerical properties.

The GMRES method was introduced by Saad

[1] This work was supported by United States Department of Energy Grant Number DE-FG02-86ER25018, Department of Defense/Army Grant Number DAAL03-88-K, National Science Foundation Grant Number DMS-0088995, all with Utah State University, and the Computation and Mathematics Research Division, Lawrence Livermore National Laboratory.

0010-4655/89/$03.50 © Elsevier Science Publishers B.V.
(North-Holland Physics Publishing Division)

and Schultz [10] as a generalization of the Paige and Saunders [9] MINRES algorithm for solving indefinite symmetric systems. It is mathematically equivalent to the generalized conjugate residual (GCR) method [5,11] and to the ORTHODIR method of Jea and Young [8]. However, GCR may break down if A is not positive real and ORTHODIR may be less numerically stable. Also, GMRES requires only about half of the storage and 2/3 of the arithmetic of those methods. See ref. [10] for a more detailed comparison. There are also many methods which are related to GMRES but not equivalent to it, all of which have the conjugate gradient method (see, e.g., ref. [6]) as at least a spiritual ancestor. These are too numerous to discuss here; see Dennis and Turner [4] for a brief review of these methods and a unifying framework for deriving and analyzing them.

Notational conventions are as follows: For positive integers p and q, $\mathbf{R}^p$ denotes p-dimensional real Euclidean space and $\mathbf{R}^{p\times q}$ denotes the space of real $p\times q$ matrices. We assume that in (1.1), A is in $\mathbf{R}^{n\times n}$ and x and b are in $\mathbf{R}^n$, although extension to the complex case requires only trivial changes in the following. Matrices are denoted by capital letters; vectors and scalars are denoted by lower case letters. Vector components and matrix entries are indicated by superscripts in parentheses, e.g., $v^{(i)}$ denotes the ith component of the vector v; subscripts are used to indicate some distinguishing or identifying property. The ith canonical basis vector, i.e., the ith column of the identity matrix I, is denoted by e_i. With or without subscripts or other distinguishing marks, the letters H, J, L, P and R always (unless explicitly noted otherwise) indicate matrices of the following respective types: upper-Hessenberg, Givens rotation, lower-triangular, Householder transformation and upper-triangular. (See Golub and Van Loan [6] for definitions and properties and also as a general linear algebra reference.) Also, the letter Q always (unless explicitly noted otherwise) indicates a matrix with orthonormal columns (an *orthogonal matrix* in the square-matrix case), so that $Q^TQ = I$. Special additional conventions for Givens rotations and Householder transformations are the following: For convenience we allow the identity matrix I to be considered both a Givens rotation and a Householder transformation. A Givens rotation J_i subscripted with a positive integer i only "rotates" components i and $i+1$ of vectors on which it acts. A similarly subscripted Householder transformation P_i only "transforms" the ith and subsequent components of vectors on which it acts. This is to say that if $P_i = I - 2u_iu_i^T$ for a vector u_i of unit length, which we refer to as the *Householder vector* determining P_i, then all components of u_i before the ith are zero. Dimensions of vectors and matrices and, when appropriate, their (possibly) nonzero elements are often implicit from the contexts in which they appear, in which case we usually do not explicitly point them out. For example, if R is $p\times q$ upper-triangular and we write $R = [ce_1, H, h]$, then H must be $p\times(q-2)$ upper-Hessenberg and h must be a p-vector with zero components after the qth.

2. Implementing the GMRES method

We begin by outlining how the least-squares problem (1.2) can be solved once a basis of $\mathcal{K}_m(r_0)$ has been determined. We assume that the dimension of $\mathcal{K}_m(r_0)$ is m, for otherwise GMRES would have already arrived at the solution in fewer than m iterations. This follows from

Proposition 2.1. If dimension $\mathcal{K}_m(r_0)$ = dimension $\mathcal{K}_{m+1}(r_0) = m$ *for some* m, *then* $\min_{z\in\mathcal{K}_m(r_0)} \|b - A(x_0+z)\|_2 = 0$.

Proof. Writing $z\in\mathcal{K}_m(r_0)$ as $z = [r_0, Ar_0, \ldots, A^{m-1}r_0]y$ for $y\in\mathbf{R}^m$, one has

$$\min_{z\in\mathcal{K}_m(r_0)} \|b - A(x_0+z)\|_2 = \min_{y\in\mathbf{R}^m} \|r_0 - [Ar_0, A^2r_0, \ldots, A^mr_0]y\|_2.$$

Since $\{r_0, Ar_0, \ldots, A^{m-1}r_0\}$ is linearly independent, so is $\{Ar_0, A^2r_0, \ldots, A^mr_0\}$. Since $\{r_0, Ar_0, \ldots, A^mr_0\}$ is linearly dependent, it follows that $r_0 = [Ar_0, A^2r_0, \ldots, A^mr_0]y$ for some $y\in\mathbf{R}^m$. □

Suppose $\{v_1, \ldots, v_m\}$ is a basis of $\mathcal{K}_m(r_0)$. We assume (reasonably) that $v_1 = r_0/\beta$, where $\beta =$

