RAL-93-097

An Arnoldi code for computing selected eigenvalues of sparse real unsymmetric matrices

by

J. A. Scott

ABSTRACT

Arnoldi methods can be more effective than subspace iteration methods for computing the dominant eigenvalues of a large sparse real unsymmetric matrix. A code, `EB12`, for the sparse unsymmetric eigenvalue problem based on a subspace iteration algorithm, optionally combined with Chebychev acceleration, has recently been described by Duff and Scott (1993) and is included in the Harwell Subroutine Library (Anon 1993). In this paper we consider variants of the method of Arnoldi and discuss the design and development of a code to implement these methods. The new code, which is called `EB13`, offers the user the choice of a basic Arnoldi algorithm, an Arnoldi algorithm with Chebychev acceleration, and a Chebychev preconditioned Arnoldi algorithm. Each method is available in blocked and unblocked form. The code may be used to compute either the right-most eigenvalues, the eigenvalues of largest absolute value, or the eigenvalues of largest imaginary part. The performance of each option in the `EB13` package is compared with that of subspace iteration on a range of test problems and, on the basis of the results, advice is offered to the user on the appropriate choice of method.

Keywords : eigenvalues, sparse unsymmetric matrices, Arnoldi's method, Chebychev polynomials

AMS(MOS) subject classification : 65F15, 65F50.

CR classification system : G.1.3.

Computing and Information Systems Department,
Atlas Centre,
Rutherford Appleton Laboratory,
Oxon OX11 0QX.

March 1995.

CONTENTS

1 Introduction

We are interested in computing a few eigenvalues and the corresponding eigenvectors of a large sparse real unsymmetric matrix. Of particular interest are the eigenvalues with the largest (or smallest) real parts, the eigenvalues of largest absolute value, and the eigenvalues with the largest imaginary parts. This problem is of considerable practical importance since it has a significant number of applications in scientific and engineering computing, including mathematical models in economics, Markov chain modelling of queueing networks, and bifurcation problems (for references, see Saad 1984, the survey of Kerner 1989, and the collection of test matrices of Bai 1993). In the past few years there has been a considerable increase in interest in solving large unsymmetric eigenvalue problems. Foremost in the discussions have been methods based on the original method of Arnoldi (1959). Papers have appeared by, amongst others, Saad (1989), Ho (1990), Ho, Chatelin, and Bennani (1990), Garratt, Moore, and Spence (1991), Garratt (1991), Sadkane (1991a, 1991b), and Sorensen (1992). In conjunction with the published papers, several codes have been developed which implement variations of Arnoldi's method. These include codes by Sadkane (1991a, 1991b) and Braconnier (1993). These codes may be regarded as experimental codes since their use requires details of Arnoldi's method to be understood before appropriate values can be given to the input parameters. At present there is no code which implements an Arnoldi based method for the unsymmetric eigenvalue problem available in the NAG Library (NAG 1993) or the Harwell Subroutine Library (Anon 1993), although a new package ARPACK (Lehoucq, Sorensen, and Vu 1994), which uses the implicit restarted Arnoldi method of Sorensen (1992) has recently been included under the directory **scalapack** in netlib (Dongarra and Grosse 1987).

The current Release 11 of the Harwell Subroutine Library includes a code `EB12` by Duff and Scott (Duff and Scott 1993) which uses a subspace iteration algorithm, optionally combined with Chebychev acceleration, to find either the eigenvalues of largest absolute value or the right-most (or left-most) eigenvalues of a sparse unsymmetric matrix. Since numerical experiments described in Saad (1980, 1984) and Garratt (1991) indicate that Arnoldi methods can be more effective than subspace iteration methods in computing the outermost part of the eigenspectrum, we are interested in designing a new code for the Harwell Subroutine Library which offers Arnoldi based methods. This is the primary goal of this paper.

The paper is organized as follows. In Section 2 we briefly review Arnoldi's method and describe some of its variants within a common framework. In Section 3 we discuss the design and development of our new code which implements these Arnoldi based methods. This code is called `EB13`. In Section 4 the performance of `EB13` on a vector supercomputer and on a high-performance workstation is tested and compared with that of the subspace iteration code `EB12`. Concluding comments are made in Section 5.

2 Arnoldi's method and its variants

2.1 The basic Arnoldi method

Several variants of Arnoldi's method have been proposed in the literature. In this section we will briefly summarize some of these. For completeness, we first outline the basic Arnoldi algorithm. Here and elsewhere $\mathbf{A}$ will denote a real sparse unsymmetric $n \times n$ matrix with eigenvalues $\lambda_1, \lambda_2, ..., \lambda_n$. Arnoldi's method for finding a few eigenvalues of $\mathbf{A}$ proceeds as follows: given an initial vector $\mathbf{x}_1$ with unit norm, at each step m ($m > 1$) construct an orthonormal basis $\mathbf{X}_m = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_m]$ for the Krylov subspace $\mathbf{K}_m$ spanned by $[\mathbf{x}_1, \mathbf{A}\mathbf{x}_1, ..., \mathbf{A}^{m-1}\mathbf{x}_1]$ by computing $\mathbf{w} = \mathbf{A}\mathbf{x}_{m-1}$ and orthonormalising $\mathbf{w}$ with respect to $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{m-1}$ to obtain $\mathbf{x}_m$. The matrix $\mathbf{H} = \mathbf{X}^{\mathrm{T}}\mathbf{A}\mathbf{X}$ is upper Hessenberg and its eigenvalues provide approximations to m eigenvalues of $\mathbf{A}$. It is not possible to state in general which eigenvalue approximations will converge first but the method favours convergence to eigenvalues that lie in the outermost part of the spectrum of $\mathbf{A}$, with rapid convergence of approximations to well separated eigenvalues. If clustered eigenvalues are sought, a large number of steps is generally required. Saad (1980) showed that as the number of steps m increases, the growth in the computational time and the storage required by Arnoldi's method becomes prohibitive. Saad proposed overcoming this difficulty by using the method iteratively so that m is fixed (in general, $m << n$) and the process is restarted every m steps.

It is convenient for introducing variants of Arnoldi's method to split the method into a number of separate steps. Suppose the eigenvalues of $\mathbf{A}$ are ordered with $|\lambda_1| > |\lambda_2| \geq ... \geq |\lambda_n|$ and that λ_1 is required. The main steps in the basic Arnoldi algorithm for finding λ_1 may be summarized as follows:

Algorithm A1: Basic Arnoldi

1. *Initialization:* Choose the number of steps m and an initial vector $\mathbf{x}_1$ with unit norm.
2. *Arnoldi steps:*
```
For $j = 1, 2, ..., m$ do
    (i) $\mathbf{w}_j = \mathbf{A}\mathbf{x}_j$
    (ii) $h_{ij} = \mathbf{x}_i^{\mathrm{T}}\mathbf{w}_j,\ i = 1, 2, ..., j.$
    (iii) $\mathbf{s}_j = \mathbf{w}_j - \sum_{i=1}^{j} \mathbf{x}_i h_{ij}$
    (iv) $h_{j+1,j} = \|\mathbf{s}_j\|_2,\ \mathbf{x}_{j+1} = \mathbf{s}_j / h_{j+1,j}.$
end do
Set $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_m]$.
```
3. *Eigenvalue computation:* Reduce the upper Hessenberg matrix $\mathbf{H} = \{h_{ij}\}$ to real Schur form $\mathbf{T} = \mathbf{Z}^{\mathrm{T}}\mathbf{H}\mathbf{Z}$, where $\mathbf{T}$ is a block triangular matrix and each diagonal block $\mathbf{T}_{ii}$ is either of order 1 or is a 2×2 matrix having complex conjugate eigenvalues, with the eigenvalues ordered in descending order of their absolute values along the diagonal blocks. Set $\mathbf{X} \Leftarrow \mathbf{X}\mathbf{Z}$.
4. *Convergence test and restart:* If the first column $\mathbf{x}_1$ of $\mathbf{X}$ satisfies the convergence criteria then accept λ_1 and **stop** else **go to 2**.

