IMPLEMENTATION OF THE gmr ALGORITHM
FOR LARGE SYMMETRIC EIGENPROBLEMS*

Abstract. We present an implementation of the generalized minimal residual (gmr) algorithm for finding an eigenpair of a large symmetric matrix. We report some numerical results for this algorithm and compare them with the results obtained for the Lanczos algorithm. A Fortran implementation of the gmr algorithm can be obtained from the Institute of Computer Science of the Polish Academy of Sciences and is also available via anonymous FTP as "pub/gmrval" on Columbia.edu 128.59.16.1 on the Arpanet. The input of this subroutine is a matrix which has been partially reduced to a tridiagonal form. Such a form can be obtained by the Lanczos process.

1. Introduction. The usual procedure for finding an eigenpair of a large symmetric matrix $A$ is to approximate eigenpairs of $A$ from its behaviour in a given subspace of small dimension. The most popular method of this type is the Lanczos algorithm which gives approximations of eigenvectors in the Krylov subspace. It is known (see [4]) that the Lanczos algorithm does not produce an approximate eigenpair of $A$ with minimal residual. The generalized minimal residual algorithm (the gmr algorithm) was introduced in [3]. It finds the eigenpair with minimal residual in a Krylov subspace. The gmr algorithm enjoys certain theoretical optimality properties. The residuals of the gmr algorithm are never greater than the residuals of the Lanczos algorithm and sometimes they are much smaller. Since the cost of both algorithms is essentially the same, the gmr algorithm seems to be preferable.

In this paper we present an implementation of the gmr algorithm for real symmetric matrices. By applying $k$ steps of the Lanczos process, a symmetric

* This paper has been partly written while the author visited the Department of Computer Science, Columbia University. This research was supported in part by the National Science Foundation under Grant DCR-82-14322.
matrix $A$ is partially reduced to a tridiagonal form, i.e.

$$Q_k^T A Q_k = D_k,$$

where $Q_k$ is an $(n \times k)$-matrix with orthonormal columns and $D_k$ is a $(k+1) \times k$-tridiagonal matrix. We assume that coefficients of the matrix $D_k$ have been already computed.

The implementation was tested for many matrices. We report results for matrices with specifically chosen coefficients as well as for random matrices. Numerical tests confirm the theoretical advantages of the gmr algorithm over the Lanczos algorithm. For all matrices the computed residuals of the gmr algorithm are never greater than the corresponding residuals of the Lanczos algorithm and sometimes they are much smaller. The sequences of residuals generated by the gmr algorithm are always nonincreasing, while the sequences produced by the Lanczos algorithm do not enjoy this property. The Lanczos algorithm often increases significantly the residuals from one step to the next one.

For matrices with specifically chosen coefficients, the gmr algorithm is significantly more efficient than the Lanczos algorithm. For random matrices the gmr algorithm is only slightly better than the Lanczos algorithm.

2. The gmr algorithm. In this section we define the gmr algorithm and introduce some of its properties which are useful for implementation.

Let $A$ be an $n \times n$ real symmetric matrix. For a given vector $b \in \mathbb{R}^n$, $\|b\| = 1$ ($\|\cdot\| = \|\cdot\|_2$), consider the $k$-th Krylov subspace

$$K_k = \text{span}(b, Ab, \ldots, A^{k-1} b), \quad k > 0.$$ 

Let

$$E_k = \{(x, q) : x \in K_k, \|x\| = 1, q \in \mathbb{R}\}.$$ 

Define $k+1$ real numbers $c_0^*, c_1^*, \ldots, c_{k-1}^*$ and $q^*$ as

$$\|(A - q^* I)(c_0^* b + c_1^* Ab + \ldots + c_{k-1}^* A^{k-1} b)\| = \min \{\|(A - q I)x\| : (x, q) \in E_k\}.$$ 

The gmr algorithm produces a pair $(x_k, q_k)$ given by

$$x_k = c_0^* b + c_1^* Ab + \ldots + c_{k-1}^* A^{k-1} b, \quad q_k = q^*.$$ 

In other words, the gmr algorithm finds the normalized vector $x_k$ from the subspace $K_k$ and the real number $q_k$ for which the residual

$$r_k = \min \{\|Ax - q x\| : (x, q) \in E_k\} = \|Ax_k - q_k x_k\|$$

is as small as possible.