$\pm \| r_0 \|_2$. Writing $z \in \mathcal{K}_m(r_0)$ as $z = [v_1, \ldots, v_m]y$ for $y \in \mathbf{R}^m$ gives

$$\begin{aligned} &\| b - A(x_0 + z) \|_2 \\ &\quad = \| r_0 - Az \|_2 = \| r_0 - [Av_1, \ldots, Av_m]y \|_2 \\ &\quad = \left\| [v_1, Av_1, \ldots, Av_m] \begin{bmatrix} \beta \\ -y \end{bmatrix} \right\|_2 . \end{aligned} \tag{2.1}$$

To solve the least-squares problem (1.2), we minimize the last term on the right over $y \in \mathbf{R}^m$. This can be done effectively with a factorization $[v_1, Av_1, \ldots, Av_m] = Q_m R_m$. In the following, Q_m is either $n \times (m+1)$ or $n \times n$, with R_m dimensioned accordingly. Note that the first column of Q_m is necessarily $\pm v_1$, so $R_m = [\pm e_1, H_m]$. Note that H_m is of full rank m, since $[Av_1, \ldots, Av_m] = Q_m H_m$ and the linear independence of $\{v_1, \ldots, v_m\}$ implies that of $\{Av_1, \ldots, Av_m\}$. Then

$$\begin{aligned} &\| b - A(x_0 + z) \|_2 \\ &\quad = \left\| Q_m R_m \begin{bmatrix} \beta \\ -y \end{bmatrix} \right\|_2 = \left\| R_m \begin{bmatrix} \beta \\ -y \end{bmatrix} \right\|_2 \\ &\quad = \| \pm \beta e_1 - H_m y \|_2 , \end{aligned}$$

and it remains to solve a full-rank upper-Hessenberg least-squares problem. Solving this problem using Givens rotations is easy, numerically sound, and also allows a very important feature of a practical GMRES implementation: *One can evaluate* $\min_{y \in \mathbf{R}^m} \| \pm \beta e_1 - H_m y \|_2$ *without having to solve for* y (ref. [10], proposition 1, p. 862). To see this, suppose we have $J_1, \ldots, J_m$ such that $J_m \ldots J_1 H_m = \bar{R}_m$. Then $\| \pm \beta e_1 - H_m y \|_2 = \| \pm w_m - \bar{R}_m y \|_2$, where $w_m = J_m \ldots J_1 \beta e_1$. Since $\bar{R}_m$ has full rank m, this has minimal value $| w_m^{(m+1)} |$. The minimizing y is found by solving a nonsingular triangular system, but this need not actually be done at the mth GMRES iteration unless the minimal residual given by $| w_m^{(m+1)} |$ is sufficiently small. We caution that as the limits of residual reduction are reached in finite-precision arithmetic, $| w_m^{(m+1)} |$ may not provide an accurate measure of the minimal residual; see ref. [13] and the numerical experiments in section 3.

One sees that solving the least-squares problem (1.2) in this way requires increasing amounts of storage as m grows; indeed, in the implementations considered here, the major storage requirement is exactly one additional vector of length n at each iteration. In practice, then, one usually imposes a maximum number of iterations, typically much less than n, and if this number is reached without a sufficient residual reduction having been achieved, then the latest approximate solution is formed and the algorithm is restarted with this as the new initial approximate solution. We note that for such a "restarted" algorithm, it is possible for the successive residual norms, although nonincreasing, not to converge to zero or indeed never to decrease on some problems; see the discussion in ref. [10], p. 865.

This discussion suggests the following general GMRES implementation.

General implementation. Suppose x_0, TOL and MAXIT are given.

1. Initialize: Compute $r_0 = b - Ax_0$ and set $\beta = \pm \| r_0 \|_2$ and $v_1 = r_0/\beta$.
2. Iterate: For $m = 1, 2, \ldots,$ MAXIT, do:
   a. Evaluate Av_m.
   b. Factor $[v_1, Av_1, \ldots, Av_m] = Q_m R_m$, $R_m = [\pm e_1, H_m]$.
   c. Factor $J_m \ldots J_1 H_m = \bar{R}_m$, and form $w_m = J_m \ldots J_1 \beta e_1$.
   d. If $| w_m^{(m+1)} | \leq$ TOL, then go to 3; otherwise, determine v_{m+1} so that $\{v_1, \ldots, v_{m+1}\}$ is a basis of $\mathcal{K}_{m+1}(r_0)$ and increment m.
3. Solve: Determine y_m which minimizes $\| \pm w_m - \bar{R}_m y \|_2$ and overwrite $x_0 \leftarrow x_0 + [v_1, \ldots, v_m]y_m$. If $| w_m^{(m+1)} | \leq$ TOL, accept x_0 as the final approximate solution; otherwise, return to 1.

Remarks. In practice, as in the specific implementations discussed here, step 2.b is carried out in only $\mathcal{O}(mn)$ arithmetic operations per iteration by updating the factorization $[v_1, Av_1, \ldots, Av_{m-1}] = Q_{m-1} R_{m-1}$. Similarly, step 2.c is carried out in $\mathcal{O}(m)$ arithmetic operations per iteration using $J_{m-1} \ldots J_1 H_{m-1} = \bar{R}_{m-1}$ and $w_{m-1} = J_{m-1} \ldots J_1 \beta e_1$.