In practice, to overcome the loss of orthogonality of the vectors $\mathbf{x}_j$, the Gram-Schmidt method in Step 2 is combined with iterative refinement (see Daniel, Gragg, Kaufman, and Stewart 1976). We have found that one step of iterative refinement is sufficient for the computation of each $\mathbf{x}_j$ (see also Sorensen 1992). Throughout the remainder of this paper it is assumed that the Gram-Schmidt method with iterative refinement is employed.

Once the eigenvalue λ_1 has been found, the corresponding (approximate) eigenvector $\mathbf{y}_1$ of $\mathbf{A}$ can be determined. The eigenvector $\mathbf{w}_1$ of the matrix $\mathbf{T}$ corresponding to λ_1 is first found using a back-substitution process (Peters and Wilkinson 1970) and then the eigenvector of $\mathbf{A}$ is taken to be $\mathbf{y}_1 = \mathbf{X}\mathbf{w}_1$ (see, for example, Duff and Scott 1993 for further details).

For simplicity, we have described the basic Arnoldi algorithm for the case when only a single eigenvalue is required but it can also be used if the r dominant eigenvalues are sought. In this case, the new starting vector may be taken to be a linear combination of the first r columns $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_r$ of $\mathbf{X}$, that is, at Step 4 take the restart vector to be of the form

$$\beta \sum_{i=1}^{r} \alpha_i \mathbf{x}_i, \tag{1.1}$$

where β is a normalising factor. The α_i's need to be chosen to balance the accuracy of the desired basis vectors. Choosing α_i to be the norm of the residual $\mathbf{A}\mathbf{y}_i - \lambda_i \mathbf{y}_i$ favours the basis vectors which converge slowly. Thus slow convergence is off-set by a starting vector which is richer in the corresponding basis vector. This choice for the α_i's is used by Braconnier (1993). Further details and discussion may be found in Saad (1980,1984).

An alternative approach for computing several eigenvalues is to employ a “locking” technique. The idea behind locking, which is sometimes termed implicit deflation (see Saad 1989), is to exploit the fact that the initial columns of $\mathbf{X}$ tend to converge before the later ones and to use this to reduce the order of the problem. Suppose we want to compute r eigenvalues and the first k–1 basis vectors $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{k-1}$ corresponding to $\lambda_1, \lambda_2, ..., \lambda_{k-1}$ have already converged ($k-1 < r$). Choose $\mathbf{x}_k$ to be orthogonal to each of the vectors $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{k-1}$. Next perform m–k steps of the Arnoldi process, with $\mathbf{x}_k$ as the starting vector and enforcing the orthogonality of $\mathbf{x}_j$, $j = k, k+1, ..., m$ against all $\mathbf{x}_i$, including $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{k-1}$. This generates an orthogonal basis for the subspace spanned by $[\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_k, \mathbf{A}\mathbf{x}_k, ..., \mathbf{A}^{m-k}\mathbf{x}_k]$. Apart from the orthogonalization of the $\mathbf{x}_j$'s ($j \geq k$), no further computations are performed with the converged basis vectors $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{k-1}$ and these basis vectors are therefore said to be “locked”. Since these vectors are locked, the leading $(k-1) \times (k-1)$ submatrix of the Schur matrix $\mathbf{T}$ will not change on subsequent steps and it is necessary only to reduce a Hessenberg matrix of order m–k to real Schur form and to compute its eigenvalues. By avoiding unnecessary recomputations, locking can result in considerable savings in computational time (illustrated by Duff and Scott 1993). In this study we will employ locking rather than using (1.1) whenever more than one eigenvalue (or complex conjugate pair of eigenvalues) is needed.

2.2 Block Arnoldi

Block Arnoldi methods have been discussed by Sadkane (1991a). These methods aim to compute approximations to a block of eigenvalues at once. They can be advantageous when the dominant eigenvalue of $\mathbf{A}$ is a multiple eigenvalue or if the wanted eigenvalues are clustered. The other major advantage of block methods is that they are able to exploit Level 3 BLAS (see Section 3.7), and this can make them competitive with unblocked methods even if only one eigenvalue is sought (see Tables 4.3a and 4.3b in Section 4).

Choosing the blocksize to be n_b ($n_b > 1$), the block Arnoldi method for finding the r eigenvalues of **A** of largest moduli proceeds as follows:

Algorithm AB1: Block Arnoldi

```
1. Initialization: Choose the number of steps $m$ and an initial $m \times n_b$ matrix $\mathbf{X}_1$ with orthonormal
   columns.
2. Block Arnoldi steps:
   For $j = 1, 2, ..., m$ do
       (i) $\mathbf{W}_j = \mathbf{A}\mathbf{X}_j$
      (ii) $\mathbf{H}_{ij} = \mathbf{X}_i^{\mathrm{T}}\mathbf{W}_j$, $i = 1, 2, ..., j$.
     (iii) $\mathbf{S}_j = \mathbf{W}_j - \sum_{i=1}^{j} \mathbf{X}_i \mathbf{H}_{ij}$
      (iv) $\mathbf{Q}_j\mathbf{R}_j = \mathbf{S}_j$ (QR factorization of $\mathbf{S}_j$), $\mathbf{H}_{j+1,j} = \mathbf{R}_j$, $\mathbf{X}_{j+1} = \mathbf{Q}_j$.
   end do
   Set $\mathbf{X} = [\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_m]$.
3. Eigenvalue computation: Reduce the block upper Hessenberg matrix $\mathbf{H}$ = $\{\mathbf{H}_{ij}\}$ to real Schur
   form $\mathbf{T} = \mathbf{Z}^{\mathrm{T}}\mathbf{H}\mathbf{Z}$ , where each diagonal block $\mathbf{T}_{ii}$ is either of order 1 or is a $2 \times 2$ matrix having
   complex conjugate eigenvalues, with the eigenvalues ordered in descending order of their
   absolute values along the diagonal blocks. Set $\mathbf{X} \Leftarrow \mathbf{X}\mathbf{Z}$.
4. Convergence test and restart: If the first $r$ columns of $\mathbf{X}$ satisfy the convergence criteria then
   accept $\lambda_1, \lambda_2, ..., \lambda_r$ and stop else let $\mathbf{X}_1$ be the first $n_b$ columns of $\mathbf{X}$ and go to 2.
```

Again, in practice iterative refinement is used in Step 2 to compensate for any loss in orthogonality.

An obvious choice for the block size is $n_b = r$, the number of required eigenvalues. However, numerical experience has shown it can be advantageous to choose a block size $n_b > r$. This can be the case if the sought-after eigenvalues are not well separated from the remaining eigenvalues. Some results which illustrate this are included in Section 4 (Tables 4.5a and 4.5b).

For problems in which the required eigenvalues can be split into groups of clustered eigenvalues with the groups well separated, the block size should be at least as large as the largest cluster of eigenvalues sought. In this case, n_b will be less than r, and, as in the unblocked method ($n_b = 1$), some work can be saved by locking a whole block of vectors as soon as all the vectors in the block have converged. If the first n_b columns of **X** have converged then, at the next iteration, m–1 block Arnoldi steps are performed with $\mathbf{X}_2$ (the matrix whose columns are the second block of n_b columns of **X**) as the starting matrix.

2.3 Arnoldi with Chebychev acceleration

In practice, the basic Arnoldi method (both blocked and unblocked) can perform badly and the number m of Arnoldi steps and the number of iterations required for convergence can be prohibitively large. To overcome this difficulty and to improve the overall performance of Arnoldi's method, Saad (1984) proposed using the basic Arnoldi method in conjunction with a polynomial filter. In this method, the restart vector is taken to be $p(\mathbf{A})\mathbf{x}_1 / \|p(\mathbf{A})\mathbf{x}_1\|$, where p is a polynomial chosen to amplify the components of $\mathbf{x}_1$ in the direction of the basis vectors corresponding to the required eigenvalues $\lambda_1, \lambda_2, ..., \lambda_r$, while damping those in the directions of the remaining basis vectors. Thus, on each iteration, p is chosen to be small on the set S of unwanted eigenvalues $\{\lambda_{r+1}, \lambda_{r+2}, ..., \lambda_m\}$ and to satisfy the normalization condition

$$p(\lambda_r) = 1. \tag{2.1}$$

A simple procedure is to look for such a polynomial on a continuous domain E containing S but excluding $\lambda_1, \lambda_2, ..., \lambda_r$. The problem of determining p becomes the minimax problem

$$\min_{p \, \varepsilon P_l} \max_{\lambda_i \, \varepsilon E} |p(\lambda)| \, , \tag{2.2}$$

where P_l is the set of all polynomials of degree not exceeding l satisfying (2.1). This problem can be solved if E is restricted to an ellipse with its centre on the real line and containing S. Let $E = E(d, c, a)$ denote an ellipse centre d, foci $d \pm c$, major semi-axis a, and which is symmetric with respect to the real axis. The polynomial solving the minimax problem (2.2) is then

$$p_l(\lambda) = \frac{T_l[(\lambda - d)/c]}{T_l[(\lambda_r - d)/c]} \, , \tag{2.3}$$

where $T_l(\lambda)$ is the Chebychev polynomial of degree l of the first kind. Here λ_r is termed the reference point. Further details may be found, for example, in Manteuffel (1977) and Saad (1984).