We now present the properties of the gmr algorithm which are useful for its implementation. Without loss of generality, assume that the vectors $b$, $Ab$, $\ldots$, $A^k b$ are linearly independent. Let $q_1, q_2, \ldots, q_{k+1}$ be an orthonormal
basis, the so-called Lanczos basis, of the subspace $K_k$ such that

$$Aq_i = \beta_i q_{i+1} + \alpha_i q_i + \beta_{i-1} q_{i-1}, \quad i = 1, 2, \ldots, k,$$

$$\alpha_i = \langle Aq_i, q_i \rangle, \quad \beta_i = \|Aq_i - \alpha_i q_i - \beta_{i-1} q_{i-1}\|, \quad i = 1, 2, \ldots, k, \quad \beta_0 = 0.$$ Let $Q_i = [q_1, q_2, \ldots, q_i]$ and $e_k = [0, \ldots, 0, 1]^T$. Then the $((k+1) \times k)$-matrix $D_k$,

$$D_k = Q_k^T A Q_k = \begin{bmatrix}
\alpha_1 & \beta_1 & 0 \\
\beta_1 & \alpha_2 & \beta_2 \\
& \ddots & \ddots & \ddots \\
& & \beta_k-1 & \alpha_k \\
0 & \beta_k & \beta_k & \beta_k
\end{bmatrix} = \begin{bmatrix}H_k \\
\beta_k e_k^T
\end{bmatrix},$$

is tridiagonal.

For $x \in K_k$, we thus have

$$x = \sum_{i=1}^{k} c_i q_i, \quad c_i \in \mathbb{R}.$$ Setting $c_0 = c_{k+1} = c_{k+2} = 0$ we get

$$(2) \quad r_k^2 = \min \left\{ \sum_{i=1}^{k+1} (c_{i-1} \beta_{i-1} + c_i \alpha_i - c_i q + \beta_i c_{i+1})^2 : q \in \mathbb{R}, \ c_i \in \mathbb{R}, \ \sum_{i=1}^{k} c_i^2 = 1 \right\}$$

$$= \min \{ \min \{ \|D_k(q) c\|^2 : \|c\| = 1 \} : q \in \mathbb{R} \}$$

$$= \min \{ \lambda_{\min}(D_k^T(q) D_k(q)) : q \in \mathbb{R} \},$$

where $D_k(q) = D_k - q I$, $D_k$ is defined by (1), and $\lambda_{\min}(X)$ denotes the smallest eigenvalue of the matrix $X$. Hence at the k-step of the gmr algorithm we want to find a number $q^*$ for which the smallest eigenvalue of the matrix $D_k^T(q) D_k(q)$ is minimal. Let $c^* = [c_1^*, \ldots, c_k^*]^T$ be the corresponding eigenvector of $D_k^T(q^*) D_k(q^*)$. Then the vector

$$x^* = \sum_{i=1}^{k} c_i^* q_i$$

is a unit vector from $K_k$ for which the minimum in (2) is attained.

In order to find the smallest eigenvalue of $D_k^T(q) D_k(q)$ we proceed as follows.

Let $H_k(q) = H_k - q I$, where $H_k$ is defined by (1). Then

$$(3) \quad D_k^T(q) D_k(q) = H_k^2(q) + \beta_k^2 e_k e_k^T.$$
Thus we want to find the smallest eigenvalue of the matrix $H_k^2(q)$ modified by the very special rank one perturbation $\beta_k^2 e_k e_k^T$. We shall use Golub's theorem about the eigenvalues of a matrix which is perturbed by a rank one matrix.

**Theorem** ([1]). Let

$$G = \text{diag}(g_i), \quad i = 1, 2, \ldots, n,$$

and

$$z = [z_1, \ldots, z_n]^T, \quad \|z\| = 1, \quad \vec{G} = G + azz^T.$$

If the $g_i$ are distinct, $a$ is nonzero and all components of the vector $z$ are nonzero, then the eigenvalues of $\vec{G}$ are the zeros of

$$\chi(t) = 1 + a \sum_{i=1}^n \frac{z_i^2}{(g_i - t)}.$$

Let $H_k(q) = U_k(A_k - qI) U_k^T$ be the spectral decomposition of the matrix $H_k(q)$, $A_k = \text{diag}(\lambda_i)$, where $\lambda_i$ are eigenvalues of $H_k$. From (3) we have

$$D_k^T(q)D_k(q) = U_k[(A_k - qI)^2 + \beta_k^2 U_k^T e_k e_k^T U_k]U_k^T.$$

Let $z = [z_1, \ldots, z_n]^T = U_k^T e_k$. Then $z$ is the last row of the matrix $U_k$. It is well known (see [4], pp. 129 and 124) that if $\beta_i \neq 0, i = 1, \ldots, k - 1$, then all elements of the vector $z$ are nonzero and all the $\lambda_i, i = 1, \ldots, k$, are distinct. Assume also that $\beta_k \neq 0$ and $q$ is chosen is such a way that