Specific implementations are obtained from the General implementation by specifying particular ways of doing the factorization in step 2.b and of determining v_{m+1} in step 2.d. We outline two

effective ways of carrying out the factorization in step 2.d which use only $\mathcal{O}(mn)$ arithmetic operations per iteration. The first uses modified Gram–Schmidt orthogonalization (ref. [6], p. 152); in it, $Q_m \in \mathbf{R}^{n\times(m+1)}$ and $R_m \in \mathbf{R}^{(m+1)\times(m+1)}$ for each m. The second uses Householder transformations; in it, $Q_m \in \mathbf{R}^{n\times n}$ and $R_m \in \mathbf{R}^{n\times(m+1)}$ for each m. Q_m itself is not formed or stored; rather it is determined implicitly as a product of $(m+1)$ Householder transformations, and the $(m+1)$ Householder vectors determining these transformations are stored.

Gram–Schmidt factorization. Set $q_1 = v_1$ and $Q_0 = [q_1]$, $R_0 = [1]$. For $m \geq 1$, suppose Av_m, $Q_{m-1} = [q_1, \ldots, q_m]$ and R_{m-1} are given. Then in step 2.b, do:

i. Set $q_{m+1} = Av_m$.
ii. For $i = 1, \ldots, m$, do:
  Set $r^{(i,m+1)} = q_i^T q_{m+1}$.
  Overwrite $q_{m+1} \leftarrow q_{m+1} - r^{(i,m+1)} q_i$.
iii. Set $r^{(m+1,m+1)} = \| q_{m+1} \|_2$.
iv. If $r^{(m+1,m+1)} \neq 0$, overwrite $q_{m+1} \leftarrow q_{m+1}/r^{(m+1,m+1)}$.
v. Set $Q_m = [Q_{m-1}, q_{m+1}]$,

$$R_m = \begin{bmatrix} R_{m-1} & r^{(1,m+1)} \\ & \vdots \\ 0 \ldots 0 & r^{(m+1,m+1)} \end{bmatrix}.$$

Remarks. If for some m $q_{m+1} = 0$ following step ii, then $Q_m = [Q_{m-1}, 0]$, in violation of our usual convention that "Q" connotes orthonormal columns. However, the lower right-hand corner element of R_m is $r^{(m+1,m+1)} = 0$, which means that in the General implementation $J_m = I$ in step 2.c and $|w_m^{(m+1)}| = 0$ in steps 2.d and 3. Then (in exact arithmetic) the exact solution is obtained at this iteration and q_{m+1} plays no further role (cf. ref. [10], proposition 2, p. 865).

Householder factorization. Choose P_1 such that $P_1 v_1 = \pm e_1$ and set $R_0 = [\pm e_1]$. For $m \geq 1$, suppose Av_m, $P_1, \ldots, P_m$ and R_{m-1} are given. Then in step 2.b, do:

i. Form $r_{m+1} = P_m \ldots P_1 Av_m$.
ii. Choose P_{m+1} such that $P_{m+1} r_{m+1}$ has only zero components below the $(m+1)$st, and overwrite $r_{m+1} \leftarrow P_{m+1} r_{m+1}$.
iii. Set $R_m = [R_{m-1}, r_{m+1}]$.

Remarks. For numerical soundness, the sign of $\pm e_1$ should be chosen opposite that of $v_1^{(1)}$. After step iii one implicitly has the desired factors Q_m and R_m with $Q_m = P_1 \ldots P_{m+1}$. If r_{m+1} found in step i has only zero components below the $(m+1)$st, then one chooses $P_{m+1} = I$. If r_{m+1} has only zero components below the mth, then one also chooses $J_m = I$ in step 2.c of the General implementation. In this case $|w_m^{(m+1)}| = 0$ in steps 2.d and 3, and (in exact arithmetic) the exact solution is obtained at this iteration.

The relative merits of these two factorizations can be summed up by saying that the Householder factorization is somewhat more numerically reliable but requires somewhat more arithmetic than the Gram–Schmidt factorization. Regarding reliability, we note that results of Björck [1] imply the following: If $S = [v_1, Av_1, \ldots, Av_m]$ and if Q is Q_m computed by the Gram–Schmidt factorization using floating point arithmetic with unit rounding error $\mathbf{u}$, then

$$Q^T Q = I + E, \quad \| E \|_2 \approx \mathbf{u} \kappa_2(S), \tag{2.2}$$

where the condition number $\kappa_2(S)$ is the ratio of the largest singular value of S to the smallest; however, if Q consists of the first $m+1$ columns of $P_1 \ldots P_{m+1}$ computed by the Householder factorization, then

$$Q^T Q = I + E, \quad \| E \|_2 \approx \mathbf{u}. \tag{2.3}$$

This suggests that if $\{v_1, Av_1, \ldots, Av_m\}$ is "nearly" linearly dependent, then the Gram–Schmidt factorization may be considerably less reliable than the Householder factorization. An indication of the practical significance of this is seen in the numerical results reported in section 3. Regarding arithmetic, one might think on the basis of the above formulations that the two factorizations require about the same amount of computational effort. This is almost certain not to be the case in practice, however, if one considers the amounts of arithmetic required overall by imple-

mentations using the two factorizations. We defer a detailed discussion of arithmetic requirements until more specific algorithms have been formulated in section 3; for now, we note that use of the Householder factorization is likely to require additional computation in the determination of v_{m+1} in step 2.d of the General implementation and perhaps in the determination of $[v_1,\dots,v_m]y_m$ in step 3 as well.