Associated with each $\lambda_j \, \varepsilon S$ is a convergence factor

$$\mathbf{C}_j(d, c) = \left| \frac{(\lambda_j - d) + ((\lambda_j - d)^2 - c^2)^{\frac{1}{2}}}{(\lambda_r - d) + ((\lambda_r - d)^2 - c^2)^{\frac{1}{2}}} \right| . \tag{2.4}$$

The choice of d, c, a which give an ellipse $E(d, c, a)$ enclosing all $\lambda_j \, \varepsilon S$ that minimises $\max\limits_{\lambda_j \, \varepsilon S} \mathbf{C}_j(d, c)$ is the optimal ellipse and is used to define the iteration polynomial (2.3). An algorithm for computing the optimal ellipse is described by Ho (1990).

Omitting for the present details of how to construct the optimal ellipse and how to choose the degree of the Chebychev polynomial, the Arnoldi algorithm with Chebychev acceleration for finding the r right-most eigenvalues of $\mathbf{A}$ may then be described as follows:

Algorithm A2: Arnoldi with Chebychev acceleration

1. *Initialization:* See algorithm A1.
2. *Arnoldi steps :* See algorithm A1.
3. *Eigenvalue computation:* Reduce the upper Hessenberg matrix $\mathbf{H} = \{h_{ij}\}$ to real Schur form $\mathbf{T} = \mathbf{Z}^{\mathrm{T}} \mathbf{H} \mathbf{Z}$, where $\mathbf{T}$ is a block triangular matrix and each diagonal block $\mathbf{T}_{ii}$ is either of order 1 or is a 2×2 matrix having complex conjugate eigenvalues, with the eigenvalues ordered in descending order of their real parts along the diagonal blocks. Set $\mathbf{X} \Leftarrow \mathbf{X}\mathbf{Z}$.
4. *Convergence test and restart:* If the first r columns of $\mathbf{X}$ satisfy the convergence criteria then accept $\lambda_1, \lambda_2, ..., \lambda_r$ and **stop** else choose the degree l of the iteration polynomial $p_l(\lambda)$. Construct the ellipse $E(d, c, a)$ containing the unwanted eigenvalues. Define $p_l(\lambda)$ and compute $\hat{\mathbf{x}} = p_l(\mathbf{A})\mathbf{x}_1$, where $\mathbf{x}_1$ is the first column of $\mathbf{X}$. Set $\mathbf{x}_1 \Leftarrow \hat{\mathbf{x}} / \|\hat{\mathbf{x}}\|_2$ and **go to 2**.

As in algorithm A1, locking can be incorporated into algorithm A2 when $r > 1$. In this case, if $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{k-1}$ have converged ($k-1 < r$), then at Step 4, $\hat{\mathbf{x}} = p_l(\mathbf{A})\mathbf{x}_k$ is computed and the restart vector is taken to be the normalised vector $\mathbf{x}_k \Leftarrow \hat{\mathbf{x}} / \|\hat{\mathbf{x}}\|_2$. On the next iteration $m-k$ Arnoldi steps are performed, enforcing the orthogonality of $\mathbf{x}_j, j = k, k+1, ..., m$ against all $\mathbf{x}_i$, including $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{k-1}$. An $(m-k) \times (m-k)$ Hessenberg matrix must be reduced to real Schur form.

By replacing $\mathbf{A}$ by $-\mathbf{A}$, algorithm A2 may be used to compute the left-most eigenvalues of $\mathbf{A}$. Arnoldi's method converges most quickly to the eigenvalues in the outermost part of the eigenspectrum and, in Step 3, by ordering the diagonal entries of $\mathbf{T}$ in descending order of their imaginary parts, algorithm A2 can be used to compute the eigenvalues of largest imaginary parts. In some practical examples it is these eigenvalues which are of practical importance. An example of this is the TOLOSA matrix which arises from the aerodynamics related to the stability analysis of a model of a plane in flight (see Bai 1993).

We remark that the matrix polynomial $p_l(\lambda)$ need not be formed explicitly since to compute $p_l(\lambda)\mathbf{x}$ it is necessary only to form l matrix-vector products with the original matrix $\mathbf{A}$ and take linear combinations. If $p_l(\lambda)$ is defined by (2.3), the three-term recurrence relation for Chebychev polynomials may be used. Since the ellipse $E(d, c, a)$ is symmetric about the real axis, d and c are either purely real or imaginary so that complex arithmetic can be avoided.

2.4 Block Arnoldi with Chebychev acceleration

Algorithm A2 can be generalized to a block algorithm. Choosing the blocksize to be n_b, the block Arnoldi method with Chebychev acceleration for finding the r right-most eigenvalues of $\mathbf{A}$ proceeds as follows:

Algorithm AB2: Block Arnoldi with Chebychev acceleration

1. *Initialization:* See algorithm AB1.
2. *Block Arnoldi steps :* See algorithm AB1.
3. *Eigenvalue computation:* Reduce the block upper Hessenberg matrix $\mathbf{H}$ = $\{\mathbf{H}_{ij}\}$ to real Schur form $\mathbf{T} = \mathbf{Z}^{\mathrm{T}}\mathbf{H}\mathbf{Z}$, where each diagonal block $\mathbf{T}_{ii}$ is either of order 1 or is a 2×2 matrix having complex conjugate eigenvalues, with the eigenvalues ordered in descending order of their real parts along the diagonal blocks. Set $\mathbf{X} \Leftarrow \mathbf{X}\mathbf{Z}$.
4. *Convergence test and restart:* If the first r columns of $\mathbf{X}$ satisfy the convergence criteria then accept $\lambda_1, \lambda_2, ..., \lambda_r$ and **stop** else choose the degree l of the iteration polynomial $p_l(\lambda)$. Construct the ellipse $E(d, c, a)$ containing the unwanted eigenvalues. Define $p_l(\lambda)$ and compute $\hat{\mathbf{X}} \Leftarrow p_l(\mathbf{A})\mathbf{X}_1$, where $\mathbf{X}_1$ is the first n_b columns of $\mathbf{X}$. Orthonormalise the columns of $\hat{\mathbf{X}}$ and let $\mathbf{X}_1$ be the resulting set of orthonormal columns. **Go to 2**.

Some work can be saved by locking vectors within a block as soon as they have converged. Suppose the first $k-1$ vectors have converged where $k-1 < n_b$. Then at Step 4, compute $\hat{\mathbf{X}} \Leftarrow [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{k-1}, p_l(\mathbf{A})\mathbf{x}_k, ..., p_l(\mathbf{A})\mathbf{x}_{n_b}]$, orthonormalise the columns of $\hat{\mathbf{X}}$ (the first $k-1$ columns are already orthonormal) and let the restart matrix $\mathbf{X}_1$ be the resulting set of orthonormal columns. m block Arnoldi steps are performed with this restart matrix and, at Step 3, a block upper Hessenberg matrix of order mn_b must again by reduced to real Schur form. The first k–1 columns of the resulting matrix $\mathbf{X}$ must be rechecked for convergence since they will have been recomputed.