$$(\lambda_i - q)^2 \neq (\lambda_j - q)^2 \quad \text{for} \ i \neq j.$$

Applying Golub's theorem to the matrix $(A_k - qI)^2$ and to the vector $z$ we see that the eigenvalues of the matrix $D_k^T(q)D_k(q)$ are the zeros of the function $\chi_q$,

$$\chi_q(t) = 1 + \beta_k^2 \sum_{i=1}^k \frac{z_i^2}{[(\lambda_i - q)^2 - \lambda_i]}.$$

If we denote by $\zeta(q)$ the smallest zero of the function $\chi_q$, then (2) yields

$$r_k^2 = \min\{\zeta(q): q \in R\}.$$

Thus in order to find the minimal residual it is sufficient to compute the global minimum of the function $\zeta$. The implementation of the gmr algorithm presented in the next section is based on this property.

3. **Implementation of the gmr algorithm.** The implementation of the gmr algorithm is described as follows.

Having the matrix $D_k$ defined by (1) we compute the global minimum of the function $\zeta$ by performing the following steps:

(a) Compute all eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_k$ of the tridiagonal matrix $H_k$ and the last components $z_1, z_2, \ldots, z_k$ of all its eigenvectors. Order them so that $\lambda_1 < \lambda_2 < \ldots < \lambda_k$. 


(b) Define $k$ intervals $I_i$:

\[ I_1 = (-\infty, (\lambda_1 + \lambda_2)/2), \quad I_2 = ((\lambda_1 + \lambda_2)/2, (\lambda_2 + \lambda_3)/2), \ldots, \]

\[ I_k = ((\lambda_k - 1 + \lambda_k)/2, + \infty). \]

(c) Calculate the limits of the function $\zeta$ at the endpoints of $I_i$,

\[ \lim_{\varrho \to (\lambda_i + \lambda_{i+1})/2} \zeta(\varrho) = (\lambda_i - \lambda_{i+1})^2/4, \quad i = 1, \ldots, k-1. \]

(d) For each interval $I_i$, find the infimum of the function $\zeta$ for $i = 1, \ldots, k$.

(e) Take as the global minimum of $\zeta$ the smallest value among numbers obtained in (c) and (d); take $\varrho_k$ as the argument of the global minimum.

We now briefly discuss the steps of the above algorithm.

To perform step (a) we can use technique described in [2]. Since we are interested in eigenvalues and only in the last components of the eigenvectors, we can calculate them in cost proportional to $k^2$.

Steps (b), (c) and (e) are simple and they do not require explanation. The cost of performing each of them is proportional to $k$.

Let us now discuss step (d). In order to find the minimum of the function $\zeta$ in $I_i$ we propose using the iterative parabola method. It is known that $\zeta$ satisfies a Lipschitz condition with constant $4\|A\|$ and is analytic in a neighbourhood of a minimum point. Having computed values $\zeta(\varrho^{(i-2)})$, $\zeta(\varrho^{(i-1)})$, $\zeta(\varrho^{(i)})$, construct an interpolating polynomial $w$ of the second degree (parabola) such that

\[ w(\varrho^{(j)}) = \zeta(\varrho^{(j)}) \quad \text{for} \quad j = i-2, i-1, i. \]

Assume that $w'$ is not a constant. Then take $\varrho^{(i+1)}$ as the unique zero of $w'$,

\[ w'(\varrho^{(i+1)}) = 0, \quad i = 0, 1, 2, \ldots \]

It is well known that if starting points $\varrho^{(-2)}$, $\varrho^{(-1)}$, $\varrho^{(0)}$ are sufficiently close to the point $\varrho_k$ at which the function $\zeta$ attains its minimum and $\zeta''(\varrho_k) \neq 0$, then the sequence $\{\varrho^{(i)}\}$ produced by the parabola method converges with order 1.32 to the point $\varrho_k$.

