Available online at www.sciencedirect.com

ScienceDirect

Linear Algebra and its Applications 429 (2008) 2315–2334

LINEAR ALGEBRA AND ITS APPLICATIONS

www.elsevier.com/locate/laa

Augmented Block Householder Arnoldi method

James Baglama [1]

Department of Mathematics, University of Rhode Island, Kingston, RI 02881, United States

Received 19 March 2007; accepted 18 December 2007
Available online 4 March 2008
Submitted by D. Szyld

Abstract

Computing the eigenvalues and eigenvectors of a large sparse nonsymmetric matrix arises in many applications and can be a very computationally challenging problem. In this paper we propose the Augmented Block Householder Arnoldi (ABHA) method that combines the advantages of a block routine with an augmented Krylov routine. A public domain MATLAB code `ahbeigs` has been developed and numerical experiments indicate that the code is competitive with other publicly available codes.

AMS classification: 65F15; 15A18

Keywords: Partial eigenvalue value decomposition; Restarted iterative method; Implicit shifts; Augmentation; Krylov; Householder

1. Introduction

Finding the extreme eigenvalues and associated eigenvectors of a large-scale eigenvalue problem

$$Ax = \lambda x \quad A \in \mathbf{R}^{n\times n} \tag{1.1}$$

is one of the most computationally challenging and commonly occurring problems to date. Applications can be found in almost all scientific disciplines, e.g. computational fluid dynamics,

E-mail address: jbaglama@math.uri.edu
URL: http://hypatia.math.uri.edu/∼jbaglama
[1] Research supported in part by NSF grant DMS-0311786.

0024-3795/$ - see front matter
doi:10.1016/j.laa.2007.12.021

electrical engineering, oceanography, and quantum chemistry. See Bai et al. [12] for a description of some applications, references, and collection of matrices.

Many algorithms are based on the Arnoldi process [2]. For a given starting vector x, the Arnoldi process builds an orthonormal basis for the Krylov subspace

$$\mathbb{K}_m(A, x) = \text{span}\{x, Ax, A^2x, \ldots, A^{m-1}x\}. \tag{1.2}$$

The orthonormal basis yields a projection matrix and a relationship that is often referred to as the Arnoldi decomposition. The eigenvalues and eigenvectors of the projection matrix are then used as an approximation to the eigenvalues and eigenvectors of A. However, in order to get an acceptable approximation to the eigenpairs, m must typically be large. This is not always possible because of storage constraints and orthogonality issues. To overcome these difficulties a restarted Arnoldi method can be used. A restarted method maintains a modest value for $m \ll n$ where each restart either implicitly or explicitly modifies the starting vector x for the next iteration so that a better approximation is obtained. This creates a sequence of Krylov subspaces that hopefully converge to an invariant subspace containing the desired eigenvectors.

In a seminal paper, Sorensen [44] proposed the Implicitly Restarted Arnoldi (IRA) method for the computation of a few eigenpairs of a large sparse nonsymmetric matrix. This restarted method implicitly modifies the starting vector on each iteration via a shifted curtailed QR-algorithm. This successful method is the foundation for the very popular eigenvalue software package ARPACK [28]. Lehoucq and Scott [27] provided a comparison of softwares and concluded that ARPACK was generally the fastest and most dependable. However, there are numerical examples that have shown that if care is not taken during the implementation, propagated round-off errors can delay or prevent convergence of desired eigenvalues and eigenvectors [27,46]. This forward numerical instability is due to the underlying QR-algorithm, see Lehoucq and Sorensen [27] for remedies. Morgan [34] showed the IRA method of Sorensen [44] can be implemented by augmenting the sequence of Krylov subspace basis by certain Ritz vectors. This mathematically equivalent implementation can be less sensitive to propagated round-off errors than the implementation in [44]. This relationship has been recently exploited by Wu and Simon [50] for symmetric eigenvalue problems, by Morgan [35,36] for linear systems and nonsymmetric eigenvalue problems, and by Baglama and Reichel [10] for singular value problems. The extension of this idea to block Krylov subspaces (1.3) has been implemented by Morgan [37] and Gu and Ciao [21] for linear systems, by Möller [33] for nonsymmetric eigenvalue problems, and by Baglama and Reichel [11] for singular value problems. These extensions to block methods have many favorable attributes, see the discussion below, eventhough they are no longer mathematically equivalent to the block form of the IRA method, see Möller [33] for details. The main focus of this paper is on creating an augmented block Krylov subspace method for the eigenvalue problem (1.1).

The block Arnoldi method only differs from the Arnoldi method in that it uses a set of starting vectors $X = [x_1, x_2, \ldots, x_r]$ and builds an orthonormal basis for the block Krylov subspace

$$\mathbb{K}_{mr}(A, X) = \text{span}\{X, AX, A^2X, \ldots, A^{m-1}X\}. \tag{1.3}$$

A block routine typically requires more computational effort and larger subspaces for acceptable approximations. However, the benefits of a block routine include the ability to compute multiple or clustered eigenvalues more efficiently than an unblocked routine [7,8,25], the use of Level 3 BLAS [15] matrix–matrix products for faster algorithms [14, Section 2.6], and the ability to compute matrix–vector products with a block of vectors. These advantages of the block routines have resulted in a considerable number of algorithms/software in recent years; see, e.g. [3,8,11,20,21,22,25,30,32,33,37,42,43,51] and references therein. In particular, Lehoucq

and Maschhoff [25] created an implicitly restarted block Arnoldi (bIRAM) method which is a straightforward generalization of the IRA method to block form.

Currently, ARPACK does not include the bIRAM method. This may be due to several difficulties in implementation, e.g. shift strategy. In the IRA method, during the implicit modification of the starting vector, a shift is applied, followed by a reduction in the number of vectors in the Arnoldi decomposition. There is a one-to-one correspondence between the reduction of the Arnoldi decomposition and the number of shifts applied, i.e. reducing m to $m-1$ in (1.2) after one shift is applied. However, in the generalization to block form there is no longer a one-to-one correspondence. The ratio becomes one-to-block size, i.e. reducing mr to $mr-r$ in (1.3) after one shift is applied. The current shift strategy in ARPACK is "exact" shifts which are the unwanted Ritz values [44]. Applying "exact" shifts in the bIRAM method may cause a significant number of unwanted Ritz values not being applied as shifts. This can slow down the convergence drastically. One solution may be to use an alternate shift strategy, e.g., the zeros of Chebyshev polynomials [40], Leja points [6], harmonic Ritz values [38], or refined Ritz values [23]. The computer code `irbleigs`,[2] which is based on the implicitly restarted block Lanczos method, uses Leja points as shifts for solving the symmetric eigenvalue problem, see [8,9] for details. However, the focus of this paper is not on shift strategies, but rather on circumventing the use of shifts by implementing an augmented block routine.

We have developed an augmented block Householder Arnoldi (ABHA) method that combines the advantages of a block routine and an augmented routine. The development of an augmented block Arnoldi method is not new, Morgan presents an augmented Arnoldi block routine in [37] to solve linear systems of equations and Möller in [33] for solving nonsymmetric eigenvalue problems. However, this paper presents a new implementation, along with a public domain MATLAB code, `ahbeigs`.[3]

This paper is organized as follows. Section 2 introduces notation and presents the block Householder Arnoldi algorithm and in Section 3 we outline the ABHA method and provide algorithms. The MATLAB code `ahbeigs` is presented in Section 4 and a few numerical examples are presented in Section 5. Concluding remarks are contained in Section 6.

2. Block Householder Arnoldi

The foundation of our ABHA method is the use of the Householder process to create an orthonormal basis for the block Krylov subspace (1.3). Algorithm 2.1 extends the Householder Arnoldi method developed by Walker [49] to block form. Although the Householder process for creating an orthonormal basis for the Krylov subspace is more expensive than the Modified Gram-Schmidt (MGS) process with partial reorthogonalization, it is less expensive than the MGS process with full reorthogonalization [39].

Our method uses the compact WY representation of the Householder product [41]. The Q matrix in the Householder QR-decomposition is formed from a product of Householder matrices [39]. The compact WY representation of Q replaces the product with the form, $I + YTY^{\mathrm{T}}$, where Y is a lower trapezoidal matrix and T is a square upper triangular matrix. See [13,48] for details on block reflectors. The advantage of this representation is the heavy use of Level 3 BLAS matrix–matrix operations.

[2] Computer code is available at http://www.math.uri.edu/∼jbaglama or http://math.nist.gov/toms/.

[3] Computer code can be downloaded from http://www.math.uri.edu/∼jbaglama or http://www.mathworks.com/matlabcentral/fileexchange.