Further computational effort may be saved if $n_b < r$ by locking a whole block of vectors once all the vectors in the block have converged. If all the columns in $\mathbf{X}_1$ have converged then, at Step 4, $\hat{\mathbf{X}} \Leftarrow p_l(\mathbf{A})\mathbf{X}_2$ is computed, where $\mathbf{X}_2$ is the $m \times n_b$ matrix whose columns are the second block of n_b columns of $\mathbf{X}$. The columns of $\hat{\mathbf{X}}$ are orthnormalised with respect to the columns of $\mathbf{X}_1$ and the restart matrix $\mathbf{X}_2$ is taken to be the resulting set of orthonormal columns. On the next iteration it is necessary to perform only m–1 block Arnoldi steps and reduce a block upper Hessenberg matrix of order $(m–1)n_b$ to real Schur form. Since the columns of $\mathbf{X}_1$ are not recomputed, they do not need rechecking for convergence.

2.5 Polynomial preconditioned Arnoldi

The idea behind polynomial preconditioned Arnoldi methods is to replace $\mathbf{A}$ by a matrix of the form $\mathbf{C} = p(\mathbf{A})$, where $p(\lambda)$ is a polynomial, and to perform an Arnoldi method using $\mathbf{C}$ in place of $\mathbf{A}$ (see Saad 1989). The polynomial $p(\lambda)$ is chosen so that a sought-after eigenvalue λ_1 is transformed by p into an eigenvalue of $\mathbf{C}$ that is very large compared with the remaining eigenvalues. Arnoldi's method using the matrix $\mathbf{C}$ can be expected to converge rapidly to this eigenvalue. The eigenvalues of $\mathbf{C}$ are related to those of $\mathbf{A}$ by $\lambda_i(\mathbf{C}) = p(\lambda_i(\mathbf{A}))$ and the approximate eigenvalues of $\mathbf{A}$ can be obtained from the computed eigenvalues of $\mathbf{C}$ by solving a polynomial equation. However, if $p(\lambda)$ is a polynomial of degree l, this process is complicated by the fact that, for each eigenvalue $\lambda_i(\mathbf{C})$ of $\mathbf{C}$, the polynomial equation has l roots that are candidates for approximating one eigenvalue $\lambda_i(\mathbf{A})$ of $\mathbf{A}$. This difficulty may be avoided by using a Galerkin process whereby, at the end of the Arnoldi steps, the $m \times m$ matrix $\mathbf{B} = \mathbf{X}^{\mathrm{T}}\mathbf{A}\mathbf{X}$ is explicitly computed. The eigenvalues of $\mathbf{B}$ then approximate m eigenvalues of $\mathbf{A}$ and the eigenvectors of the real Schur form for $\mathbf{B}$ may be used to obtain the corresponding approximate eigenvectors of $\mathbf{A}$. This is analogous to the procedure used in the subspace iteration algorithm (see Stewart 1976 and Duff and Scott 1993).

The scaled and shifted Chebychev polynomial given by equation (2.3) is a suitable choice for the preconditioning polynomial. Again, the matrix $\mathbf{C} = p_l(\mathbf{A})$ is not formed explicitly but the three-term recurrence relation for Chebychev polynomials is used to form matrix-vector products with the matrix $\mathbf{C}$.

The Chebychev preconditioned Arnoldi method for finding the r right-most eigenvalues of $\mathbf{A}$ may be summarized as follows:

Algorithm A3: Chebychev preconditioned Arnoldi

1. *Initialization:* Choose the number of steps m and an initial vector $\mathbf{x}_1$ with unit norm. Set $l=1$, $p_l(\lambda)=\lambda$.
2. *Arnoldi steps :* See algorithm A1 with $\mathbf{A}$ replaced by $p_l(\mathbf{A})$ in (i).
3. *Eigenvalue computation:* Compute $\mathbf{B}=\mathbf{X}^{\mathrm{T}}\mathbf{AX}$. Reduce $\mathbf{B}$ to real Schur form $\mathbf{T}=\mathbf{Z}^{\mathrm{T}}\mathbf{BZ}$, where each diagonal block $\mathbf{T}_{ii}$ is either of order 1 or is a 2×2 matrix having complex conjugate eigenvalues, with the eigenvalues ordered in descending order of their real parts along the diagonal blocks. Set $\mathbf{X}\Leftarrow\mathbf{XZ}$.
4. *Convergence test and restart:* If the first r columns of $\mathbf{X}$ satisfy the convergence criteria then accept $\lambda_1, \lambda_2, \ldots, \lambda_r$ and **stop** else choose the degree l of the iteration polynomial $p_l(\lambda)$. Construct the ellipse $E(d,c,a)$ containing the unwanted eigenvalues. Define $p_l(\lambda)$. Let $\mathbf{x}_1$ be the first column of $\mathbf{X}$ and **go to 2**.

At each iteration the polynomial preconditioned algorithm is significantly more expensive than algorithm A2 since it requires $(m-1)$ matrix-vector products of the form $p_l(\mathbf{A})\mathbf{x}$, compared with only one such product for algorithm A2. In addition, algorithm A3 requires the explicit computation of the matrix $\mathbf{B}=\mathbf{X}^{\mathrm{T}}\mathbf{AX}$ and $\mathbf{B}$ has first to be reduced to upper Hessenberg form before it can be reduced to real Schur form. It is probably because of the extra cost per iteration that algorithm A3 has not been implemented in any of the experimental codes cited in Section 1. However, we have found that for more difficult problems and for problems for which more than one eigenvalue is required, the extra work per iteration may not lead to an increase in the overall cost since algorithm A3 frequently requires only a small number of iterations to achieve convergence. This is illustrated in Section 4 (see Tables 4.4a and 4.4b).

As in algorithms A1 and A2, some savings in the computational costs can be realised by employing locking when the number of required eigenvalues is greater than 1.

2.6 Block preconditioned Arnoldi

In the same way that algorithm A2 can be generalized to the block algorithm AB2, the preconditioned Arnoldi algorithm A3 can be generalized to a block method. This algorithm will be called algorithm AB3 and is as follows:

Algorithm AB3: Block Chebychev preconditioned Arnoldi

1. *Initialization:* Choose the number of steps m and an initial $m\times n_b$ matrix $\mathbf{X}_1$ with orthonormal columns. Set $l=1$, $p_l(\lambda)=1$.
2. *Block Arnoldi steps :* See algorithm AB1 with $\mathbf{A}$ replaced by $p_l(\mathbf{A})$ in (i).
3. *Eigenvalue computation:* See algorithm A3.
4. *Convergence test and restart:* If the first r columns of $\mathbf{X}$ satisfy the convergence criteria then accept $\lambda_1, \lambda_2, \ldots, \lambda_r$ and **stop** else choose the degree l of the iteration polynomial $p_l(\lambda)$. Construct the ellipse $E(d,c,a)$ containing the unwanted eigenvalues. Define $p_l(\lambda)$. Let $\mathbf{X}_1$ be the matrix whose columns are the first n_b columns of $\mathbf{X}$ and **go to 2**.

Again, as with algorithm AB2, locking can be used to reduce computational costs if some of the columns of $\mathbf{X}$ converge before all the required r columns have converged.

3 The new code `EB13`

In this section we discuss the design of a new code, `EB13`, which implements all the variants of Arnoldi's method considered in Section 2. We look at the user interface, the use of control parameters, the user-supplied parameters, the stopping criteria, the ellipse construction, and updating the degree of the Chebychev polynomial. We also consider the exploitation of BLAS by `EB13`.

3.1 The user interface

Subroutines in the `EB13` package are named according to the naming convention of the Harwell Subroutine Library. The single-precision version subroutines all have names that commence with `EB13` and have one more letter. The corresponding double-precision versions have the same names with an additional letter `D`. For clarity, in the remainder of this paper we will refer only to the single-precision subroutines. There are three subroutines in the `EB13` package that are called directly by the user. These are as follows:

(a) `EB13I:` Initialization of control parameters. This subroutine is normally called once prior to calls to `EB13A` and `EB13B`.

(b) `EB13A:` Computation of the right-most eigenvalues, the eigenvalues of largest absolute value, or the eigenvalues of largest imaginary parts..

(c) `EB13B:` Computation of the eigenvectors and scaled eigenvector residuals corresponding to the computed eigenvalues. Use of this subroutine is optional.