Consider now the $i$-th interval

\[ I_i = ((\lambda_{i-1} + \lambda_i)/2, (\lambda_i + \lambda_{i+1})/2) \]

and let $\varrho \in I_i$. Then it is easy to see that the smallest zero of the function $\chi_\varrho$ lies in the interval $J_i$. Here

\[ J_1 = ((\lambda_1 - \varrho)^2, (\lambda_2 - \varrho)^2), \]

\[ J_i = ((\lambda_i - \varrho)^2, \min((\lambda_{i-1} - \varrho)^2, (\lambda_{i+1} - \varrho)^2)), \quad i = 2, \ldots, k-1, \]

\[ J_k = ((\lambda_k - \varrho)^2, (\lambda_{k-1} - \varrho)^2). \]
Note that the end points of the intervals $J_i$, $i = 1, 2, \ldots, k$, are the smallest two arguments for which the function $\chi_\epsilon$ has poles. In order to find the smallest zero of the function $\chi_\epsilon$ we use bisection to the equation $\chi_\epsilon(t) = 0$. One can also use other methods safeguarded with bisection. To find the minimum of the function $\zeta$ in the interval $I_i$ we perform a few (up to 6) steps of the parabola iterative method starting from $\lambda_i$ and two other points chosen close to $\lambda_i$. If at any step of the parabola method we obtain the point outside $I_i$, then we terminate and take as the minimum the smallest computed value of $\zeta(q)$ in the $I_i$. It is easy to see that the cost of this step is proportional to $k^2$. Thus the cost of performing all the steps (a)-(e) is of order $k^2$.

Having values $q_k$ and $\lambda_{\text{min}}(D_k^T(q_k)D_k(q_k))$ we can perform one step of the Wielandt algorithm to get the corresponding eigenvector $c^* = [c_1^*, \ldots, c_k^*]^T$ of $D_k^T(q_k)D_k(q_k)$. Some technical tricks for performing one step of Wielandt’s method without computing $D_k^T(q_k)D_k(q_k)$ effectively are given in the Appendix. Using this technique we can calculate the corresponding eigenvector $c^*$ performing $O(k)$ arithmetic operations. The cost of computing the vector

$$x_k = \sum_{i=1}^{k} c_i^* q_i$$

is of order $nk$ operations.

We end this section by the following remark. We have assumed that we were given the coefficients of the matrix $D_k$ and we dealt only with this matrix. If the coefficients $\alpha_1, \ldots, \alpha_k$ and $\beta_1, \ldots, \beta_k$ are not known, they and the orthonormal basis $q_1, q_2, \ldots, q_{k+1}$ can be found using the Lanczos process applied to the Krylov subspace, i.e., to the vectors $b$, $Ab$, $\ldots$, $A^k b$. Formulas for $\alpha_i$, $\beta_i$ and $q_i$ given in the previous section are, in general, very sensitive to roundoff errors and some reorthogonalization process is necessary. We will not discuss this subject here. The reader is referred to [4], where the detailed description of the selective reorthogonalization technique can be found. We stress that the cost of constructing the basis $q_1, q_2, \ldots, q_{k+1}$ and coefficients $\alpha_i$ and $\beta_i$ is proportional to $nk$, which is much more than $k^2$ for $k \ll n$.

### 4. Numerical results and comparison with the Lanczos algorithm

In this section we present some numerical results for the gmr algorithm and compare them with the results obtained for the Lanczos algorithm. This algorithm (see [4], p. 257) also uses the Krylov information

$$N_k(A, b) = [b, Ab, \ldots, A^k b].$$

The Lanczos algorithm, in fact, disregards the last codiagonal element $\beta_k$ in (1) since $\beta_k$ is only used to estimate the accuracy of the approximations. It deals with the resulting $(k \times k)$-matrix $H_k$. The algorithm produces pairs $(Q_k u_i, \lambda_i)$, $i = 1, 2, \ldots, k$, where $(u_i, \lambda_i)$, $i = 1, 2, \ldots, k$, are all eigenpairs of the matrix $H_k$, as approximations of eigenpairs of $A$. The cost of the Lanczos
The gmr algorithm

algorithm is essentially the same as the cost of the gmr algorithm. It is known that the smallest residual \( r_k^L \) of the Lanczos algorithm satisfies

\[
r_k^L = \min \{ \| A Q_k u_i - \lambda_i Q_k u_i \| : 1 \leq i \leq k \}
= |\beta_k| \min \{ |u_{ki}| : 1 \leq i \leq k \} \leq |\beta_k|,
\]

where \( u_{ki} \) is the last, \( k \)-th, component of the vector \( u_i \). It is also known that

\[
r_k^G = \min \{ \sqrt{\| Ax \|^2 - (Ax, x)^2} : x \in K_k, \| x \| = 1, (A - (Ax, x)I)x \bot K_k \}.
\]