Algorithm 2.1. Block Arnoldi Householder Algorithm

Input: $A \in \mathbf{R}^{n\times n}$;

Output: $Y \in \mathbf{R}^{n\times mr+r}$, $T \in \mathbf{R}^{mr+r\times mr+r}$,

$H_{(i,j)} \in \mathbf{R}^{r\times r}, \quad j = 0,\ldots m,\ i = 1,\ldots,j+1;$

(1) Choose r random vectors x_i and set $X := [x_1,\ldots,x_r] \in \mathbf{R}^{n\times r}$;

(2) for $j = 0, 1, \ldots, m$

(3) Compute the Householder QR-decomposition where

$$X(jr+1:n, 1:r) = QR \quad \text{and} \quad Q = (I + WSW^{\mathrm{T}}) \begin{bmatrix} I \\ 0 \end{bmatrix} \begin{matrix} \} \in \mathbf{R}^{r\times r} \\ \} \in \mathbf{R}^{(n-r)\times r} \end{matrix}$$

(4) if $j = 0$

(a) Set $\begin{cases} Y := W \\ T := S \\ H_{(1,0)} := R \end{cases}$

else

(b) Set $\begin{cases} \begin{bmatrix} H_{(1,j)} \\ \vdots \\ H_{(j,j)} \end{bmatrix} := \begin{bmatrix} \\ X(1:jr, 1:r) \\ \\ \end{bmatrix} \end{cases} \in \mathbf{R}^{jr\times r}$

$[H_{(j+1,j)}] \qquad [\qquad R \qquad]\} \in \mathbf{R}^{r\times r}$

(c) Set $\begin{cases} W := \begin{bmatrix} 0 \\ W \end{bmatrix} \begin{matrix} \} \in \mathbf{R}^{jr\times r} \\ \} \in \mathbf{R}^{n-jr\times r} \end{matrix} \\ T := \begin{bmatrix} T & TY^{\mathrm{T}}WS \\ 0 & S \end{bmatrix} \in \mathbf{R}^{(j+1)r\times(j+1)r} \\ Y := [Y \quad W] \in \mathbf{R}^{n\times(j+1)r} \end{cases}$

end

(5) if $j < m$

(6) Compute $X := (I + YT^{\mathrm{T}}Y^{\mathrm{T}})A(I + YTY^{\mathrm{T}}) \begin{bmatrix} 0 \\ I \\ 0 \end{bmatrix} \begin{matrix} \} \in \mathbf{R}^{jr\times r} \\ \} \in \mathbf{R}^{r\times r} \\ \} \in \mathbf{R}^{(n-jr-r)\times r} \end{matrix}$

(7) end

(8) end

Consider the following orthonormal matrix:

$$V_{mr+r} = [V_{(1)}, V_{(2)}, \ldots, V_{(m+1)}] = (I + YTY^{\mathrm{T}})I_{mr+r}, \quad V_{(i)} \in \mathbf{R}^{n\times r}, \tag{2.1}$$

that is created from output of Algorithm 2.1. It is easy to see that Algorithm 2.1 computes a truncated QR decomposition of the matrix $[X, AV_{(1)}, \ldots, AV_{(m)}]$ such that

$$(I + YT^{\mathrm{T}}Y^{\mathrm{T}})\,[X, AV_{(1)}, \ldots, AV_{(m)}] = \begin{bmatrix} H_{(1,0)} & \begin{bmatrix} H_{(1,1)} \\ H_{(2,1)} \end{bmatrix} & \cdots & \begin{bmatrix} H_{(1,m)} \\ \vdots \\ H_{(m+1,m)} \end{bmatrix} \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

where $H_{(i,j)} \in \mathbf{R}^{r\times r}, i = 1,\ldots,j, j = 1,\ldots,m$ are $r \times r$ blocks, and $H_{(j+1,j)} \in \mathbf{R}^{r\times r}$ are upper triangular blocks, that are obtained from step 4 of Algorithm 2.1.

Let

$$H_{mr+r} = \begin{bmatrix} H_{(1,1)} & & \cdots & & H_{(1,m)} \\ H_{(2,1)} & H_{(2,2)} & & & \\ & & \ddots & & \vdots \\ & & \ddots & \ddots & \\ & & & H_{(m,m-1)} & H_{(m,m)} \\ 0 & & & & H_{(m+1,m)} \end{bmatrix} \in \mathbf{R}^{mr+r\times mr}, \tag{2.2}$$

be the upper block Hessenberg matrix then the following is a block Arnoldi decomposition

$$\begin{aligned} AV_{mr} &= V_{mr+r}H_{mr+r} \\ &\quad \text{or} \\ AV_{mr} &= V_{mr}H_{mr} + V_{(m+1)}H_{(m+1,m)}E_r^{\mathrm{T}}. \end{aligned} \tag{2.3}$$

During the iteration of the block Arnoldi method a sub-diagonal block $H_{(j+1,j)}$ of the blocked Hessenberg matrix (2.2) may become singular, implying that a set of vectors in (1.3) have become linearly dependent on previously generated vectors. Unlike the single vector Arnoldi method, this occurrence of linearly dependent vectors may not imply an invariant subspace has been computed unless $H_{(j+1,j)} \equiv 0$. This "breakdown" rarely occurs, but still needs to be addressed in the development of a robust software. See [5] for details on the implication of the breakdown when using the MGS process in the implicitly restarted block Lanczos method and for a solution. The Householder block Arnoldi method handles this in step 3 of Algorithm 2.1 via the underlying LAPACK's QR algorithm with no additional steps required, i.e. a random vector is introduced at step 3 so that a valid QR factorization is computed.

3. Restarting with Schur vectors

Following the approach by Stewart [46] and more recently by Möller [33] we now outline our block restarted method.

Assume the block Arnoldi decomposition (2.3) is available and compute the real Schur decompostion of H_{mr} such that,

$$H_{mr}Q_{mr}^{(H_{mr})} = Q_{mr}^{(H_{mr})}U_{mr}^{(H_{mr})}, \tag{3.1}$$

where $U_{mr}^{(H_{mr})}$ is a quasi-triangular matrix with the eigenvalues of H_{mr} occurring on the diagonal as either a real 1×1 matrix or a real 2×2 matrix. The latter case constitutes complex conjugate pairs and $Q_{mr}^{(H_{mr})} = [q_1^{(H_{mr})}, q_2^{(H_{mr})}, \ldots, q_{mr}^{(H_{mr})}]$ is an orthogonal matrix. The real Schur decomposition can be reordered so that the desired eigenvalues occur in the upper left part, $U_k^{(H_{mr})}$ of the matrix $U_{mr}^{(H_{mr})}$. This can be accomplished via LAPACK's subroutine `dtrsen` [1]. See Bai and Demmel [4] for details on reordering the real Schur decomposition.

Let k be the number of desired eigenvalues and we assume for ease of presentation that k does not split a conjugate pair. In practice, if k does split a conjugate pair it is replaced with $k + 1$. After reordering the real Schur decomposition of H_{mr} and truncating the last $mr - k$ columns we have

$$H_{mr}Q_k^{(H_{mr})} = Q_k^{(H_{mr})}U_k^{(H_{mr})}. \tag{3.2}$$

For the given matrix A we determine the approximate real partial Schur decomposition $AQ_k^{(A)} = Q_k^{(A)} U_k^{(A)}$ from (3.1), where

$$Q_k^{(A)} = [q_1^{(A)}, q_2^{(A)}, \ldots, q_k^{(A)}] = V_{mr} Q_k^{(H_{mr})} \quad \text{and} \quad U_k^{(A)} = U_k^{(H_{mr})}. \tag{3.3}$$

The partial eigenvalue decomposition of A is easily obtained by computing the eigenvalue decomposition $U_k^{(H_{mr})} S_k = S_k D_k^{(H_{mr})}$ and setting

$$V_k^{(A)} = V_{mr} Q_k^{(H_{mr})} S_k, \quad D_k^{(A)} = D_k^{(H_{mr})} \quad \text{to get } AV_k^{(A)} = V_k^{(A)} D_k^{(A)}. \tag{3.4}$$

Using (2.3), (3.1), and (3.4) we have the following

$$AV_k^{(A)} - V_k^{(A)} D_k^{(A)} = V_{(m+1)} H_{(m+1,m)} E_r^{\mathrm{T}} Q_k^{(H_{mr})} S_k. \tag{3.5}$$

We see from (3.5) that we have an acceptable approximate partial eigenvalue decomposition of A when

$$\| H_{(m+1,m)} E_r^{\mathrm{T}} Q_k^{(H_{mr})} S_k \| \leqslant \alpha \cdot tol, \tag{3.6}$$

where *tol* is a user input tolerance value and α is chosen to assure a small backward error, see ARPACK user guide [29] for a discussion on choosing α.