We now consider the issue of determining v_{m+1} in step 2.d of the General implementation. Perhaps the most obvious choice is $v_{m+1} = Av_m = A^m v_1$ for each m, so that $\{v_1,\dots,v_m\}$ is (up to scalar multiplication) the "natural" basis $\{r_0, Ar_0,\dots, A^{m-1}r_0\}$ of $\mathscr{K}_m(r_0)$ for each m. This choice is not recommended. A minor difficulty is that $\|A^m v_1\|_2$ is likely to grow rapidly with m; this is easily resolved by renormalization. The major difficulty is that as m grows, $\{v_1, Av_1,\dots, Av_m\}$ is likely to become very "nearly" linearly dependent very quickly. This means that the least-squares problem

$$\min_{y\in\mathbf{R}^m}\left\|[v_1, Av_1,\dots, Av_m]\begin{bmatrix}\beta\\ -y\end{bmatrix}\right\|_2 \tag{2.4}$$

from (2.1) is likely to become very ill-conditioned very quickly, in which case (2.4) cannot be solved accurately for y_m regardless of the stability of the factorization method used. The seriousness of this difficulty is seen in the numerical experiments in section 3; see also Hindmarsh and Walker [7] for results of stiff ODE solving experiments.

A very satisfactory choice of v_{m+1} for each m is the $(m+1)$st column of Q_m. This choice implies that for each m, $v_1,\dots,v_m$ are (up to sign) the same as the first m columns of Q_m and therefore orthonormal. Furthermore, $\{v_1,\dots,v_m\}$ generated via this choice must be a basis of $\mathscr{K}_m(r_0)$ for each m. To see this, note that in the General implementation, one needs to determine v_{m+1} for some m only if R_m is of full rank $m+1$. Indeed, if R_m fails to be of full rank, then one would have $J_m = I$ in step 2.c and $|w_m^{(m+1)}| = 0$ in step 2.d, and the exact solution would be reached (in exact arithmetic) at this iteration. This if v_{m+1} has been determined by this choice, then one has

$$\begin{aligned}[v_1, Av_1,\dots, Av_m] &= Q_m R_m\\ &= [v_1, v_2,\dots, v_{m+1}]\tilde{R}_m,\end{aligned}$$

where $\tilde{R}_m \in \mathbf{R}^{(m+1)\times(m+1)}$ is nonsingular and is obtained by multiplying each column of R_m by ± 1 and truncating after the first $m+1$ rows if $R_m \in \mathbf{R}^{n\times(m+1)}$. Then the following proposition implies that each $\{v_1,\dots,v_m\}$ generated with this choice is a basis of $\mathscr{K}_m(r_0)$.

Proposition 2.2. If $\{v_1,\dots,v_m\}$ is linearly independent and for $k = 1,\dots,m-1$,

$$[v_1, Av_1,\dots, Av_k] = [v_1,\dots, v_{k+1}]M_k, \tag{2.5}$$

where $M_k \in \mathbf{R}^{(k+1)\times(k+1)}$ is invertible, then $\{v_1,\dots,v_k\}$ is a basis of $\mathscr{K}_k(v_1)$ for $k = 1,\dots,m$.

Proof. Certainly $\{v_1\}$ is a basis of $\mathscr{K}_1(v_1)$. Suppose $\{v_1,\dots,v_k\}$ is a basis of $\mathscr{K}_k(v_1)$ for some $k \geq 1$. Then span $\{v_1,\dots,v_k\} = \text{span}\{v_1, Av_1,\dots, A^{k-1}v_1\}$, which implies $\text{span}\{Av_1,\dots, Av_k\} = \text{span}\{Av_1, A^2v_1,\dots, A^kv_1\}$ and $\text{span}\{v_1, Av_1,\dots, Av_k\} = \text{span}\{v_1, Av_1,\dots, A^kv_1\} = \mathscr{K}_{k+1}(v_1)$. It follows from (2.5) and the assumption of linear independence that $\{v_1,\dots,v_{k+1}\}$ is a basis of $\mathscr{K}_{k+1}(v_1)$, and the induction is complete. □

3. Detailed implementations and numerical experiments

Here, we give in detail three implementations of the GMRES method derived from the framework outlined in section 2. We note that neither the notation used below nor the outlines of the implementations correspond exactly to their counterparts in section 2 but are intended to reflect how they might be coded up and to allow a clear assessment of the arithmetic, storage, etc., required by each implementation. In all of these implementations, the Krylov subspace basis vectors are chosen in the "very satisfactory" way given in section 2, i.e., corresponding to choosing each v_{m+1} in step 2.d of the General implementation to be the $(m+1)$st column of Q_m in step 2.b. Following the outlines of the implementations, we give the results of some numerical experiments.

The first implementation uses Gram–Schmidt factorization; this is essentially the original GMRES implementation of ref. [10].

Gram–Schmidt implementation. Suppose x_0, TOL and MAXIT are given.