The residual of the \( k \)-th step of the gmr algorithm is given by

\[
r_k^G = \min \{ \sqrt{\| Ax \|^2 - (Ax, x)^2} : x \in K_k, \| x \| = 1 \}.
\]

It is easy to see that \( r_k^G \leq r_k^L \). Moreover, it is known that \( r_1^G = r_1^L \) and \( r_n^G = r_n^L = 0 \). This and similarity of the formulas for residuals might suggest that \( r_k^G \) should be close to \( r_k^G \) for \( k = 1, 2, \ldots, n \). This intuition is incorrect. As shown in [3] the small difference in the formulas leads to completely different values for the residuals of the two algorithms (see Example 1).

For all examples, both the gmr and Lanczos algorithms are tested for \( k = 1, 2, \ldots, n \) and their residuals are compared. Without loss of generality we confine ourselves to tridiagonal matrices. For simplicity we set the vector \( b = [1, 0, \ldots, 0]^T \). Numerical tests were performed on a DEC-20 computer with 8 decimal accuracy at the Computer Science Department of Columbia University. Some tests were also performed on a DEC-20 computer at the Computer Science Department of the University of Utah in Salt Lake City and on VAX 750 computer at AT & T Bell Laboratory in Murray Hill. We first report the results for the following matrix.

**Example 1.** Let

\[
\alpha_i = 0, \quad i = 1, 2, \ldots, 101,
\]

\[
\beta_i = 0.5, \quad i = 1, 2, \ldots, 100, \quad i \neq 1, 11, 21, \ldots, 91,
\]

and

\[
\beta_1 = \beta_{11} = \beta_{21} = \ldots = \beta_{91} = 0.05.
\]

The sequence of residuals of the gmr algorithm is strictly decreasing, while the sequence of residuals of the Lanczos algorithm does not have this property. In fact, only the subsequence \( \{ r_{2k-1}^L \} \), for \( k \geq 10 \), of the Lanczos residuals is nonincreasing. The gmr residuals \( r_{2k-1}^G \) are 2 or 3 times smaller than \( r_{2k-1}^L \). Both algorithms terminate at the 71-st step by reaching residuals smaller than \( 10^{-8} \). For even indices larger than 16, the Lanczos algorithm does not take full advantage of the available information and produced large residuals. For instance,

\[
r_{64}^L = 4.2_{10} - 4, \quad r_{66}^L = 3.9_{10} - 4, \quad r_{68}^L = 5.0_{10} - 4, \quad r_{70}^L = 1.2_{10} - 3,
\]
while

\[ r_{63}^L = r_{65}^L = r_{67}^L = r_{69}^L = 4.9_{10} - 8. \]

This means that at the 69-th step the Lanczos algorithm guarantees 7 correct decimal digits, while at the next step only 3. The Lanczos algorithm increases the residual more than 24000 times at the 70-th step. By contrast, we stress that the residuals of the gmr algorithm are

\[ r_{65}^G = 2.8_{10} - 8 \geq r_{66}^G \geq r_{67}^G \geq r_{68}^G \geq r_{69}^G \geq r_{70}^G = 2.0_{10} - 8. \]

**EXAMPLE 2.** Let \( \alpha_i = 0 \) and \( \beta_i = 1/2 \) for \( i = 1, 2, \ldots, n \) for \( n \geq 800 \). For this matrix both algorithms produce decreasing sequences of residuals. Table 1 shows how many steps one has to perform using the gmr (G) and Lanczos (L) algorithms to get residuals not greater than \( \varepsilon \). The gmr algorithm uses significantly fewer steps.

<table>
<thead>
<tr>
<th>( \varepsilon )</th>
<th>( 5_{10} - 1 )</th>
<th>( 1_{10} - 1 )</th>
<th>( 5_{10} - 2 )</th>
<th>( 1_{10} - 2 )</th>
<th>( 5_{10} - 3 )</th>
<th>( 1_{10} - 3 )</th>
<th>( 5_{10} - 4 )</th>
<th>( 1_{10} - 4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>* L</td>
<td>1</td>
<td>7</td>
<td>12</td>
<td>36</td>
<td>58</td>
<td>170</td>
<td>270</td>
<td>780</td>
</tr>
<tr>
<td>* G</td>
<td>1</td>
<td>6</td>
<td>9</td>
<td>21</td>
<td>30</td>
<td>69</td>
<td>98</td>
<td>221</td>
</tr>
</tbody>
</table>