Rearranging (3.5) we have the following relationship:

$$AQ_k^{(A)} = [Q_k^{(A)} \;\; V_{(m+1)}] \begin{bmatrix} U_k^{(A)} \\ H_{(m+1,m)} E_r^{\mathrm{T}} Q_k^{(H_{mr})} \end{bmatrix}. \tag{3.7}$$

Möller [33] showed via orthogonal transformations that (3.7) can be transformed into a k block Arnoldi decomposition. Therefore, the block Arnoldi algorithm can be restarted or continued with the matrix $V_{(m+1)}$. In order to continue our block Arnoldi Householder Algorithm 2.1 the orthogonal matrix $[Q_k^{(A)} \;\; V_{(m+1)}]$ must be placed into the compact WY representation of the Householder product. This is accomplished in Algorithm 3.1. The Algorithm makes use of the fact that the matrix $[Q_k^{(A)} \;\; V_{(m+1)}]$ is orthogonal, see step 3 of Algorithm 3.1. Then Algorithm 3.1 computes the product $Q_k^{(A)} := V_{mr} Q_k^{(H_{mr})}$ in steps 1 and 6, while placing the matrix $[Q_k^{(A)} \;\; V_{(m+1)}]$ into WY representation. Notice when n is large the dominating computational expense of Algorithm 3.1 occurs at step 6a, which parallels the computational cost that is encountered when updating the block Arnoldi decomposition in the bIRAM [25].

Algorithm 3.1. Algorithm orthogonal

Input: Householder WY-compact matrices: $Y \in \mathbf{R}^{n \times \ell + r}$ and $T \in \mathbf{R}^{\ell + r \times \ell + r}$

Orthogonal matrix: $Q_k^{(H_{mr})} \in \mathbf{R}^{\ell \times k}$

Output: Householder WY-compact matrices: $\bar{Y} \in \mathbf{R}^{n \times k + r}$ and $\bar{T} \in \mathbf{R}^{k+r \times k+r}$

Diagonal matrix of ± 1s, $R \in \mathbf{R}^{k+r \times k+r}$ such that

$$(I + \bar{Y}\, \bar{T}\, \bar{Y}^{\mathrm{T}}) I_{k+r} = (I + YTY^{\mathrm{T}}) I_{\ell+r} \begin{bmatrix} Q_k^{(H_{mr})} & 0 \\ 0 & I_{r \times r} \end{bmatrix} R$$

1. Compute first $k + r$ rows and columns: $\bar{T}_1 := I_{k+r}^{\mathrm{T}} (I + YTY^{\mathrm{T}}) I_{\ell+r} \begin{bmatrix} Q_k^{(H_{mr})} & 0 \\ 0 & I_{r \times r} \end{bmatrix}$
2. Set $\bar{Y}(1 : k + r, 1 : k + r) = \bar{T}_1$
3. Compute Householder vectors for $\bar{Y}(1 : k + r, 1 : k + r)$

(a) for $i = 1, \ldots, k + r$

(b) $R(i, i) = \text{sign}(\bar{Y}(i, i))$

(c) $\alpha = 1 + R(i, i)\bar{Y}(i, i)$

(d) $\bar{Y}(i, i) = \bar{Y}(i, i) + R(i, i)$

(e) $D(i, i) = 1/\bar{Y}(i, i)$

(f) $\bar{Y}(1 : k + r, i + 1 : k + r) = \bar{Y}(1 : k + r, i + 1 : k + r) - \frac{R(i,i)}{\alpha}\bar{Y}(1 : k + r, i)\bar{Y}(i, i + 1 : k + r)$

(g) end

4. Set $\bar{Y}(1 : k + r, 1 : k + r) = \bar{Y}(1 : k + r, 1 : k + r)D$ and $R = -R$
5. Compute $\bar{T} = \bar{Y}(1 : k + r, 1 : k + r)^{-1}(\bar{T}_1 R - I)\bar{Y}(1 : k + r, 1 : k + r)^{-T}$
6. Set $U = \begin{bmatrix} Q_k^{(H_{mr})} & 0 \\ 0 & I_{r\times r} \end{bmatrix} R(\bar{T}\bar{Y}(1 : k + r, 1 : k + r))^{-1}$

(a) $\bar{Y}(k + r + 1 : n, 1 : k + r) = Y(k + r + 1 : n, 1 : m_1 + r)TY(1 : m_1 + r, 1 : m_1 + r)U$

(b) $\bar{Y}(k + r + 1 : k + r, 1 : k + r) = U(k + r + 1 : k + r, 1 : k + r) + \bar{Y}(k + r + 1 : k + r, 1 : k + r)$

Using Algorithm 3.1 we have,

$$(I + \bar{Y}\bar{T}\bar{Y}^{\mathrm{T}})I_{k+r} = [Q_k^{(A)} V_{(m+1)}]R, \tag{3.8}$$

where the R matrix is a diagonal matrix of ± 1s created in step 3 of Algorithm 3.1 to avoid numerical cancellation.

Multiplying (3.7) by $\widehat{R} := R_{(k,k)} \in \mathbf{R}^{k\times k}$, the first k columns and rows of R, from the right and using (3.8) we have,

$$\begin{aligned} AQ_k^{(A)}\widehat{R} &= [Q_k^{(A)} \;\; V_{(m+1)}]RR \begin{bmatrix} U_k^{(A)} \\ H_{(m+1,m)}E_r^{\mathrm{T}}Q_k^{(H_{mr})} \end{bmatrix} \widehat{R} \\ &= (I + \bar{Y}\bar{T}\bar{Y}^{\mathrm{T}})I_{k+r} \left[R \begin{bmatrix} U_k^{(A)} \\ H_{(m+1,m)}E_r^{\mathrm{T}}Q_k^{(H_{mr})} \end{bmatrix} \widehat{R} \right] \\ &= (I + \bar{Y}\bar{T}\bar{Y}^{\mathrm{T}})I_{k+r}\bar{H}_{k+r} \end{aligned} \tag{3.9}$$

where

$$\bar{H}_{k+r} = \left[R \begin{bmatrix} U_k^{(A)} \\ H_{(m+1,m)}E_r^{\mathrm{T}}Q_k^{(H_{mr})} \end{bmatrix} \widehat{R} \right]. \tag{3.10}$$

Let $Q_k^{(A)}\widehat{R} = [q_1, q_2, \ldots, q_k]$. The block Householder Algorithm can now be continued with the next set of r vectors

$$V_{(m+1)} := (I + \bar{Y}\bar{T}\bar{Y}^{\mathrm{T}}) \begin{bmatrix} 0 \\ I \\ 0 \end{bmatrix} \begin{matrix} \} \in \mathbf{R}^{k\times r} \\ \} \in \mathbf{R}^{r\times r} \\ \} \in \mathbf{R}^{(n-(k+r))\times r} \end{matrix}$$

such that

$$(I + \bar{Y}\bar{T}^{\mathrm{T}}\bar{Y}^{\mathrm{T}})[Aq_1, Aq_2, \ldots, Aq_k, AV_{m+1}] = \begin{bmatrix} \begin{bmatrix} & \\ \bar{H}_{k+r} & \\ & \end{bmatrix} & \begin{matrix} \bar{H}_{(1,k+r+1)} \\ \vdots \\ \bar{H}_{(k+r+1,k+r+1)} \\ \bar{H}_{(k+r+2,k+r+1)} \end{matrix} \\ 0 & 0 \end{bmatrix},$$

where we have computed the Householder QR-decomposition of $X(k+r+1:n, 1:r) := QR$ for $X := (I + \bar{Y}\bar{T}^{\mathrm{T}}\bar{Y}^{\mathrm{T}})AV_{(m+1)}$ to get

$$Q = (I + WSW^{\mathrm{T}}) \begin{bmatrix} I \\ 0 \end{bmatrix} \begin{matrix} \} \in \mathbf{R}^{r\times r} \\ \} \in \mathbf{R}^{(n-r)\times r} \end{matrix} \tag{3.11}$$

and

$$\begin{bmatrix} \bar{H}_{(1,k+r+1)} \\ \vdots \\ \bar{H}_{(k+r+1,k+r+1)} \end{bmatrix} := \left.\begin{bmatrix} \\ X(1:k+r, 1:r) \\ \\ \end{bmatrix}\right\} \in \mathbf{R}^{k+r+1\times r},$$

$$[\bar{H}_{(k+r+2,k+r+1)}] \qquad [\qquad R \qquad]\} \in \mathbf{R}^{r\times r}. \tag{3.12}$$

Matrices $\bar{Y}$ and $\bar{T}$ are updated with W and S from (3.11) to get the next set of vectors and the restarted method is continued. Algorithm 3.2 illustrates the restarting process.