One of the main features of `EB13` is the use of reverse communication, so that the user is not required to supply the matrix $\mathbf{A}$ explicitly but whenever the code requires matrix-vector products, control is returned to the user. This allows full advantage to be taken of the sparsity and structure of $\mathbf{A}$, and of vectorisation and parallelism. It also gives the user greater freedom in choosing how to store the matrix $\mathbf{A}$ and allows for cases where the matrix is not explicitly available but only the action of $\mathbf{A}$ on vectors is known. Furthermore, it enables the more sophisticated user to employ explicit Wielandt deflation techniques (see, for example, Saad 1989). In addition, reverse communication allows flexibility in using the code to implement shift-and-invert strategies (again, see Saad 1989). Shift-and-invert replaces the matrix $\mathbf{A}$ by the shifted and inverted matrix $\mathbf{C} = (\mathbf{A} - \sigma\mathbf{I})^{-1}$, so that the original problem $(\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = 0$ is transformed into $(\mathbf{A} - \sigma\mathbf{I})\mathbf{x} = \lambda^{-1}\mathbf{x}$. The shift σ is chosen so that the matrix $\mathbf{C}$ has a spectrum with much better separation properties than the original matrix $\mathbf{A}$ and should therefore require fewer iterations for convergence. The price which must be paid for this is that, since for large problems it is impractical to compute $\mathbf{C}^{-1}$, when control is returned to the user, the matrix-vector product $\mathbf{w} = \mathbf{C}^{-1}\mathbf{u}$ should be computed by solving the linear system of equations

$$\mathbf{C}\mathbf{w} = \mathbf{u} \tag{3.1}$$

for $\mathbf{w}$. This can be done by forming the LU factorization of $\mathbf{C}$ and then solving (3.1) reduces to performing a forward and back-substitution. There are library codes available for doing this, including the recent Harwell Subroutine Library code `MA48` (see Duff and Reid 1993), which requires that $\mathbf{C}$ is available in standard sparse matrix format. Whenever a new shift σ is chosen, a new LU factorization of **C** must be formed.

A more general strategy than shift-and-invert which may also be implemented using EB13 is preconditioning. Here a matrix $\phi(\mathbf{A})$ is used in place of **A**. Use of the preconditioner $\phi(\lambda) = p_l(\lambda)$, the Chebychev polynomial given by (2.3), is offered by EB13 as an option and no action is required by the user other than specifying this option using a control parameter (see Section 3.2). However, other preconditioners may be used if, on each return to the user before convergence is achieved, $\mathbf{w} = \phi(\mathbf{A})\mathbf{u}$ is computed. A preconditioner which has received attention recently is the generalized Cayley transformation used by Garratt (1991).

Use of reverse communication also allows EB13 to be employed for the solution of generalized eigenvalue problems of the form

$$\mathbf{Ax} = \lambda \mathbf{Bx}, \tag{3.2}$$

where **A** and **B** are real, sparse, unsymmetric matrices. If **B** is nonsingular, (3.2) can be treated as a standard unsymmetric eigenvalue problem by working with the matrix $\mathbf{B}^{-1}\mathbf{A}$. Again, when control is returned to the user, a linear system of equations must be solved. For this problem, the LU factorization of **B** need only be formed once at the start of the computation. If **B** is singular, the problem of solving (3.2) becomes harder. A shift γ may be introduced so that $(\mathbf{A} - \gamma\mathbf{B})$ is nonsingular and then

$$(\mathbf{A} - \gamma\mathbf{Bx}) = (\lambda - \gamma)\,\mathbf{Bx} \tag{3.3}$$

may be treated as a standard eigenvalue problem by working with the matrix

$$(\mathbf{A} - \gamma\mathbf{B})^{-1}\mathbf{B}. \tag{3.4}$$

Parlett and Saad (1985) have considered ways of dealing with the case where **A** and **B** are real but the shift γ is complex. One possibility is to replace the matrix (3.4) by the real matrix

$$Re[(\mathbf{A} - \gamma\mathbf{B})^{-1}\mathbf{B}], \tag{3.5}$$

which has the same eigenvectors as the original problem and eigenvalues v_i which are related to the eigenvalues λ_i of (3.2) by

$$v_i = \frac{1}{2}\left[\frac{1}{\lambda_i + \bar{\gamma}_i} + \frac{1}{\lambda_i - \gamma_i}\right]. \tag{3.6}$$

If (3.5) is used in place of (3.4), EB13, which uses only real arithmetic, may be employed. An LU factorization of $(\mathbf{A} - \gamma\mathbf{B})$ must be formed (using, for example, the Harwell Subroutine Library code ME48 if γ is complex) and each time control is returned for the matrix-vector product $\mathbf{w} = Re[(\mathbf{A} - \gamma\mathbf{B})^{-1}\mathbf{B}]\mathbf{u}$, forward and back-substitutions must be done in the usual way before the real part of the resulting vector taken to yield **w**.

3.2 Control and information parameters

Another important design feature of EB13 is that no common blocks are used. We feel this is an aid to portability and it is consistent with the policy employed by other major recent additions to the Harwell Subroutine Library, including MA42, MA47, and MA48 (see Anon 1993). In EB13 the following arrays take the place of common blocks.

(i) An integer array ICNTL and a real array CNTL which control the action of EB13A. The control variables include the maximum number of matrix-vector products allowed, the maximum number of Arnoldi iterations, the level of printing, the stream numbers for error and warning messages, and the convergence tolerance (see Section 3.4). Using the control parameter ICNTL(10) the user can specify whether an initial estimate for the starting vector (or matrix for a block method) is to be supplied. This can be useful if, for example, $\mathbf{A}$ is a parameter-dependent matrix $\mathbf{A} = \mathbf{A}(\alpha)$ and the right-most eigenvalues are required as the parameter α varies. A converged basis vector for a given value of α can be used as the starting vector for $\alpha + \delta\alpha$. This problem arises in the detection of Hopf bifurcation points (see, for instance, Garratt 1991). If no estimate is available (the default), the starting vector (or matrix) for the Arnoldi process is generated using the Harwell Subroutine Library random number generator FA04. A control variable is also used to determine which of the variants of Arnoldi's method discussed in Section 2 is implemented. EB13 allows the user to choose which variant is employed since there is no loss in efficiency in offering each the variants A1/AB1, A2/AB2, A3/AB3 (much of the code is common to all the methods) and practical experience has shown that the method which gives the best performance for one problem may not necessarily give the best performance for another (see Section 4). The control variables are given default values by the initialization routine EB13I, which should normally be called once by the user at the start of the computation. Should the user want the control variables to have values other than the defaults, the appropriate variables should be reset after the call to EB13I.

(ii) Two arrays INFO and RINFO, which on each return to the user provide integer and real information regarding the execution of the code. This information includes the current ellipse parameters, the degree of the current Chebychev polynomial, and the number of eigenvalues which have converged.

(iii) Two arrays IKEEP and KEEP, which are used to hold variables which must be preserved between calls to EB13A.

Further details of ICNTL, CNTL, INFO, RINFO, IKEEP, and KEEP are given in the specification sheets (see Scott 1993).

3.3 Input parameters

The user is required to set the following parameters prior to the first call to EB13A.

(a) N: the order of the matrix $\mathbf{A}$.

(b) NUMEIG: the number of required eigenvalues.

(c) NBLOCK: the size of the block to be used (if NBLOCK $= 1$ an unblocked method is used).

(d) NSTEPS: the number of Arnoldi steps on each iteration.

(e) LN: the first dimension of the array which on successful exit will hold the converged basis vectors. LN $\geq$ N is required.

(f) IND: indicates whether the user wants to compute the eigenvalues of largest absolute value, the right-most eigenvalues, or the eigenvalues of largest imaginary parts.

Of these parameters, the only ones the user may need advice on choosing are NBLOCK and NSTEPS. Deciding whether or not to use a block method depends, in part, upon the separation of the eigenvalues. If the user requires λ_1 and λ_2 where $Re(\lambda_1) >> Re(\lambda_2)$, convergence will generally be achieved more quickly and in a smaller number of iterations using an unblocked method. This is illustrated by tests performed with the matrix IMPCOLA in Section 4 (Tables 4.4a and 4.4b). Conversely, if the required eigenvalues are multiple eigenvalues or are clustered, there can be significant advantages in using a block method. Tests using the matrix GRE1107 illustrate this in Section 4.