1. Initialize: Compute $r_0 = b - Ax_0$ and set $v_1 = r_0/\| r_0 \|_2$ and $w = \| r_0 \|_2 e_1 \in \mathbf{R}^{\mathrm{MAXIT}+1}$.
2. Iterate: For $m = 1, 2, \ldots,$ MAXIT, do:
   a. Evaluate $v_{m+1} = Av_m$.
   b. For $i = 1, \ldots, m$, do:
      i. Set $r^{(i)} = v_i^{\mathrm{T}} v_{m+1}$.
      ii. Overwrite $v_{m+1} \leftarrow v_{m+1} - r^{(i)} v_i$.
   c. Set $r^{(m+1)} = \| v_{m+1} \|_2$.
   d. If $r^{(m+1)} \neq 0$, overwrite $v_{m+1} \leftarrow v_{m+1}/r^{(m+1)}$.
   e. Set $r = [r^{(1)}, \ldots, r^{(m+1)}, 0, \ldots, 0]^{\mathrm{T}} \in \mathbf{R}^{\mathrm{MAXIT}+1}$.
   f. If $m > 1$, overwrite $r \leftarrow J_{m-1} \ldots J_1 r$.
   g. If $r^{(m+1)} \neq 0$, find J_m such that $(J_m r)^{(m+1)} = 0$; otherwise, go to i.
   h. Overwrite $r \leftarrow J_m r$ and $w \leftarrow J_m w$.
   i. Set

$$R_m = \begin{cases} [r], & \text{if } m = 1; \\ [R_{m-1}, r], & \text{if } m > 1. \end{cases}$$

   j. If $|w^{(m+1)}| < \mathrm{TOL}$, then go to 3; otherwise, increment m.
3. Solve:
   a. Determine y_m by solving a nonsingular upper-triangular system with the first m rows of R_m as the coefficient matrix and the first m components of w as the right-hand side.
   b. Overwrite $x_0 \leftarrow x_0 + [v_1, \ldots, v_m] y_m$.
   c. If $|w^{(m+1)}| < \mathrm{TOL}$, accept x_0 as the final approximate solution; otherwise, return to 1.

We mention a variant of the Gram–Schmidt implementation called the Gram–Schmidt implementation with reorthogonalization. Here we consider this only in its most basic form, in which one simply repeats step 2.b, adding the new $r^{(i)}$ values to those previously generated. For a detailed discussion of the Gram–Schmidt process with (possibly repeated) reorthogonalization, see ref. [3]. The object of reorthogonalization is to make the resulting v_{m+1} as nearly orthogonal to $v_1, \ldots, v_m$ as possible and thereby improve the numerical properties of the method. A possibility is to reorthogonalize only when $r^{(m+1)} = \| v_{m+1} \|_2$ in step 2.c is found to be below some minimum value, thus saving reorthogonalizations which are unlikely to be necessary. When referring to the Gram–Schmidt implementation with reorthogonalization below, though, we assume reorthogonalization is done exactly once for every m.

The remaining two implementations use Householder factorization. These implementations are essentially algorithms 2.2 and 3.1 of ref. [13]. The first is obtained in a fairly straightforward way from the discussion in section 2. The second is obtained from the first by expanding products of Householder transformations as sums. See ref. [13] for details of the derivations, especially that of the second implementation. The second implementation was given in ref. [13] as an alternative to the first which may offer advantages in some circumstances on parallel computers. We observe below that it actually requires less computational effort as well.

First Householder implementation. Suppose x_0, TOL and MAXIT are given.

1. Initialize: Compute $r_0 = b - Ax_0$ and determine P_1 such that $P_1 r_0 = \pm \| r_0 \|_2 e_1 \equiv w$.
2. Iterate: For $m = 1, 2, \ldots,$ MAXIT, do:
   a. Evaluate $r \equiv P_m \ldots P_1 A P_1 \ldots P_m e_m$.
   b. If $r^{(m+1)} = \cdots = r^{(n)} = 0$, then go to e.
   c. Determine P_{m+1} such that $P_{m+1} r$ has zero components after the $(m+1)$st.
   d. Overwrite $r \leftarrow P_{m+1} r$.
   e. If $m > 1$, overwrite $r \leftarrow J_{m-1} \ldots J_1 r$.
   f. If $r^{(m+1)} \neq 0$, find J_m such that $(J_m r)^{(m+1)} = 0$; otherwise, go to h.
   g. Overwrite $r \leftarrow J_m r$ and $w \leftarrow J_m w$.
   h. Set

$$R_m = \begin{cases} [r], & \text{if } m = 1; \\ [R_{m-1}, r], & \text{if } m > 1. \end{cases}$$

   i. If $|w^{(m+1)}| < \mathrm{TOL}$, then go to 3; otherwise, increment m.
3. Solve:
   a. Determine y_m by solving a nonsingular upper-triangular system with the first m rows of R_m as the coefficient matrix and the first m components of w as the right-hand side.
   b. For $k = 1, \ldots, m$, do:
      Overwrite $x_0 \leftarrow x_0 + y_m^{(k)} P_1 \ldots P_k e_k$.
   c. If $|w^{(m+1)}| < \mathrm{TOL}$, accept x_0 as the final approximate solution; otherwise, return to 1.

Second Householder implementation. Suppose x_0, TOL, and MAXIT are given.