Algorithm 3.2. Block Arnoldi Householder Algorithm, cont

Input: $A \in \mathbf{R}^{n\times n}$;

$\bar{Y} \in \mathbf{R}^{n\times k+r}$, $\bar{T} \in \mathbf{R}^{k+r\times k+r}$, $\bar{H}_{k+r}$, and m_1

Output: $\bar{Y} \in \mathbf{R}^{n\times k+r+m_1 r+r}$, $\bar{T} \in \mathbf{R}^{k+r+m_1 r+r\times k+r+m_1 r+r}$,

$\bar{H}_{k+r+m_1 r+r}$; $\bar{H}_{(k+r+i,k+r+j)} \in \mathbf{R}^{r\times r}$, $j = 1, \ldots m_1$, $i = 1, \ldots, j+1$;

(1) for $j = 1, 2, \ldots, m_1$

(2) Compute $X := (I + \bar{Y}\bar{T}^{\mathrm{T}}\bar{Y}^{\mathrm{T}})A(I + \bar{Y}\bar{T}\bar{Y}^{\mathrm{T}}) \begin{bmatrix} 0 \\ I \\ 0 \end{bmatrix} \begin{matrix} \} \in \mathbf{R}^{k+(j-1)r\times r} \\ \} \in \mathbf{R}^{r\times r} \\ \} \in \mathbf{R}^{(n-jr-k)\times r} \end{matrix}$

(3) Compute the Householder QR-decomposition, where

$X(k+jr+1:n, 1:r) = QR$ and $Q = (I + WSW^{\mathrm{T}}) \begin{bmatrix} I \\ 0 \end{bmatrix} \begin{matrix} \} \in \mathbf{R}^{r\times r} \\ \} \in \mathbf{R}^{(n-r)\times r} \end{matrix}$

(4) (a) Set $\left\{ \begin{array}{l} \begin{bmatrix} \bar{H}_{(1,k+r+j)} \\ \vdots \\ \bar{H}_{(k+r+j,k+r+j)} \end{bmatrix} := \left.\begin{bmatrix} \\ X(1:k+jr, 1:r) \\ \\ \end{bmatrix}\right\} \in \mathbf{R}^{k+jr\times r} \\ [\bar{H}_{(k+r+j+1,k+r+j)}] \quad [\qquad R \qquad]\} \in \mathbf{R}^{r\times r} \end{array} \right.$

(b) Set $\left\{ \begin{array}{l} W := \begin{bmatrix} 0 \\ W \end{bmatrix} \begin{matrix} \} \in \mathbf{R}^{k+jr\times r} \\ \} \in \mathbf{R}^{n-jr-k\times r} \end{matrix} \\ T := \begin{bmatrix} T & TY^{\mathrm{T}}WS \\ 0 & S \end{bmatrix} \in \mathbf{R}^{k+(j+1)r\times k+(j+1)r} \\ Y := [Y \quad W] \in \mathbf{R}^{n\times k+(j+1)r} \end{array} \right.$

(5) end

After m_1 steps where m_1 is chosen so that $k + r + m_1 r \leqslant mr$ we have the following relationship analogous to (2.1)–(2.3),

$$\begin{aligned}\bar{V}_{k+r+m_1 r+r} &= [q_1, \ldots, q_k, V_{(m+1)}, \bar{V}_{(1)}, \ldots, \bar{V}_{(m_1+1)}] \\ &= (I + \bar{Y}\bar{T}\bar{Y}^{\mathrm{T}}) I_{k+r+m_1 r+r} \end{aligned} \tag{3.13}$$

and

$$\begin{aligned} A\bar{V}_{k+r+m_1 r} &= \bar{V}_{k+r+m_1 r+r}\bar{H}_{k+r+m_1 r+r} \\ &\text{or} \\ A\bar{V}_{k+r+m_1 r} &= \bar{V}_{k+r+m_1 r}\bar{H}_{k+r+m_1 r} + \bar{V}_{(m_1+1)}\bar{H}_{(k+r+m_1+1,k+r+m_1)}E_r^{\mathrm{T}}. \end{aligned} \tag{3.14}$$

where

$$\bar{H}_{k+r+m_1 r+r} = \begin{bmatrix} & & \bar{H}_{(1,k+r+1)} & \cdots & \bar{H}_{(1,k+r+m_1)} \\ & & \bar{H}_{(2,k+r+1)} & & \\ & \bar{H}_{k+r} & \vdots & \cdots & \vdots \\ & & \bar{H}_{(k+r+1,k+r+1)} & & \\ & & \bar{H}_{(k+r+2,k+r+1)} & & \\ & & & \ddots & \bar{H}_{(k+r+m_1,k+r+m_1)} \\ & 0 & & & \bar{H}_{(k+r+m_1+1,k+r+m_1)} \end{bmatrix}.$$

Algorithm 3.3 combines the results and outlines our restarted method.

Algorithm 3.3. Augmented Block Arnoldi Householder Algorithm

Input: $A \in \mathbf{R}^{n\times n}$, k, m, r, tol, such that $(m-1)r \geqslant k$;
Output: eigenvalues $\{\lambda_j\}_{j=1}^k$ and eigenvectors $\{x_j\}_{j=1}^k$ of A;

1. Perform m steps of the block Arnoldi Householder Algorithm 2.1 to get the block Arnoldi decomposition (2.3);
2. Compute and sort the Real Schur decomposition (3.1) of the matrix H_{mr} or $\bar{H}_{k+r+m_1 r+r}$;
3. Check convergence of the k desired eigenvalues using (3.6)
   (a) if all k values converge then compute $\{\lambda_j\}_{j=1}^k$ and $\{x_j\}_{j=1}^k$ using (3.4) and exit;
4. Compute restarting vectors (3.8) using Algorithm 3.1;
5. Compute matrix $\bar{H}_{k+r,k}$, (3.10);
6. Perform m_1 steps of the block Arnoldi Householder Algorithm 3.2 to get the block Arnoldi decomposition (3.14);
7. Go to step 2;

During the iterations of the algorithm desired Schur vectors converge at different rates. Once a Schur vector converges, it can either be computed and stored, *hard locking*, or simply left alone, *soft locking*. Hard locking requires the algorithm to store the converged Schur vectors and orthogonalize them against all future generated Krylov subspaces. Soft locking, which is not locking the Schur vectors, refers to continuously updating the Schur vectors regardless of residual values. Hard locking has the benefit of reducing the overall computational cost, however if the

space spanned by converged Schur vectors is not computed accurately enough then orthognalizing against it could slow down convergence, see Stathopoulos [45] for some details. The MATLAB code `irbleigs` [9] implements hard locking and often has difficulty in computing a large number of desired eigenvalues, e.g. [51] and Example 1 in Section 5. For this reason we have decided to use soft locking (i.e. nonlocking) in our computer code `ahbeigs` even though it is more computational expensive than hard locking.

4. Software `ahbeigs`

The algorithms presented in this paper are implemented in the MATLAB program `ahbeigs`.[4] The program was written in MATLAB because of its portability and ease of use. The drawback to a program written entirely in MATLAB is that it can be significantly slower (CPU-time) than the same program written in FORTRAN. MATLAB is a gateway to the FORTRAN subroutines in LAPACK [1] and whenever possible we use MATLAB's built-in internal functions, e.g. MATLAB's `qr` function is used to get the Householder vectors in step 3 of Algorithm 2.1. However, not all routines that are needed in the algorithms in this paper are accessible via built-in internal functions, e.g. the Schur reordering routine `dtrsen` is not accessible via a built-in internal function. A link to the LAPACK routines can be created with MATLAB using MEX files; see [31]. This would maintain a fast CPU-time but decreases the portability and ease of use of the program. Therefore, we use only MATLAB syntax and any built-in internal functions whenever possible.

The actual implementation of the algorithms presented in the paper have been implemented with a small adjustment that exploits the approaches recently advocated in Lehoucq [24] and Baglama and Reichel [10,11] for faster convergence. The technique, which is not new, is to have the number of augmenting vectors ℓ used at each restart to be larger than the number of desired eigenpairs k. For example, if an user wants k eigenpairs, the program `ahbeigs` will search for $\ell = k + adjust$ eigenpairs. The value $adjust$ is automatically increased by the number of converged desired eigenvectors during the iterations. However, the maximum storage requirement remains fixed, that is the maximum number of vectors of length n is always $\leqslant blsz \cdot (nbls + 1)$. Convergence is determined by setting tol in (3.6) to be the machine epsilon. The initial value of $adjust$ is set at 3, see Table 4.1.