Experience has shown us that the last vectors in a block can be slow to converge but that convergence of the requested NUMEIG eigenvalues can often be achieved more rapidly by choosing the block size NBLOCK to be greater than NUMEIG. Again, this is demonstrated in Section 4 (see Tables 4.5a and 4.5b). However, EB13 does allow the user to choose a block method with NBLOCK smaller than NUMEIG. This option can be useful if the wanted eigenvalues are clustered in groups which are themselves well separated. Of course, the user may not have prior knowledge of the distribution of the eigenvalues. One of our primary aims when designing EB13 was that the code, while being flexible, should be robust in the event of the user making a poor choice for the input parameters. We have designed the stopping criteria (which are discussed in Section 3.4 below) so that the computation will terminate if convergence to the requested accuracy is not being achieved with acceptable speed. In this event, the user is advised to assign a different value to NBLOCK (and, optionally, to NSTEPS) and start the computation again. The (inaccurate) approximations to the eigenvalues already computed should allow the user to choose values which will lead to successful convergence on a subsequent run.

The parameter NSTEPS is the other important parameter which must be set by the user. The value of NBLOCK*NSTEPS influences the effectiveness of EB13 since the storage required is proportional to $(\texttt{NBLOCK}*\texttt{NSTEPS})^2$ and, at each iteration, the cost of the Arnoldi steps is proportional to $(\texttt{NBLOCK}*\texttt{NSTEPS})^2*\texttt{N}$, and computing the eigenvalues of the Hessenberg matrix **H** is proportional to $(\texttt{NBLOCK}*\texttt{NSTEPS})^3$. This is discussed by Sadkane (1991a). It is clear that if NSTEPS is chosen to be large, the computational costs per iteration and the storage requirements become prohibitively large, but if NSTEPS is too small, the Krylov subspace will not contain enough information and convergence may not be achieved even with a large number of iterations. In practice, if NSTEPS is chosen too small, the stopping criteria used by EB13 will terminate the computation with a warning to the user that the requested accuracy was not achieved. Should this happen, we do allow the user to restart the computation with a revised value for NSTEPS. This restart exploits the approximations to the basis vectors already computed. Full details of restarting the computation are given in the specification sheets for EB13 (see Scott 1993). Our numerical experience indicates that, in general, NSTEPS should be chosen to be at least 6 and for more difficult problems (problems for which there is a poor separation of the required eigenvalues or problems where the unwanted eigenvalues are not well represented by a Chebychev ellipse) a larger value is needed. Block methods generally converge using a smaller value of NSTEPS than unblocked methods (see Section 4).

3.4 The stopping criteria

At the end of each iteration it is necessary to test the computed basis vectors for convergence (Step 4 of the algorithm descriptions in Section 2). In many papers on iterative methods for the computation of eigenvalues, the criteria used for accepting a computed eigenpair $(\lambda, \mathbf{y})$ is based on demanding that the norm of the residual $\mathbf{Ay} - \lambda\mathbf{y}$ should be less than a prescribed tolerance, that is,

$$\|\mathbf{Ay} - \lambda\mathbf{y}\|_2 \leq \varepsilon \tag{3.7}$$

(see, for example, Saad 1980). Recently, Chatelin and Frayssé (1993) have recommended that stopping criteria should be based on the backward error. The idea of backward error is to measure the shortest distance between the original problem with computed solution $\mathbf{y}$ and a perturbed problem with exact solution $\mathbf{y}$. The normwise backward error associated with the approximate eigenpair $(\lambda, \mathbf{y})$ is defined to be

$$\eta = min\{\delta > 0 : \|\Delta\mathbf{A}\|_2 \leq \delta\|\mathbf{A}\|_2, (\mathbf{A} + \Delta\mathbf{A})\mathbf{y} = \lambda\mathbf{y}\}.$$

It can be shown (Deif 1989) that, if $\|\mathbf{y}\|_2 = 1$,

$$\eta = \frac{\|\mathbf{Ay} - \lambda\mathbf{y}\|_2}{\|\mathbf{A}\|_2}. \tag{3.8}$$

Chatelin and Frayssé propose as the stopping criteria $\eta \leq \varepsilon$. Note that, since $\|\mathbf{A}\|_2 \leq \|\mathbf{A}\|_\mathrm{F}$, where $\|\mathbf{A}\|_\mathrm{F} = (\sum_i\sum_j a_{ij}^2)^{\frac{1}{2}}$ denotes the Fobenius norm of $\mathbf{A}$, a lower bound on the backward error can be obtained by replacing $\|\mathbf{A}\|_2$ in (3.8) by $\|\mathbf{A}\|_\mathrm{F}$.

The smallest backward error is of order the machine precision u. But if there are uncertainties in the values of the entries a_{ij} of the matrix $\mathbf{A}$ (for example, $\mathbf{A}$ may be obtained from experimental readings), there is no advantage in reducing η below the uncertainty in the a_{ij}'s. Using the backward error, an estimate of the error in the computed solution may be obtained from the first order approximation

$$\textbf{Forward Error} \leq \textbf{Condition Number} \times \textbf{Backward Error}.$$

One of the features of Arnoldi's method (algorithm A1) is that, if $\mathbf{z}$ is the eigenvector of the $m \times m$ Hessenberg matrix $\mathbf{H} = \{h_{ij}\}$ associated with λ, then in exact arithmetic

$$\|\mathbf{Ay} - \lambda\mathbf{y}\|_2 = h_{m+1,m}|\mathbf{e}_m^\mathrm{T}\mathbf{z}|, \tag{3.9}$$

where $\mathbf{e}_m = (0, 0, ..., 1)^\mathrm{T}$ (see Saad 1980). The residual $\mathbf{r}_d = \|\mathbf{Ay} - \lambda\mathbf{y}\|_2$ is termed the direct residual and $\mathbf{r}_\mathbf{A} = h_{m+1,m}|\mathbf{e}_m^\mathrm{T}\mathbf{z}|$ is termed the Arnoldi residual (see Braconnier 1993). We call

$$\eta_d = \frac{\mathbf{r}_d}{\|\mathbf{A}\|_\mathrm{F}} \tag{3.10a}$$

the direct backward error and

$$\eta_\mathbf{A} = \frac{\mathbf{r}_\mathbf{A}}{\|\mathbf{A}\|_\mathrm{F}} \tag{3.10b}$$

the Arnoldi backward error. The Arnoldi residual has been preferred in numerical computations to the direct residual since it is faster and easier to compute. However, there can be problems in using $\mathbf{r}_\mathbf{A}$ (or $\eta_\mathbf{A}$) if the matrix is highly non-normal. Recall that a matrix $\mathbf{A}$ is said to be normal if $\mathbf{A}^\mathrm{T}\mathbf{A} = \mathbf{A}\mathbf{A}^\mathrm{T}$. Chatelin and Frayssé (1993) and Godet-Thobie (1992) define the relative departure of $\mathbf{A}$ from normality to be the Henrici number

$$He = \frac{\|\mathbf{A}^{\mathrm{T}}\mathbf{A} - \mathbf{A}\mathbf{A}^{\mathrm{T}}\|_{\mathrm{F}}}{\|\mathbf{A}^2\|_{\mathrm{F}}}. \tag{3.11}$$

The Henrici number is homogeneous in **A** (that is, invariant under the scalar transformation $\mathbf{A} \rightarrow \alpha\mathbf{A}$). Assuming **A** to be diagonalizable, Smith (1967) derives the following bound for the condition number of the eigenbasis **Y** of **A**

$$cond_2(\mathbf{Y}) = \|\mathbf{Y}\|_2 \|\mathbf{Y}^{-1}\|_2 \geq (1 + 0.5He)^{\frac{1}{4}}.$$

Thus a large departure from normality leads to an ill-conditioned eigenbasis, which can be extremely difficult to compute. Numerical experiments have shown that, although the direct and Arnoldi backward errors (η_d and $\eta_{\mathbf{A}}$, respectively) are mathematically equivalent in exact arithmetic, the difference between the computed η_d and $\eta_{\mathbf{A}}$ in finite precision becomes more and more significant as the departure from normality increases: $\eta_{\mathbf{A}}$ can become very small (down to machine zero) while η_d remains constant. Consequently, for such matrices using $\eta_{\mathbf{A}}$ can lead to incorrect conclusions concerning the precision of the computed eigenpairs. Our aim was to develop a robust code which would be suitable for a large class of problems so we have decided against using the Arnoldi residual in the stopping criteria.