**EXAMPLE 3.** The increase of the Lanczos residuals observed in Example 1 occurs quite often. For instance, for a tridiagonal matrix of dimension 100 defined as follows:

\[ \alpha_1 = \alpha_2 = 1/3, \quad \alpha_3 = \alpha_4 = -1/3, \quad \alpha_5 = \alpha_6 = 1/3, \quad \ldots, \quad \alpha_{99} = \alpha_{100} = 1/3 \]

and

\[ \beta_i = (-1)^{i+1}(1/3), \quad i = 1, 2, \ldots, 99, \]

the Lanczos algorithm increases the residual error at every fourth step and the increase is very large. For instance,

\[ r_{50}^L = 1.8_{10} - 3, \quad r_{54}^L = 1.6_{10} - 3, \quad r_{58}^L = 1.4_{10} - 3, \quad r_{62}^L = 1.3_{10} - 3, \]

while all other residuals from the step 49 to 61 vary between \( 4.5_{10} - 7 \) and \( 2.2_{10} - 8 \).

**EXAMPLE 4.** One of the goals of testing is to establish empirically how fast residuals of the gmr and Lanczos algorithms converge for symmetric matrices. For the gmr algorithm it has been proved in [3] that for any symmetric matrix \( A \) and \( k < n \)

\[ r_k^G \leq \| A \|/k, \]

and for any \( k < n \) there exists a real symmetric matrix \( A \) for which

\[ r_k^G \geq \| A \|/2k. \]
Similarly, for the Lanczos algorithm the bounds are
\[ r_k^G \leq \|A\|/\sqrt{k}, \]
and for any \( k < n \) there exists a symmetric matrix \( A \) for which
\[ r_k^L \geq \|A\|/(\sqrt{k} + 1). \]
We want to find out how sharp these bounds are for specific matrices with norm bounded by unity. In order to measure the speed of convergence define the sequences \( \{p_k^G\}, \{p_k^L\} \) as
\[ r_k^G = (k p_k^G)^{-1}, \quad r_k^L = (k p_k^L)^{-1}, \quad k = 2, 3, \ldots, n-1. \]
From theory we know that \( p_k^G \geq 1 \) and \( p_k^L \geq 1/2 \). We computed \( p_k^G \) and \( p_k^L \) for many tridiagonal matrices with norm bounded by unity. The smallest values of \( p_k^G \) and \( p_k^L \) were obtained for matrices with zeros on the main diagonal and with slightly increasing codiagonal elements. We report three examples of such matrices.

(i) For the matrix of dimension 501 with codiagonal elements \( \beta_i \) equal to \( i/(2(n-1)) \), the gmr residuals decrease at every second step, while the Lanczos residuals do not decrease at all. Both algorithms begin at the same residuals equal to \( 1_{10} - 3 \) and at the 500-th step they reach
\[ r_{500}^L = 1.1_{10} - 2 \quad \text{and} \quad r_{500}^G = 3.6_{10} - 4. \]
The sequences \( p_k^L \) and \( p_k^G \) decrease very slowly for \( k \geq 50 \). For the Lanczos algorithm we obtain
\[ p_{50}^L = 0.79, \quad p_{250}^L = 0.74, \quad p_{500}^L = 0.72. \]
For the gmr algorithm we get
\[ p_{50}^G = 1.33, \quad p_{250}^G = 1.29, \quad p_{500}^G = 1.27. \]
(ii) We also tested the \((201 \times 201)\)-matrix with codiagonal elements
\[ \beta_i = \sqrt{i/(n-1)/2}, \quad i = 1, 2, \ldots, 200. \]
The gmr residuals decrease very slowly at every step. For instance,
\[ r_1^G = 3.5_{10} - 2, \quad r_{50}^G = 7.7_{10} - 3, \quad r_{100}^G = 5.5_{10} - 3, \]
\[ r_{150}^G = 4.5_{10} - 3, \quad r_{200}^G = 3.9_{10} - 3. \]
The Lanczos residuals are constant for all 200 steps:
\[ r_1^L = r_2^L = \ldots = r_{200}^L = 3.5_{10} - 2. \]
The sequences \( p_k^L \) and \( p_k^G \) are both decreasing. For the Lanczos algorithm we obtain

\[
p_{20}^L = 1.12, \quad p_{50}^L = 0.85, \quad p_{100}^L = 0.73, \\
p_{150}^L = 0.67, \quad p_{175}^L = 0.65, \quad p_{200}^L = 0.631;
\]

while for the gmr algorithm we have

\[
p_{20}^G = 1.48, \quad p_{50}^G = 1.24, \quad p_{100}^G = 1.13, \\
p_{150}^G = 1.08, \quad p_{175}^G = 1.06, \quad p_{200}^G = 1.046.
\]

(iii) The small values of \( p_k^L \) and \( p_k^G \) are also obtained for the \((200 \times 200)\)-matrix with

\[
\beta_i = \log(i+1)/(2\log(n)), \quad i = 1, 2, \ldots, 199,
\]

on the codiagonal. A few results for both algorithms are shown in Table 2.