The computer code `ahbeigs` can also solve the generalized eigenvalue problem

$$Ax = \lambda Bx \quad A, B \in \mathbf{R}^{n\times n} \tag{4.1}$$

and find eigenvalues located near an input numeric value NVAL, see $sigma$ in Table 4.1. If the user inputs a numeric value NVAL or wishes to solve the generalized eigenvalue problem (4.1) the computer code `ahbeigs` will use the transformation

$$(A - (NVAL)\ B)^{-1}Bx = \theta x \tag{4.2}$$

where $\theta = 1/(\lambda - (NVAL))$, see [17] for details. The matrix $(A - (NVAL)\ B)^{-1}$ is factored using MATLAB's built-in internal function `lu`. If the matrix B is positive definite and $sigma$ is nonnumeric then MATLAB's built-in Cholesky factorization `chol` is used to compute the upper triangular Cholesky factor R to get $B = R^{\mathrm{T}}R$. The generalized eigenvalue problem (4.1) can be a transformed into a standard eigenvalue problem by setting, $A := R^{-T}AR^{-1}$. If the standard eigenvalue problem (1.1) is to be solved where $sigma$ is a nonnumeric input value then no factorization is required and only matrix–vector products with A are used.

[4] Computer code can be downloaded from http://www.math.uri.edu/∼jbaglama or http://www.mathworks.com/matlabcentral/fileexchange.

Table 4.1
Parameters for `ahbeigs.m`

$adjust$	Initial number of vectors added to the k restart vectors to speed up convergence. Default value: $adjust = 3$
$blsz$	Block size of block Arnoldi Hessenberg matrix, H_{mr}. The parameter specifies the value of r in (2.2)–(2.3). Default value: $blsz = 3$
$cholb$	Indicates if the Cholesky factorization of the matrix B is available. If the Cholesky factorization matrix R is available then set $cholb = 1$ and replace the input matrix B with R where $B = R^{\mathrm{T}}R$. Default value: $cholb = 0$
$dispr$	When $dispr > 0$, available approximations of the k desired eigenvalues and norms of associated residual errors are displayed each iteration; $dispr = 0$ inhibits display of these quantities. Default value: $dispr = 0$
$permB$	Permutation vector for the Cholesky factorization of $B(permB, permB)$. When the input matrix B is replaced with R where $B(permB, permB) = R^{\mathrm{T}}R$ then the vector $permB$ is the permutation vector. Default value: $permB = 1 : N$
k	Number of desired eigenvalues. Default value: $k = 6$
$maxit$	Maximum number of restarts. Default value: $maxit = 100$
$nbls$	Number of blocks in the block Arnoldi Hessenberg matrix, H_{mr}. This parameter specifies the largest value of m in (2.2)–(2.3). If value of $nbls$ is not sufficiently large enough then `ahbeigs` will not converge or miss some desired eigenvalues. Default value: $m = 10$
$sigma$	Two letter string or numeric value specifying the location of the desired eigenvalues 'LM' or 'SM' Largest or Smallest magnitude 'LR' or 'SR' Largest or Smallest real part 'LI' or 'SI' Largest or Smallest imaginary part 'LA' or 'SA' Largest or Smallest algebraic (symmetric problems only) NVAL A numeric value. The program searches for the k closest eigenvalues to the numeric value NVAL. (`ahbeigs` will factor the matrix A, see (4.2).) Default value: $sigma$ ='LM'
tol	Tolerance used for convergence (3.6). Default value: $tol = 10^{-6}$
V_0	Initial matrix of r columns for the block Arnoldi Method, Algorithm 2.1. Default value: $V_0 = randn$

The MATLAB function `ahbeigs` requires certain user-specified parameters to be set. Table 4.1 describes these parameters, their possible values, as well as their default values. The function will use the default values unless a user specifies otherwise.

The input sequence of `ahbeigs` is given as

`ahbeigs(A, OPTS)` or `ahbeigs('Afunc', n, OPTS)` or `ahbeigs(A, B, OPTS)`

where the first input argument must be the matrix A which can be passed as a numeric matrix or as a M-file ('Afunc') that computes the product $A \cdot X$ where X is a $n \times r$ matrix. If A is passed as a M-file then the second input argument n is the size of the matrix A. For the generalized eigenvalue problem (4.1) the matrices A and B must be numeric. The last input value $OPTS$ is a structure array with the field values as parameter names. The input parameters can be given in any order and the structure $OPTS$ may contain some or all of the input parameters. The string for the input parameters can contain upper or lower case characters.

The output options are given as

`ahbeigs(...)`

Displays the desired eigenvalues.

`D=ahbeigs(...)`

Returns the desired eigenvalues in the vector D.

`[X,D]=ahbeigs(...)`

D is a diagonal matrix that contains the desired eigenvalues along the diagonal and the matrix X contains the corresponding eigenvectors, such that $AX = XD$ or $AX = BXD$.

`[X,D,FLAG]=ahbeigs(...)`

Returns the same as the above option plus a two dimensional array $FLAG$ that reports if the algorithm converges and the number of matrix vector products. If $FLAG(1) = 0$, this implies normal return and all eigenvalues have converged. If $FLAG(1) = 1$, then the maximum number of iterations have been reached before all desired eigenvalues have converged. $FLAG(2)$ contains the number of matrix vector products used by the code. If the maximum number of iterations are reached then the matrices X and D contain any eigenpairs that have converged plus the computed Ritz pair approximations from the last iteration for the eigenpairs that have not converged.

The following are the MATLAB commands used to determine the 4 eigenvalues of smallest magnitude and corresponding eigenvectors of a matrix A using a block size 4 and tolerance of 10^{-12},

```
>> OPTS.sigma='SM';
>> OPTS.k=4;
>> OPTS.blsz=4;
>> OPTS.tol=1d-12;
>> [X,D]=ahbeigs(A,OPTS);
```

5. Numerical examples

In this section we provide examples to illustrate the performance of the code `ahbeigs`. We will make direct comparisons with `irbleigs` [5] [9], `eigifp`[6] [19], `jdqr`[7] [18], `jdqz`[8] [18] and MATLAB's internal function `eigs`. The codes `irbleigs` and `eigifp` are only for symmetric eigenvalue problems and will only be used in Example 1. We will also make indirect comparisons with the number of matrix–vector products reported in the block papers [22,25,33]. We will refer to the two methods presented in [33] by `Möller(S)` and `Möller(L)`. The method presented in [25] as `bIRAM` and the method in [22] as `Jia`. Currently, there are no public domain computer codes available for these methods. We will not compare CPU times when referring to results from the block papers [22,25,33]. Both methods `jdqr` and `jdqz` are designed to be used with a preconditioner. When a good preconditioner is known these methods are highly competitive. However, we will assume no good preconditioner is known for the examples presented in this paper and use the methods `jdqr` and `jdqz` unpreconditioned.

The MATLAB function `eigs`, which uses ARPACK, is a FORTRAN code except for a small amount of MATLAB syntax for parsing input/output and handling matrix–vector products via the reverse communication feature of ARPACK. Since the majority of the code for `eigs` is written in FORTRAN it yields the shortest total CPU times when compared with other similar methods written entirely in MATLAB. Note that the CPU times on average for the matrix–vector products in the examples is less for `ahbeigs` even though the number of matrix–vector products is often more. This is due to the fact that multiplying a matrix by a group of vectors is often faster than multiplying by only one vector at a time, an advantage of a block routine.

The methods behind the codes `eigs` and `ahbeigs` (with block size $r = 1$) are mathematically equivalent, however the codes will rarely yield similar number of matrix–vector products for a given matrix even with the same input vector and common parameters set equal. This is due in part to how the author(s) of the code implements the method, e.g. orthogonalization of the Arnoldi

[5] Computer code is available at http://www.math.uri.edu/∼jbaglama or http://math.nist.gov/toms/.

[6] Computer code is available at http://www.ms.uky.edu/∼qye/eigifp.html.

[7] Computer code is available at http://www.math.uu.nl/people/sleijpen/index.html.

[8] Computer code is available at http://www.math.uu.nl/people/sleijpen/index.html.

vectors (Householder, Gram-Schmidt with full or partial re-orthogonalization), what tolerance to use to reorthogonalize the Arnoldi vectors, which method of deflation to use and when to deflate, when and how to adjust the number of augmenting vectors ($adjust$) for `ahbeigs` or the number of shifts for `eigs` to avoid stagnation or increase convergence, and what criteria to use for convergence.

We remark that in order to achieve the least total CPU-time for a block method depends on the combination of the block size and number of blocks, the problem at hand, and on the architecture of the computer used. For many problems, the main advantage of using block size $r > 1$ is increased reliability, see Example 1.

In the computed examples, we determine the initial block V_0 by orthonormalizing the columns of an $n \times r$ matrix with normally distributed random entries. The initial vector for the non-block routines `jdqr`, `eigifp`, `jdqz`, and `eigs` was chosen to be the first column of V_0. There are numerous choices and combinations of parameter values for each of the methods. Some choices and combinations yield faster convergence than others. For the direct comparisons we will use the default values and only change block size, location of eigenvalues desired, tolerance, and storage requirements.