To base the stopping criteria on the direct backward error η_d requires the eigenvalues corresponding to the computed eigenvalues to be determined on each iteration. In EB13 we work with the Schur vectors since this is more stable (see Stewart 1976). Computing the eigenvectors of the Hessenberg matrix **H** and then the corresponding eigenvectors of **A** at each iteration requires additional computational effort which may be avoided by basing our stopping criteria on demanding that the relation

$$\mathbf{AX} = \mathbf{XT}$$

is almost satisfied. The normwise backward error associated with $(\mathbf{T}, \mathbf{X})$, where $\mathbf{T} = \mathbf{X}^{\mathrm{T}}\mathbf{AX}$ is the projection of **A** on the invariant subspace with orthonormal basis **X**, is defined by

$$\eta = min\{\delta > 0 : \|\Delta\mathbf{A}\|_2 \leq \delta\|\mathbf{A}\|_2, (\mathbf{A} + \Delta\mathbf{A})\mathbf{X} = \mathbf{XT}\}.$$

It can be shown that

$$\eta = \frac{\|\mathbf{AX} - \mathbf{XT}\|_2}{\|\mathbf{A}\|_2}. \tag{3.12}$$

Again, a lower bound may be obtained by replacing $\|\mathbf{A}\|_2$ with replaced by $\|\mathbf{A}\|_{\mathrm{F}}$ to give

$$\eta_{\mathrm{T}} = \frac{\|\mathbf{AX} - \mathbf{XT}\|_2}{\|\mathbf{A}\|_{\mathrm{F}}}. \tag{3.13}$$

Computing the bound on the backward error η_d or η_{T} requires $\|\mathbf{A}\|_{\mathrm{F}}$. Assuming the action of **A** on vectors is known, $\|\mathbf{A}\|_{\mathrm{F}}$ can be computed using at most *n* matrix-vector products of the form **Ax**. Although this calculation need only be done once before the start of the eigenvalue computation, if *n* is large the cost may be unacceptably high and, indeed may be as great as the cost of computing the sought-after eigenvalues. If the user has an estimate of the norm of **A** available this could be used. An estimate of $\|\mathbf{A}\|_1$ or $\|\mathbf{A}\|_\infty$ can be found using at most 5 matrix-vector products of the form **Ax** and 5 of the form $\mathbf{A}^{\mathrm{T}}\mathbf{x}$ by employing the Harwell Subroutine Library code MC41 (Anon 1993). If $|\mathbf{A}|\mathbf{x}$ can be computed, where $|\mathbf{A}|$ is the matrix with entries $|a_{ij}|$ then, choosing $\mathbf{x} = (1, 1, ..., 1)^{\mathrm{T}}$, $\|\mathbf{A}\|_\infty$ can be computed with just one matrix-vector product.

The difficulty of requiring the norm of **A** can be avoided by accepting the i th eigenvalue if the scaled residual R_i given by

$$R_i = \frac{\|(\mathbf{AX}-\mathbf{XT})_i\|_2}{\|(\mathbf{AX})_i\|_2}. \tag{3.14}$$

is less than a prescribed tolerance. R_i was used in the stopping criteria employed by the code EB12 (see Duff and Scott 1993 for details). Since $\|(\mathbf{AX})_i\|_2 \le \|\mathbf{A}\|_2$ (recall $\|\mathbf{X}_i\|_2 = 1$), the backward error η given by (3.12) satisfies $\eta \le R_i$.

Another reason for possibly not wishing to use the backward error η_T is that it can lead to misleading conclusions. Well-conditioned matrices with a small departure from normality can have large norms. For example, the matrix

$$\begin{pmatrix} 1 & a \\ 0 & 10^k \end{pmatrix}$$

has Henrici number

$$He = \frac{2a^2(a^2+(10^k-1)^2)}{1+10^{2k}+(1+10^k)^2a^2}.$$

If a is fixed, as k increases He tends to $\sqrt{2}a$ but $\|\mathbf{A}\|_F$ is unbounded. In such cases, η_T can be very small while the computed eigenvalues are totally inaccurate. In some practical situations, eigenvalues are used to study stability and the interest is in whether the right-most eigenvalue has a nonpositive real part. Examples of this are our test problems GRT200 and GRT400 in Section 4. Since high precision in the computed eigenvalues is not required, the user may decide to set the convergence tolerance to be, for instance, 10^{-4}. But $\|\mathbf{A}\|_F$ is of order 10^5 for GRT400 (see Table 4.2) and we found that if $\eta_T \le \varepsilon$ is used as the stopping criteria, a computed λ with no precision is accepted and the wrong conclusion drawn as to the stability of the system. For this problem, **A** is a banded matrix and $\|(\mathbf{AX})_i\|_2$ is order 1 so that, if (3.14) is used, this difficulty is not encountered. Clearly, if the backward error is to be used as a stopping criteria, the user should take into account the size of $\|\mathbf{A}\|$ when selecting the convergence tolerance.

In EB13 we use as the default criteria for the acceptance of the i th eigenvalue

$$res_i = \frac{\|(\mathbf{AX}-\mathbf{XT})_i\|_2}{\|\mathbf{A}\|} \le \varepsilon. \tag{3.15}$$

However, because of the potential problems associated with the backward error, we offer the user the option of using $R_i \le \varepsilon$. With the use of reverse communication, it is straightforward to offer this without incuring any additional overheads. If the user wishes to take advantage of this option the control parameter ICNTL(7) must be set to 2 (the default is 0). If (3.15) is used the user is asked to provide $\|\mathbf{A}\|$ (or an estimate of $\|\mathbf{A}\|$) on the first call to EB13A. If an estimate is not available, the user must ICNTL(7) to 1 and EB13 will then compute $\|\mathbf{A}\|_F$ using n matrix-vector products.

The convergence tolerance used in EB13 is $\varepsilon=$ CNTL(1), where CNTL(1) is a control parameter. The default value for CNTL(1) is $u*10^3$, where u is the machine precision. This value is assigned to CNTL(1) by the call to the initialization subroutine EB13I. The code EB13 only accepts the basis vectors in the order $i = 1, 2,..., r$ so that res_j (or Res_j) is only computed and tested for convergence once res_i (or R_i), $i = 1, 2,..., j-1$ have all satisfied the convergence criteria.

We have already remarked that the backward error η is bounded above R_i and that the η cannot be less than u. Therefore, if R_i is used in the stopping criteria and the user sets `CNTL(1)` to be very small, the convergence criteria may not be met. Consequently, in `EB13` additional stopping criteria are needed to terminate the computation if this should happen. Additional stopping criteria are also need if the user has chosen the number of Arnoldi steps to be too small for the problem being solved since, in this case, the Arnoldi iterations can continue indefinitely without the residuals decreasing significantly. In our numerical experiments we observed that, as the number of iterations increases, if convergence has not been achieved, the residuals start to oscillate. As a result of our findings we have decided to terminate the computation on the k th iteration if

$res_i(k) < res_i(k{-}1)$, and
$res_i(k{-}1) > res_i(k{-}2)$, and
$res_i(k{-}2) < res_i(k{-}3)$, and
$res_i(k{-}3) > res_i(k{-}4)$,

where $res_i(k)$ is res_i (or R_i) on the k th iteration.

In the event of a problem failing to converge, we ensure a finite termination of the computation by imposing limits on the maximum number of matrix-vector products and the maximum number of Arnoldi iterations allowed. The limit on the number of matrix-vector products is determined by the control parameter `ICNTL(5)`, which has a default value of 20000. The maximum number of matrix-vector products allowed is `ICNTL(5)*NUMEIG` (`NUMEIG` is the number of wanted eigenvalues). If the number of matrix-vector products required exceeds this, the user is offered the option of increasing `ICNTL(5)` and continuing the computation from the point at which it was halted. The limit on the number of Arnoldi iterations is given by `ICNTL(11)`, which has default value 100. Again, if convergence is not reached in `ICNTL(11)` iterations, the user can increase `ICNTL(11)` and restart the computation at the next iteration. For details, see Scott (1993).