<table>
<thead>
<tr>
<th>( k )</th>
<th>25</th>
<th>50</th>
<th>75</th>
<th>100</th>
<th>125</th>
<th>150</th>
<th>175</th>
<th>180</th>
<th>190</th>
<th>199</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_k^L )</td>
<td>5.10−2</td>
<td>3.90−2</td>
<td>3.30−2</td>
<td>3.00−2</td>
<td>2.70−2</td>
<td>2.50−2</td>
<td>2.30−2</td>
<td>2.30−2</td>
<td>2.30−2</td>
<td>2.20−2</td>
</tr>
<tr>
<td>( r_k^G )</td>
<td>2.10−2</td>
<td>1.30−2</td>
<td>9.50−3</td>
<td>7.40−3</td>
<td>6.20−3</td>
<td>5.30−3</td>
<td>4.60−3</td>
<td>4.50−3</td>
<td>4.30−3</td>
<td>4.10−3</td>
</tr>
<tr>
<td>( p_k^L )</td>
<td>0.93</td>
<td>0.83</td>
<td>0.79</td>
<td>0.76</td>
<td>0.75</td>
<td>0.74</td>
<td>0.726</td>
<td>0.725</td>
<td>0.721</td>
<td>0.719</td>
</tr>
<tr>
<td>( p_k^G )</td>
<td>1.21</td>
<td>1.11</td>
<td>1.08</td>
<td>1.06</td>
<td>1.05</td>
<td>1.05</td>
<td>1.042</td>
<td>1.041</td>
<td>1.039</td>
<td>1.038</td>
</tr>
</tbody>
</table>

For large \( k \), the sequence \( p_k^G \) is quite close to 1. We believe that for a larger dimension \( n \) the sequence \( p_k^G \) would be even closer to 1. Observe that for the last two matrices the Lanczos sequence \( p_k^L \) is relatively close to 1/2. We believe that there exists a symmetric matrix for which the sequence \( p_k^L \) approaches 1/2.

For the same matrix as before, Table 3 shows how many steps are needed to reduce the first residual

\[
r_1^G = r_1^L = 6.5_{10}−2
\]

by a factor of \( q \) using the Lanczos or gmr algorithms.

<table>
<thead>
<tr>
<th>( q )</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td># ( L )</td>
<td>78</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td># ( G )</td>
<td>10</td>
<td>23</td>
<td>38</td>
<td>51</td>
<td>63</td>
<td>77</td>
<td>89</td>
<td>104</td>
<td>117</td>
<td>130</td>
<td>144</td>
<td>156</td>
<td>172</td>
<td>186</td>
<td>200</td>
</tr>
</tbody>
</table>

Example 5. Random matrices. We tested many tridiagonal matrices with coefficients generated pseudo-randomly with uniform distribution in the interval \([-1/3, 1/3]\). We do not observe large differences between residuals of both algorithms. However, very often the sequence of Lanczos residuals is not strictly decreasing, though the increase is rather small. In general, the \( k \)-th residual \( r_k^L \) does not exceed the \((k−1)\)-st residual multiplied by 3 or 4. However, for a few matrices, \( r_k^L = 20r_{k−1}^L \) for some \( k \).
Both algorithms were efficient. For random matrices of dimension 201 they computed the residuals smaller than $4_{10}^{-8}$ after about 25 steps. Fast convergence of both algorithms for random matrices can be easily explained. Indeed, the sequences of numbers generated pseudo-randomly from the interval $[-1/3, 1/3]$ are unlikely to be increasing, and almost surely some codiagonal elements are small. These two properties make the residuals of both algorithms small.

Both algorithms were tested for 80 random $(201 \times 201)$-matrices. For each matrix the gmr residuals are smaller than the corresponding Lanczos residuals. The differences between them are usually insignificant. For each of eighty matrices we compute the number of steps needed to make the residual less than $\varepsilon$. Table 4 presents the average number of steps needed by the Lanczos and gmr algorithms for a few values of $\varepsilon$. These tests suggest that for random matrices the efficiency of both algorithms is nearly the same.