In all examples, except example 6, the matrix A was accessed only by calls to a function with input $X \in \mathbb{R}^{n\times blsz}$ and output AX. This approach is "matrix-free" in the sense that the matrix A does not have to be stored.

All computations for direct comparisons were carried out using MATLAB version 7.3.0.267 (R2006b) on a Dell 530 workstation with two 2.4 GHz (512k cache) Xeon processors and 2 GB (400 MHz) of memory running under the Windows XP operating system. Machine epsilon is $\epsilon = 2.2 \times 10^{-16}$.

Example 1. Let $A \in \mathbb{R}^{1600\times 1600}$ be obtained by discretizing the 2-dimensional negative Laplace operator on the unit square by the standard 5-point stencil with Dirichlet boundary conditions. The MATLAB command

$$A = \texttt{delsq(numgrid('S', 42))} \tag{5.1}$$

determines this matrix. We will compare the MATLAB programs `ahbeigs`, `jdqr`, `eigifp`, `eigs`, and `irbleigs` for the computation of the three smallest eigenvalues and again for the 100 smallest eigenvalues. The eigenvalues of the matrix A are well-known [47, Section 8.4] and the largest multiplicity of the 100 smallest eigenvalues is two. We would like the computed Ritz values to satisfy (3.6) with $tol = 10^{-12}$. Non-block methods require a smaller tolerance in order to compute the desired eigenvalues with proper multiplicity, see e.g. [3,7,8,25,26]. This requirement of a smaller tolerance for a non-block method is demonstrated with `ahbeigs` with block size 1. In Table 4.1 we set the parameters $blsz = 1, nbls = 20, k = 3, sigma =$ 'SA', $tol = 10^{-12}$, and left all other parameters at the default value. The smallest eigenvalue of A has multiplicity one, however the second and third smallest eigenvalues of A coincide. The graphs in Fig. 5.1 show the convergence of the three smallest Ritz values. The top graph is the convergence of the smallest Ritz value and the bottom graph illustrates the convergence of the next two smallest Ritz values. Notice Ritz(3) does not converge to the multiple eigenvalue until the residual is small $\approx 10^{-12}$. The big spike in the bottom graph for Ritz(3) is the introduction of the new poorly approximated Ritz vector for the multiple eigenvalue. Experiments showed that the methods `jdqr`, `eigifp`, and `eigs` all occasionally missed multiple eigenvalues with tolerance settings above 10^{-12}. Therefore, to ensure proper calculation of the multiplicity for the nonblock methods (`jdqr`, `eigifp`, `eigs`) we set the tolerance for all methods to be $tol = 10^{-12}$. We set the appropriate parameters in all

Table 5.1
Example 1: ($k = 3$) 2-dimensional negative Laplace operator

Method	Block size	# of blocks	# mvps	CPU time (s)		$\|AQ_k^{(A)} - Q_k^{(A)}U_k^{(A)}\|_2$
				mvps	Total	
`ahbeigs`	1	20	472	0.233	1.53	$O(10^{-14})$
$tol = 10^{-12}$	2	10	932	0.295	2.39	$O(10^{-14})$
`ahbeigs`	2	10	762	0.141	1.94	$O(10^{-12})$
$tol = 10^{-10}$	2	20	516	0.078	1.95	$O(10^{-12})$
`ahbeigs`	2	10	636	0.095	1.78	$O(10^{-10})$
$tol = 10^{-8}$	2	20	448	0.096	1.78	$O(10^{-10})$
`irbleigs`	2	10	1040	0.234	2.03	$O(10^{-8})$
`eigifp`	1	20	907	0.41	1.69	$O(10^{-13})$
`jdqr`	1	20	637	0.186	1.48	$O(10^{-13})$
`eigs`	1	20	585	0.31	0.86	$O(10^{-14})$

The advantage of using `ahbeigs` allows for a lower tolerance without missing a multiple eigenvalue. $tol = 10^{-12}$ was used for all other routines.

Table 5.2
Example 1: ($k = 100$) 2-dimensional negative Laplace operator

Method	Block size	# of blocks	# mvps	CPU time (s)		$\|AQ_k^{(A)} - Q_k^{(A)}U_k^{(A)}\|_2$
				mvps	Total	
`ahbeigs`	1	120	1180	0.563	88.1	$O(10^{-13})$
$tol = 10^{-12}$	2	60	914	0.21	42.2	$O(10^{-13})$
`ahbeigs`	2	60	860	0.248	38.7	$O(10^{-11})$
$tol = 10^{-10}$	2	120	896	0.203	31.9	$O(10^{-14})$
`ahbeigs`	2	60	824	0.219	36.1	$O(10^{-9})$
$tol = 10^{-8}$	2	120	772	0.235	26.8	$O(10^{-8})$
`irbleigs`	2	60	44066	12.45	320.2	$O(10^{-8})$
`eigifp`	1	120	22244	10.89	150.5	$O(10^{-13})$
`jdqr`	1	120	9737	3.70	203.0	$O(10^{-13})$
`eigs`	1	120	887	0.231	10.9	$O(10^{-13})$

methods such that maximum storage requirement is 20 vectors and that the program searches for the 3 smallest eigenvalues of A. All other parameters were left at the default values. Table 5.1 shows the number of matrix–vector products, CPU-time for the matrix–vector products, total CPU-time, and the error. Setting the parameters for the number of desired eigenvalues to be 100 and the maximum storage requirement to 120 vectors, Table 5.2 shows the number of matrix–vector products, CPU-time for the matrix–vector products, total CPU-time, and the error. This example shows that the MATLAB program `ahbeigs` is competitive with other available software and can compute multiple eigenvalues efficiently.

Example 2. We consider the matrix CK656[9] from the Non-Hermitian Eigenvalue Problem (NEP) Collection [12]. This is a 656×656 real nonsymmetric matrix with 3884 nonzero entries. It is known to have eigenvalues of multiplicity two. According to [12] the goal is to compute the eigenvalues with magnitude greater than one. There are 22 eigenvalues of CK656 with magnitude

[9] Matrix available at http://math.nist.gov/MatrixMarket/.

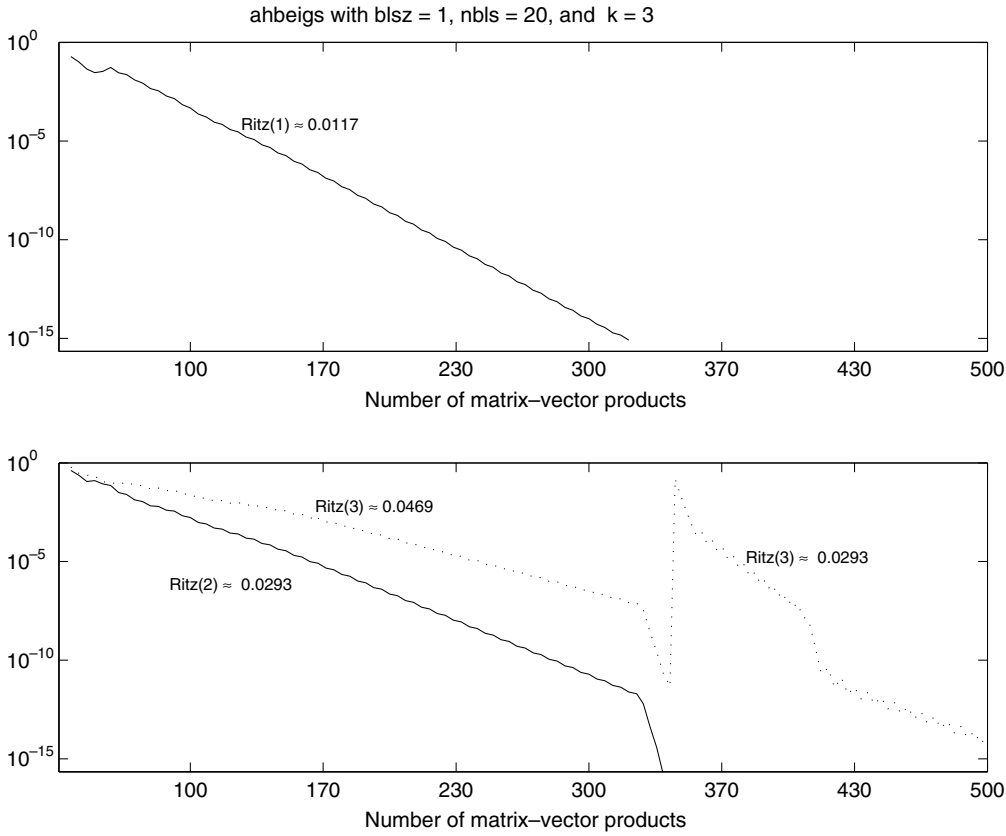

Fig. 5.1. Example 1: The residuals calculated via (3.6) of the 3 smallest Ritz values for the matrix (5.1). The bottom graphs shows that the program `ahbeigs` with block size 1 has difficulty approximating a multiple eigenvalue.

greater than one. As in Example 1, we must set the tolerance low so that non-block methods, `jdqr` and `eigs` will compute the desired eigenvalues with the proper multiplicity. We set the tolerance for all methods to be 10^{-12}. We set the appropriate parameters in all methods such that maximum storage requirement is 72 vectors and that the program searches for the 22 eigenvalues of A with largest magnitude. All other parameters were left at the default values. We also recorded the matrix–vector products for the methods `Möller(S)` and `Möller(L)` [33]. Table 5.3 displays the results.