1. Initialize: Compute $r_0 = b - Ax_0$, determine $P_1 = I - 2u_1u_1^T$ such that $P_1r_0 = \pm \| r_0 \|_2 e_1 \equiv w$, and set $U_1 = [u_1]$ and $L_1 = [1]$.
2. Iterate: For $m = 1, 2, \ldots,$ MAXIT, do:
   a. Evaluate $r \equiv \{I - 2U_mL_m^{-1}U_m^T\}A\{I - 2U_m(L_m^T)^{-1}U_m^T\}e_m$.
   b. If $r^{(m+1)} = \cdots = r^{(n)} = 0$, then go to e.
   c. Determine $P_{m+1} = I - 2u_{m+1}u_{m+1}^T$ such that $P_{m+1}r$ has zero components after the $(m+1)$st.
   d. Overwrite $r \leftarrow P_{m+1}r$.
   e. If $m > 1$, overwrite $r \leftarrow J_{m-1} \ldots J_1 r$.
   f. If $r^{(m+1)} \neq 0$, find J_m such that $(J_m r)^{(m+1)} = 0$; otherwise, go to h.
   g. Overwrite $r \leftarrow J_m r$ and $w \leftarrow J_m w$.
   h. Set
$$R_m = \begin{cases} [r], & \text{if } m = 1; \\ [R_{m-1}, r], & \text{if } m > 1. \end{cases}$$
   i. If $|w^{(m+1)}| \leq$ TOL, then go to 3; otherwise, continue.
   j. Set $U_{m+1} = [U_m, u_{m+1}]$, evaluate
$$L_{m+1} = \begin{bmatrix} L_m & 0 \\ 2u_{m+1}^T U_m & 1 \end{bmatrix},$$
   and increment m.
3. Solve:
   a. Determine y_m by solving a nonsingular upper-triangular system with the first m rows of R_m as the coefficient matrix and the first m components of w as the right-hand side.
   b. Overwrite $x_0 \leftarrow x_0 + \{I - 2U_m(L_m^T)^{-1}U_m^T\}[e_1, \ldots, e_m]y_m$.
   c. If $|w^{(m+1)}| <$ TOL, accept x_0 as the final approximate solution; otherwise, return to 1.

In comparing these implementations, we first consider storage and arithmetic. In addition to storage for x_0, incidental constants, etc., the Gram–Schmidt implementation requires an array of dimension (MAXIT + 1) × n to store the orthonormal basis vectors $\{v_1, \ldots, v_{m+1}\}$ together with MAXIT(MAXIT + 1)/2 locations for the nonzero elements of R_m and 2 or 3 times MAXIT locations for w and $J_1, \ldots, J_m$. The First Householder implementation requires less storage, since the Householder vectors are stored instead of the basis vectors. These are of decreasing length, and so the nonzero elements of R_m can be stored in the (MAXIT + 1) × n array containing the Householder vectors. The second Householder implementation requires the same storage as the Gram–Schmidt implementation, since it requires storage of L_m as well as R_m. Thus, the First Householder implementation has a storage advantage of MAXIT(MAXIT + 1)/2 locations over the other two implementations. This is unlikely to be of major importance, since MAXIT $\ll n$ in most applications.

Table 1
$\mathcal{O}(n)$ multiplication counts

Method	At the mth iteration	After m iterations and solution
GS	$2(m+1)n$	$(m^2+4m+2)n$
GSR	$2(2m+1)n$	$(2m^2+5m+2)n$
H1	$4mn$	$(3m^2+2m+2)n$
H2	$2(2m+1)n$	$(2m^2+5m+2)n$

GS = Gram–Schmidt implementation
GSR = Gram–Schmidt implementation with Reorthogonalization
H1 = First Householder implementation
H2 = Second Householder implementation

Each implementation requires the same number of evaluations of products of A with vectors, viz., one per iteration. To compare the arithmetic required by the implementations in addition to these A-products, we counted numbers of multiplications, ignoring terms of less than $\mathcal{O}(n)$. The results are given in table 1, in which we include counts for the Gram–Schmidt implementation with reorthogonalization for completeness. These counts are straightforward to obtain; see the discussion of ref. [13], p. 158. One sees from these counts that the Gram–Schmidt implementation requires the least arithmetic, the Second Householder implementation and the Gram–Schmidt implementation with reorthogonalization require about the same arithmetic, which is about twice that of the Gram–Schmidt implementation, and the first Householder implementation requires the

most arithmetic, about three times that of the Gram–Schmidt implementation. It is interesting to note that the First Householder implementation requires slightly less arithmetic during the iterations than the Second Householder implementation and the Gram–Schmidt implementation with reorthogonalization but that it loses its advantage in the solution stage.

We now consider the numerical reliability of these implementations. The results of Björck [1] mentioned in section 2 (see (2.2) and (2.3)) suggest that the Householder implementations are somewhat more numerically sound than the Gram–Schmidt implementation. We have conducted a number of numerical experiments with these implementations, and our feelings are these: If a moderate amount of residual reduction is desired, then all implementations are about equally reliable, especially if the system is fairly well-conditioned. If one would like to obtain nearly the greatest residual reduction possible for a given working precision or if nearly all possible accuracy is desired, then the Householder implementations may offer some advantage over the Gram–Schmidt implementation, even with reorthogonalization, especially if the system is ill-conditioned. Also, as the limits of residual reduction are reached, the measure of the residual given by $|w^{(m+1)}|$ in the detailed implementations is likely to be inaccurate for all implementations but somewhat less so for the Householder implementations. We have seen no evidence that either Householder implementation is numerically superior to the other, and so our preference is for the Second since it requires less arithmetic. We conclude by briefly describing some of the numerical experiments on which these opinions are based.