<table>
<thead>
<tr>
<th>$\varepsilon$</th>
<th>$1_{10}^{-1}$</th>
<th>$1_{10}^{-2}$</th>
<th>$1_{10}^{-3}$</th>
<th>$1_{10}^{-4}$</th>
<th>$1_{10}^{-5}$</th>
<th>$1_{10}^{-6}$</th>
<th>$1_{10}^{-7}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average number of steps</td>
<td>$L$</td>
<td>2.1</td>
<td>5.94</td>
<td>10.09</td>
<td>15.1</td>
<td>18.23</td>
<td>20.96</td>
</tr>
<tr>
<td></td>
<td>$G$</td>
<td>2.06</td>
<td>5.44</td>
<td>9.16</td>
<td>13.88</td>
<td>17.95</td>
<td>20.79</td>
</tr>
</tbody>
</table>

**Appendix.** We describe how to perform one step of the Wielandt algorithm in order to find the eigenvector of the matrix $D_k^T(q)D_k(q)$ corresponding to the smallest eigenvalue. Assume that we have a sufficiently good approximation $\lambda$, $\lambda \geq 0$, of the smallest eigenvalue of the matrix $D_k^T(q)D_k(q)$. We must solve the system of linear equations

$$(D_k^T(q)D_k(q) - \lambda I)u = w \quad \text{for given } w \in \mathbb{R}^k,$$

which appears in the Wielandt algorithm.

Assume that the matrices $H_k^2(q) - \lambda I$ and $D_k^T(q)D_k(q) - \lambda I$ are nonsingular. Then from the formula of Sherman, Morrison and Woodbury

$$(A + uv^T)^{-1} = A^{-1} - 1/(1 + v^T A^{-1} u) A^{-1} u v^T A^{-1},$$

applied to the matrix $A = H_k^2(q) - \lambda I$ and the vectors $u = v = e_k$, we obtain

$$(D_k^T(q)D_k(q) - \lambda I)^{-1} = (H_k^2(q) - \lambda I + \beta_k^2 e_k e_k^T)^{-1}$$

$$= [I - 1/(1 + \omega_k) \beta_k^2 (H_k^2(q) - \lambda I)^{-1} e_k e_k^T] (H_k^2(q) - \lambda I)^{-1},$$

where $\omega_k = \beta_k^2 e_k^T (H_k^2(q) - \lambda I)^{-1} e_k$.

Let

$$s = (H_k^2(q) - \lambda I)^{-1} w = (H_k(q) - \sqrt{\lambda} I)^{-1} (H_k(q) + \sqrt{\lambda} I)^{-1} w.$$

Then

$$\beta_k e_k^T (H_k^2(q) - \lambda I)^{-1} w = \beta_k e_k^T s = \beta_k s_k, \quad \text{where } s = (s_1, \ldots, s_k)^T.$$
Thus we have
\[
(D_k^T(q)D_k(q) - \lambda I)^{-1}w = s - \beta_k^2 s_k/(1 + \omega_k)(H_k^T(q) - \lambda I)^{-1}e_k.
\]
Put \( t = (t_1, \ldots, t_k)^T = (H_k^2(q) - \lambda I)^{-1}e_k \). It is easy to calculate that \( \omega_k = \beta_k^2 t_k \) and
\[
u = (D_k^T(q)D_k(q) - \lambda I)^{-1}w = s - \beta_k^2 s_k/(1 + \beta_k^2 t_k) t,
\]
where
\[
s = (H_k(q) - \sqrt{\lambda} I)^{-1}(H_k(q) + \sqrt{\lambda} I)^{-1}w,
\]
\[
t = (H_k(q) - \sqrt{\lambda} I)^{-1}(H_k(q) + \sqrt{\lambda} I)^{-1}e_k.
\]
To solve systems of equations with matrices \( H_k(q) + \sqrt{\lambda} I \) and \( H_k(q) - \sqrt{\lambda} I \) we can use any numerically stable method (we use Gaussian elimination with partial pivoting) for solving systems of linear equations.

Acknowledgement. I am very pleased to acknowledge Professor Henryk Woźniakowski for his many helpful discussions concerning the subject of this paper. Without his comments and suggestions it would have been impossible to write this paper. I would like to thank Professor Joseph Traub, who carefully read earlier versions of this paper and suggested some improvements. Many thanks are due to Professors Frank Stenger and Kris Sikorski and Dr. David Lee for their cooperation in performing numerical tests.

References