Example 3. We consider the matrix HOR131[10] from the from Harwell-Boeing Sparse Matrix Collection [16]. This is a 434×434 real nonsymmetric matrix with 4710 nonzero entries. Goal is to compute the 8 eigenvalues of largest real part. We set the appropriate parameters in all methods such that maximum storage requirement is 24 vectors and that the programs search for the 8 eigenvalues of A with largest real part. We set the tolerance for all methods to be 10^{-12}. We also recorded the matrix–vector products for the methods `Möller(S)` and `Möller(L)` [33]

[10] Matrix available at http://math.nist.gov/MatrixMarket/.

Table 5.3
Example 2: Finding the 22 eigenvalues of largest magnitude for the matrix CK656 with tolerance set at 10^{-12}

Method	Block size	# of blocks	# mvps	CPU time (s)		$\|AQ_k^{(A)} - Q_k^{(A)}U_k^{(A)}\|_2$
				mvps	Total	
ahbeigs	1	72	435	0.020	2.64	$O(10^{-14})$
	2	36	460	0.016	1.76	$O(10^{-13})$
	3	24	663	0.047	2.16	$O(10^{-13})$
	4	18	876	0.078	2.31	$O(10^{-13})$
jdqr	1	72	1440	0.298	25.3	$O(10^{-13})$
eigs	1	72	433	0.022	0.83	$O(10^{-14})$
Möller(S)	1	72	408			
	2	36	456			
	4	18	840			
Möller(L)	1	72	428			
	2	36	478			
	3	24	669			
	4	18	814			

Table 5.4
Example 3: Finding the 8 eigenvalues of largest real part for the matrix HOR131 with tolerance set at 10^{-12}

Method	Block size	# of blocks	# mvps	CPU time (s)		$\|AQ_k^{(A)} - Q_k^{(A)}U_k^{(A)}\|_2$
				mvps	Total	
ahbeigs	1	24	67	0.00	0.407	$O(10^{-15})$
	2	12	96	0.00	0.160	$O(10^{-13})$
	3	8	168	0.00	0.183	$O(10^{-14})$
	4	6	288	0.00	0.306	$O(10^{-13})$
jdqr	1	24	175	0.03	1.172	$O(10^{-13})$
eigs	1	24	74	0.00	0.353	$O(10^{-15})$
bIRAM	1	24	77			
	2	12	84			
	3	8	99			
	4	6	108			
Möller(S)	1	24	88			
	2	12	136			
	4	6	264			
Möller(L)	1	24	79			
	2	12	93			
	4	6	105			

and bIRAM [25]. Table 5.4 displays the results. The CPU-time for matrix–vector products for all methods except jdqr was less than 10^{-3} and hence is displayed as 0 in Table 5.4.

Example 4. We consider the matrix TOLS2000[11] from the Non-Hermitian Eigenvalue Problem (NEP) Collection [12]. This is a 2000×2000 real nonsymmetric matrix with 5184 nonzero entries. The matrix arises in the stability analysis of a model of an airplane in flight. The goal is to compute the eigenvalues with largest imaginary parts. The Tolosa matrix is highly nonnormal and finding the eigenvalues is very difficult numerically. We compare our results with Jia [22]. The

[11] Matrix available at http://math.nist.gov/MatrixMarket/.

Table 5.5
Example 4: Finding the 3 eigenvalues of largest imaginary part for the matrix TOLS2000 with tolerance set at 10^{-9}

Method	Block size	# of blocks	# mvps	CPU time (s)		$\|AQ_k^{(A)} - Q_k^{(A)}U_k^{(A)}\|_2$
				mvps	Total	
`ahbeigs`	1	30	510	0.031	2.43	$O(10^{-7})$
	2	30	1410	0.188	7.76	$O(10^{-7})$
	3	30	1854	0.203	11.59	$O(10^{-9})$
`eigs`	1	30	766	0.078	1.52	$O(10^{-7})$
`Jia`	2	30	1980			
	3	30	1800			

Table 5.6
Example 5: Finding the 10 eigenvalues of largest magnitude for the matrix AF23560 with tolerance set at 2.2×10^{-16}

Method	Block size	# of blocks	# mvps	CPU time (s)		$\|AQ_k^{(A)} - Q_k^{(A)}U_k^{(A)}\|_2$
				mvps	Total	
`ahbeigs`	1	24	134	0.800	8.68	$O(10^{-12})$
	2	12	268	0.875	11.78	$O(10^{-11})$
	3	8	444	1.22	18.45	$O(10^{-10})$
	4	6	592	1.73	21.33	$O(10^{-10})$
`eigs`	1	24	136	0.61	2.48	$O(10^{-10})$
`Möller(S)`	1	24	136			
	2	12	264			
	4	6	600			
`Möller(L)`	1	24	140			
	2	12	156			
	4	6	300			

goal is to compute the 3 eigenvalues of largest imaginary part. We fixed the number of vectors to be 30 and set the tolerance to be 10^{-9} in all methods. Table 5.5 displays the results. `jdqr` did not converge with 30, tolerance set at 30, and all other settings at the default values.

Example 5. We consider the matrix AF23560[12] from the Non-Hermitian Eigenvalue Problem (NEP) Collection [12]. This is a 23560×23560 real nonsymmetric matrix with 484256 nonzero entries and is the largest real nonsymmetric matrix in the collection. The goal is to compute 10 eigenvalues of largest magnitude. We set the tolerance for all methods to be machine precision, i.e $\approx 2.2 \times 10^{-16}$. We set the appropriate parameters in all methods such that maximum storage requirement is 30 vectors and that the program searches for the 10 eigenvalues of A with largest magnitude. We also recorded the matrix–vector products for the methods `Möller(S)` and `Möller(L)` [33]. Table 5.6 displays the results.

Example 6. We consider the matrices BFW782A and BFW782B[13] from the Non-Hermitian Eigenvalue Problem (NEP) Collection [12]. We will solve the generalized eigenproblem . The eigenvalues and corresponding eigenvectors of interest are the ones with positive real parts, which correspond to the propagation modes of a waveguide. The matrix A is nonsymmetric and B is

[12] Matrix available at http://math.nist.gov/MatrixMarket/.
[13] Matrices available at http://math.nist.gov/MatrixMarket/.

Table 5.7
Example 6: Finding the 4 eigenvalues of largest real part for the generalized eigenvalue problem with matrices BFW782A and BFW782B

Method	Block size	# of blocks	Total CPU time (s)	$\|AQ_k^{(A)} - BQ_k^{(A)}U_k^{(A)}\|_2$
ahbeigs	1	48	5.77	$O(10^{-10})$
	2	24	6.97	$O(10^{-10})$
	3	16	10.78	$O(10^{-9})$
	4	12	14.00	$O(10^{-9})$
jdqz	1	48	10.56	$O(10^{-10})$

symmetric indefinite. We set the appropriate parameters in all methods such that maximum storage requirement is 48 vectors and that the program searches for the 4 eigenvalues of A with largest real part. We only can make a comparison with jdqz. The program eigs requires B to be positive definite or for the user to input a matrix–vector product routine to compute $B^{-1}Ax$. Tolerance for ahbeigs was set at 10^{-6} and tolerance for jdqz had to be set to 10^{-12} in order to get the same accuracy. Table 5.7 displays the results.

6. Conclusion

This paper presents a block form of the Arnoldi Householder algorithm and shows how to implement an augmented block Krylov method by utilizing a block form of the Householder vectors. Computed examples illustrate that the MATLAB function ahbeigs is a competitive software that can compute multiple eigenvalues more reliably than a single vector method and in certain examples may determine desired eigenvalues to specified accuracy faster than other public domain MATLAB functions.

Acknowledgments

I would like thank the referees for valuable comments.