Experiment 1. This is the first experiment considered in ref. [13]. The problem to solve is a nonsymmetric linear system arising from the discretization of the boundary value problem

$$\Delta u + cu + d\frac{\partial u}{\partial x} = f \quad \text{in } D,$$
$$u = 0 \quad \text{on } \partial D,$$

where $D = [0, 1] \times [0, 1]$ and $c \geq 0$ and d are constants. As reported in ref. [13], a number of trials were run with the different implementations with $f(x) \equiv 1$, a 100×100 mesh of discretization points in D, a variety of values of c and d ranging from 1 to 1000, and MAXIT ranging from 5 to 50. No preconditioning was used, and computing was done in single precision on a Digital Equipment Corporation Micro VAX II running Ultrix and using the f77 Fortran compiler. Plots were made of the logarithm of the residual norm versus the number of iterations, with the residual norms obtained "from scratch" rather than from the values of $|w^{(m+1)}|$. Typical results for the Gram–Schmidt implementation and the First Householder implementation are given in fig. 1 of ref. [13], in which $c = d = 100$ and MAXIT = 10. (The results for the Gram–Schmidt implementation with reorthogonalization and the Second Householder implementation are visually indistinguishable from the respective results for the Gram–Schmidt implementation and the First Householder implementation in this case.) Here we show results for $c = 1$, $d = 100$ and MAXIT = 20 in fig. 1. We have also included in fig. 1 the results for a GMRES implementation which uses Householder factorization in conjunction with the (not recommended) choice $v_{m+1} = Av_m = A^m v_1$ in step 2.d of the General implementation. This is referred to as the Naive implementation in fig. 1;

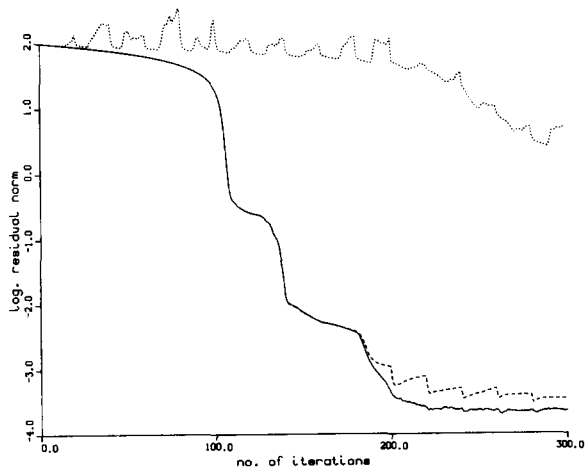

Fig. 1. The solid, dashed, and dotted lines are from the First Householder implementation, the Gram–Schmidt implementation, and the Naive implementation, respectively.

details may be found in ref. [12]. The very poor results shown in fig. 1 indicate why this implementation is not recommended.

Experiment 2. This is a stiff ODE – solving experiment reported as Test problem 3 in ref. [14]. The problem comes from a coupled system of reaction–diffusion equations in 1-D:

$$\frac{\partial c_1}{\partial t} = \frac{\partial^2 c_1}{\partial x^2} + R_1(c_1, c_2),$$

$$\frac{\partial c_2}{\partial t} = R_2(c_1, c_2),$$

$$R_1(c_1, c_2) = c_1[a(1 - c_1) - c_2]$$

$$R_2(c_1, c_2) = -c_2[a(1 - c_1) + c_2].$$

The domain is $-1 \le x \le 1$, $t \ge 0$, and $a > 0$ is a free parameter. The boundary conditions are Dirichlet with $c_i = 0$ at $x = \pm 1$. After discretizing in x, one has an ODE in which stiffness can arise from both the reaction and the diffusion rates. Solutions of this ODE asymptotically approach a steady-state solution at which the ODE Jacobian is very close to a matrix having a large-dimensional null-space.

In this experiment, the GMRES method was used to compute Newton steps in the corrector iterations in a backward differentiation formula method applied to the ODE. The iteration matrices were of the form $I - h\beta_0 J$, where h is the time step, β is a method constant and J is the ODE Jacobian. For large time, as h became large and the solution approached steady-state, these matrices became very ill-conditioned with many small singular values. Products of these matrices with vectors in the GMRES iterations were obtained both by approximating J-vector products with finite-differences and by analytic evaluation. The GMRES implementations used were the Gram–Schmidt implementation and the First Householder implementation. Computing was done on both a Cray at Lawrence Livermore National Laboratory and on a MicroVAX II at Utah State University. Complete details of the trials and the results are given in ref. [14]; we summarize the results here.