References

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen, LAPACK Users' Guide, 3rd ed., SIAM, Philadelphia, PA, 1999.
[2] W.E. Arnoldi, The principle of minimized iterations in the solution of the matrix eigenvalue problem, Quart. Appl. Math. 9 (1951) 17–29.
[3] Z. Bai, D. Day, Q. Ye, ABLE: an adaptive block Lanczos method for non-Hermitian eigenvalue problems, SIAM J. Matrix Anal. Appl. 20 (1999) 1060–1082.
[4] Z. Bai, J. Demmel, On swapping diagonal blocks in real Schur form, Linear Algebra Appl. 186 (1993) 73–95.
[5] J. Baglama, Dealing with linear dependence during the iterations of the restarted block Lanczos methods, Numer. Algs. 25 (2000) 23–36.
[6] J. Baglama, D. Calvetti, L. Reichel, Fast Leja points, Electron. Trans. Numer. Anal. 7 (1998) 124–140.
[7] J. Baglama, D. Calvetti, L. Reichel, A. Ruttan, Computation of a few small eigenvalues of a large matrix with applications to liquid crystal modeling, J. Comput. Phys. 146 (1998) 203–226.
[8] J. Baglama, D. Calvetti, L. Reichel, IRBL: an implicitly restarted block Lanczos method for large-scale Hermitian eigenproblems, SIAM J. Sci. Comput. 29 (5) (2003) 1650–1677.
[9] J. Baglama, D. Calvetti, L. Reichel, irbleigs: A MATLAB program for computing a few eigenpairs of a large sparse Hermitian matrix, TOMS, 29 (5) (2003) 337–348.

[10] J. Baglama, L. Reichel, Augmented implicitly restarted Lanczos bidiagonalization methods, SIAM J. Sci. Comput. 27 (2005) 19–42.
[11] J. Baglama, L. Reichel, Restarted block Lanczos bidiagonalization methods, Numerical Algorithms 43 (2006) 251–272.
[12] Zhaojun Bai, David Day, James Demmel, Jack Dongarra, A test matrix collection for non-Hermitian eigenvalue problems, release 1.0, September 1996.
[13] C. Bischof, Y. Sun, On orthogonal block elimination, Technical Report MCS-P441-0594, Mathematics and Computer Science Division, Argonne National Laboratory, 1994.
[14] J.W. Demmel, Applied Numerical Linear Algebra, SIAM, Philadelphia, PA, 1997.
[15] J.J. Dongarra, J. DuCroz, I.S. Duff, S. Hammarling, A set of level 3 basic linear algebra subprograms, ACM Trans. Math. Software 16 (1990) 1–17.
[16] I.S. Duff, R.G. Grimes, J.G. Lewis, User's Guide for the Harwell-Boeing sparse matrix collection (Release I), Technical Report TR/PA/92/86, CERFACS, Toulouse, France, 1992. Matrices available at <http://math.nist.bov/MatrixMarket/>.
[17] T. Ericsson, A. Ruhe, The spectral transformation Lanczos method for the numerical solution of large sparse generalized symmetric eigenvalue problems, Math. Comput. 35 (1980) 1251–1268.
[18] D.R. Fokkema, G.L.G. Sleijpen, H.A. van der Vorst, Jacobi-Davidson style QR and QZ algorithms for the reduction of matrix pencils, SIAM J. Sci. Comput. 20 (1998) 94–125.
[19] G.H. Golub, Q. Ye, An inverse free preconditioned Krylov subspace method for symmetric generalized eigenvalue problems, SIAM J. Sci. Comput. 24 (2002) 312–334.
[20] R.G. Grimes, J.L. Lewis, H.D. Simon, A shifted block Lanczos algorithm for solving sparse symmetric generalized eigenproblems, SIAM J. Matrix Anal. 15 (1994) 228–272.
[21] G. Gu, Z. Cao, A block GMRES method augmented with eigenvectors, Appl. Math. Comput. 121 (2001) 271–289.
[22] Z. Jia, A refined iterative algorithm based on the block Arnoldi process for large unsymmetric eigenproblems, Linear Algebra Appl. 270 (1998) 171–189.
[23] Z. Jia, Polynomial characterizations of the approximate eigenvectors by the refined Arnoldi method and an implicitly restarted refined Arnoldi algorithm, Linear Algebra Appl. 287 (1998) 191–214.
[24] R.B. Lehoucq, Implicitly restarted Arnoldi methods and subspace iteration, SIAM J. Matrix Anal. Appl. 23 (2001) 551–562.
[25] R.B. Lehoucq, K.J. Maschhoff, Implementation of an implicitly restarted block Arnoldi method, Preprint MCS-P649-0297, Argonne National Laboratory, Argonne, IL, 1997.
[26] R.B. Lehoucq, J.A. Scott, An evaluation of software for computing eigenvalues of sparse nonsymmetric matrices, Technical report MCS-P547-1195, Argonne National Laboratory, 1995.
[27] R.B. Lehoucq, D.C. Sorensen, Deflation techniques for an implicitly restarted Arnoldi iteration, SIAM J. Matrix Anal. Appl. 17 (1996) 789–821.
[28] R.B. Lehoucq, K. Maschhoff, D.C. Sorensen, and C. Yang, ARPACK, http://www.caam.rice.edu/software/ARPACK/ (1998).
[29] R.B. Lehoucq, D.C. Sorensen, C. Yang, ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods, SIAM Publications, Philadelphia, 1998.
[30] O.A. Marques, BLZPACK: A Fortran 77 implementation of the block Lanczos algorithm. Code available from: <http://www.nersc.gov/~osni/marques-software.html>.
[31] The MathWorks, MATLAB Application Program Interface Guide, Version 5, The MathWorks, Inc., Natick, 1998.
[32] K. Meerbergen, J. Scott, The design of a block rational Lanczos code with partial reorthogonalization and implicit restarting, Technical Report RAL-TR-2000-011, 2000.
[33] J. Möller, Implementations of the Implicitly Restarted Block Arnoldi Method, Technical report, Royal Institute of Technology (KTH), Department of Numerical Analysis and Computer Science, TRITA-NA-0446, 2004.
[34] R. Morgan, On restarting the Arnoldi method for large nonsymmetric eigenvalue problems, Math. Comput. 65 (1996) 1213–1230.
[35] R. Morgan, Computing interior eigenvalues of large matrices, Linear Algebra Appl. 154–156 (1991) 289–309.
[36] R. Morgan, GMRES with Deflated Restarting, Technical Report, 2001.
[37] R. Morgan, Restarted block GMRES with deflation of eigenvalues, Appl. Numer. Math. 54 (2005) 222–236.
[38] R. Morgan, M. Zeng, Harmonic projection methods for large non-symmetric eigenvalue problems, Numer. Linear Algebra Appl. 5 (1998) 33–55.
[39] Y. Saad, Numerical Methods for Large Eigenvalue Problems, Halsted Press, New York, 1992.
[40] Y. Saad, Chebyshev acceleration techniques for solving nonsymmetric eigenvalue problems, Math. Comput. 42 (166) (1984) 567–588.

[41] R. Schreiber, C. Van Loan, A storage-efficient WY representation for products of householder transformations, SIAM J. Sci. Stat. Comput. 10 (1989) 53–57.
[42] J.A. Scott, An Arnoldi code for computing selected eigenvalues of sparse real unsymmetric matrices, ACM Trans. Math. Software 21 (1995) 432–475.
[43] D.S. Scott, LASO2 – FORTRAN implementation of the Lanczos process with selective orthogonalization, Code and documentation available from Netlib.
[44] D.C. Sorensen, Implicit application of polynomial filters in a *k*-step Arnoldi method, SIAM J. Matrix Anal. Appl. 13 (1992) 357–385.
[45] A. Stathopoulos, Locking issues for finding a large number of eigenvectors of hermitian matrices, Tech Report: WM-CS-2005-09, July, 2005, Revised June 2006.
[46] G.W. Stewart, A Krylov–Schur algorithm for large eigenproblems, SIAM J. Matrix Anal. Appl. 23 (3) (2001) 601–614.
[47] J. Stoer, R. Bulirsch, Introduction to Numerical Analysis, 2nd ed., Springer, New York, 1993.
[48] X. Sun, C.A. Bischof, Basis-kernel representation of orthogonal matrices, SIMAX 16 (4) (1995) 1184–1196.
[49] H. Walker, Implementation of the GMRES method using Householder transformations, SIAM J. Sci. Comput. 9 (1988) 152–163.
[50] K. Wu, H. Simon, Thick-restart Lanczos method for large symmetric eigenvalue problems, SIAM. J. Matrix Anal. Appl. 22 (2001) 602–616.
[51] Y. Zhou, Y. Saad, Block Krylov–Schur method for large symmetric eigenvalue problems, Technical report Department of Computer Science and Engineering, University of Minnesota, 2004.