The First Householder implementation always showed greater run times than the Gram–Schmidt implementation, presumably reflecting the greater arithmetic that the First Householder implementation requires. However, when finite-difference J-vector products were used, the First Householder implementation often succeeded at reaching larger final times than the Gram–Schmidt implementation and often needed fewer time steps and function evaluations to obtain the final times successfully reached by the Gram–Schmidt implementation. When analytic evaluation of J-vector products was used, the performance of the two implementations was usually about the same, although on one Cray run the Gram–Schmidt implementation failed while the First Householder implementation succeeded. We note that on all failed runs, failure occurred because numerical error allowed the development of unstable ODE solution modes, the rapid growth of which brought about failure. Overall, the test results suggest that both implementations will fail when the combination of ill-conditioning and inaccuracy in J-vector products is sufficient to allow the development of unstable modes through numerical error; however, it appears that the First Householder implementation will succeed for somewhat greater levels of ill-conditioning and inaccuracy than the Gram–Schmidt implementation.

Experiment 3. This is the second experiment considered in ref. [13]. The object of this experiment is to assess the numerical soundness of the GMRES implementations which appear best suited to take advantage of parallel computing environments. The Gram–Schmidt implementation uses modified Gram–Schmidt orthogonalization in step 2.b, and the First Householder implementation applies products of Householder transformations to vectors in steps 2.a and 3.b. These are inherently sequential procedures which seem likely to inhibit the exploitation of parallelism in some circumstances. The Second Householder implementation, on the other hand, replaces the products of Householder transformations in the First Householder implementation with matrix–vector operations which can be carried out with a high degree of concurrency. The only apparent variation of the Gram–Schmidt implementation which similarly allows such a degree

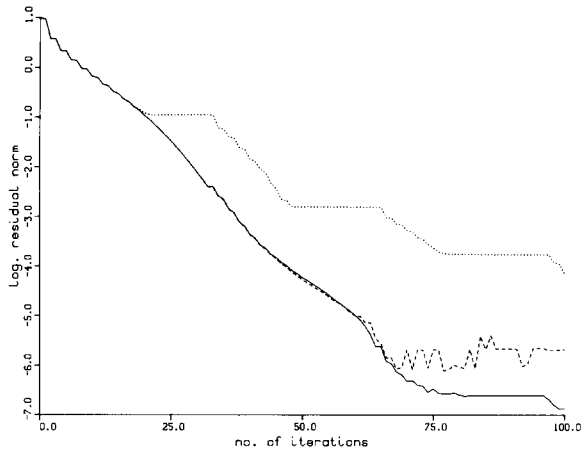

Fig. 2. The solid, dashed, and dotted lines are from the Second Householder implementation, the Gram–Schmidt implementation with reorthogonalization, and the Gram–Schmidt implementation (without reorthogonalization), respectively, with both Gram–Schmidt implementations using classical Gram–Schmidt orthogonalization.

of concurrency is that in which modified Gram–Schmidt orthogonalization is replaced with classical Gram–Schmidt orthogonalization, which is potentially unstable. (See ref. [6].) In this experiment, we compared the Second Householder implementation with Gram–Schmidt implementations using classical Gram–Schmidt orthogonalization, with and without reorthogonalization.

The system to be solved is $Ax = b$ with

$$A = \begin{pmatrix} 1 & 0 & \cdots & 0 & \alpha \\ 0 & 2 & \cdots & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & n \end{pmatrix} \text{ and } b = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}.$$

In our trials, we used various values of n and α and carried out the same procedures in the same computing environment as in experiment 1. Typical results are given in fig. 2 of ref. [13], in which $n = 100$, $\alpha = 2000$ and MAXIT = 32 and which we reproduce as fig. 2 for completeness. Note the flat areas on the dotted curve representing the performance of the Gram–Schmidt implementation using the classical Gram–Schmidt process without reorthogonalization. These apparently indicate that method's inability to further reduce the residual norm because of numerical error.

References

[1] Å. Björck, BIT 7 (1967) 1.
[2] P.N. Brown, Lawrence Livermore National Laboratory Report UCRL-98630 (1988).
[3] J.W. Daniel, W.B. Gragg, L. Kaufman and G.W. Stewart, Math. Comput. 30 (1976) 772.
[4] J.E. Dennis, Jr., and K. Turner, Lin. Alg. Appl. 88&89 (1987) 187.
[5] H.C. Elman, Ph. D. thesis, Computer Science Dept., Yale Univ., New Haven, CT (1982).
[6] G.H. Golub and C.F. Van Loan, Matrix Computations (Johns Hopkins Univ. Press, Baltimore, 1983).
[7] A.C. Hindmarsh and H.F. Walker, Lawrence Livermore Nat. Lab. Tech. Report UCID-20899 (1986).
[8] K.C. Jea and D.M. Young, Lin. Alg. Appl. 34 (1980) 159.
[9] C.C. Paige and M.A. Saunders, SIAM J. Num. Anal. 12 (1975) 617.
[10] Y. Saad and M.H. Schultz, SIAM J. Num. Anal. 7 (1986) 856.
[11] P.K.W. Vinsome, in: Proc. Fourth Symp. on Reservoir Simulation (Soc. Petroleum Engineers of AIME, 1976) p. 149.
[12] H.F. Walker, Lawrence Livermore Nat. Lab. Report UCRL-93589 (1985).
[13] H.F. Walker, SIAM J. Sci. Stat. Comput. 9 (1988) 152.
[14] H.F. Walker, Lawrence Livermore Nat. Lab. Report UCID-21343 (1988).