In `EB13`, once all the requested eigenvalues have converged, we offer the user the option of computing the eigenvectors $\mathbf{y}_i$, $i = 1, 2, ..., r$ and the scaled eigenvector residuals

$$Res_i = \frac{\|(\mathbf{A}\mathbf{y}_i - \lambda_i \mathbf{y}_i)\|_2}{\|\mathbf{A}\|}, \quad 1 \le i \le r, \tag{3.16a}$$

or, if (3.14) was used,

$$Res_i = \frac{\|(\mathbf{A}\mathbf{y}_i - \lambda_i \mathbf{y}_i)\|_2}{\|(\mathbf{A}\mathbf{y})_i\|_2}, \quad 1 \le i \le r. \tag{3.16b}$$

To obtain these residuals, the user must call `EB13B`. On the first call, `EB13B` computes the approximate eigenvectors of $\mathbf{A}$ by first computing the approximate eigenvectors $\mathbf{w}_i$ of the real Schur form $\mathbf{T}$ and then setting $\mathbf{y}_i = \mathbf{X}\mathbf{w}_i$. Control is returned to the user for the matrix-matrix product $\mathbf{AY}$ to be formed, where $\mathbf{Y}$ has columns $\mathbf{y}_1$, $\mathbf{y}_2$,..., $\mathbf{y}_r$. The user must then recall `EB13B` for the eigenvector residuals to be computed.

3.5 Ellipse construction

When using algorithm A2 or A3 (or their block generalizations), at each iteration it is necessary to construct an ellipse $E(d, c, a)$ enclosing the unwanted part of the eigenspectrum. Manteuffel's technique (Manteuffel 1975 and 1977) for constructing the optimal ellipse in the case of the solution of linear equations was extended by Saad (1984) to the unsymmetric eigenvalue problem. Since Manteuffel's algorithm requires the reference eigenvalue λ_r to be real (see (2.3)), when the reference eigenvalue is complex, Saad replaces λ_r by the point γ on the real line having the same convergence ratio as λ_r with respect to the ellipse found on the previous iteration. In general, $\gamma \neq \lambda_r$ and so the ellipse found by Saad is only an approximation to the optimal ellipse.

Ho (1990) introduced a new algorithm which avoids the shortcomings of Saad's method and computes the optimal ellipse with respect to λ_r. For full details of this algorithm the reader is referred to Ho (1990) and Ho, Chatelin, and Bennani (1990). Recently, Braconnier (1993) proposed an algorithm which is much simpler than that of either Saad or Ho, but does not necessarily determine the optimal ellipse. If the unwanted eigenvalues $\lambda_{r+1},...,\lambda_m$ are ordered in decreasing order of their real parts, Braconnier's algorithm proceeds as follows:

1. Set $d = \dfrac{Re(\lambda_{r+1}) + Re(\lambda_m)}{2}$, $a = Re(\lambda_{r+1}) - d$, and $\nu = 0.0$.
2. Set $b = \max\limits_i \dfrac{|\, aIm(\lambda_i)|}{\sqrt{a^2 - (Re(\lambda_i) - d)^2}}$ and $c^2 = a^2 - b^2$.
3. If $E(d, c, a)$ is feasible then **stop** else

   Set $\nu = \nu + 1$, $a = a + \dfrac{|Re(\lambda_{r+1}) - \nu Re(\lambda_r)|}{(\nu + 1.0)}$.

   go to 2.

At each iteration, Ho (1990) and Braconnier (1993) use only the current estimates of the unwanted eigenvalues to construct the ellipse. Our numerical experience has shown that, at iteration *k*, it is necessary to construct the positive complex hull containing the current estimates of the unwanted eigenvalues and, if the right-most eigenvalues are sought, points on the convex hull from iteration *k*–1 lying to the left of the current estimate of the right-most wanted eigenvalue (that is, λ_r). If the eigenvalues of largest imaginary parts are wanted, the points on the convex hull with imaginary parts which are less than the imaginary part of the current estimate of λ_r are used in constructing the convex hull. The ellipse is determined from the points on the convex hull. This is discussed further by Duff and Scott (1993). When implementing the algorithms of Saad, Ho, and Braconnier within the code `EB13` we have modified them to use the convex hull.

In his experimental code, Braconnier (1993) offers the user the option of employing either the ellipse construction algorithm he has developed or that of Ho (1990). However, Braconnier does not provide numerical results to illustrate the effects of not determining the optimal ellipse. We have performed some experiments on the test examples described in Section 4. In these tests algorithms A2 and A3 were combined with the different ellipse construction algorithms. The ellipse algorithms involve only scalar operations. Moreover, since the convex hull is used, the number of points on which the ellipse is constructed is small and, in each test case, the time for finding the ellipse was negligible compared with that for the matrix-vector products and for the reduction to Schur form and checking for convergence. There appeared to be little to choose between the performance of the ellipse algorithms and none of them displayed a consistent advantage over the others. Since Ho's algorithm does compute the optimal ellipse, in `EB13` we have decided to use Ho's algorithm as the default ellipse construction algorithm but offer the user the option of using either the algorithm of Braconnier or the algorithm of Saad. This choice is controlled by the parameter `ICNTL(8)` (see Scott 1993 for further details).

3.6 The degree of the Chebychev polynomial

It is well known that a severe limitation of combining Arnoldi's method with the use of Chebychev polynomials is the sensitivity of performance with the degree l of the polynomial. If l is chosen to be too small, convergence will be unnecessarily slow and there may be no convergence. However, if l is too large, more matrix-vector products than are really necessary will be performed and again convergence may not be achieved. We would like a code which is efficient on a range of problems but is also robust. As a result of our numerical experiments, in `EB13` the following criteria for determining the degree of the Chebychev polynomial are used. Here we let $l(k)$ denote the degree of the polynomial on the kth iteration.

(1) On the first few iterations, we anticipate that we may not have a good ellipse (that is, an ellipse which contains the unwanted spectrum of **A** rather than the computed unwanted eigenvalues). In this case it is sensible to update the ellipse quite quickly. In particular, we require $l(k+1) \le l_1$ where

$$l_1 = l(k) \times (1 + \log_{10}(k+1)) \ , \tag{3.4}$$

with $l(1) = 40$. This bound ensures that $l(k+1)$ varies from $l(k)$ in a controlled manner and that the degree of the iteration polynomial increases as the ellipses improve. The initial value $l(1) = 40$ was chosen as a result of numerical experiments. So that $l(k)$ does not grow without limit, we impose a maximum restriction on the degree $l(k) \le$ `ICNTL(3)` for all k, where `ICNTL(3)` is a control parameter (see Section 3.2). The default value for `ICNTL(3)` set by the initialization subroutine `EB13I` is 800.

(2) We impose the restriction on the degree of the Chebychev polynomial used by the code `EB12`, namely $l(k+1) \le l_2$ where

$$l_2 = 0.5 \times (1 + \log_{10}(u^{-1})/\log_{10}(ratio)). \tag{3.5}$$

Here *ratio* is the ratio of the convergence rates of the slowest and fastest converging eigenvalues (see Duff and Scott 1993).

(3) Near convergence we attempt to limit the number of unnecessary matrix-vector products by limiting the degree of the polynomial. If $res_r \le \texttt{CNTL(1)} * 10^2$ (see (3.1)), we set $l(k+1) \le l_3$ where

$$l_3 = t \times (1 + |\log_{10}(res_r/\texttt{CNTL(1)})|), \tag{3.6}$$

with $t = 40$. This condition is taken from Duff and Scott (1993). We found that it was necessary only to impose this condition for algorithms A3 and AB3. In general, for algorithms A2 and AB2 the number of unnecessary matrix-vector products which results from choosing too large a degree for the Chebychev polynomial is small and we found that imposing the restriction (3.6) on these algorithms could result in more iterations being required and an increase in the overall cost of achieving convergence.

If the user does not wish the code to use the above criteria for determining the degree of the polynomial, the degree may be chosen at each iteration using the control parameter `ICNTL(4)`. If `ICNTL(4)` has the default value 0, the above criteria (1)–(3) are used to select the polynomial degree, but if `ICNTL(4)` is greater than 0 then the degree of the Chebychev polynomial is taken to be `ICNTL(4)`. We have included this option since it allows the user to experiment with choosing different degrees for the iteration polynomials and it provides additional flexibility if it proves difficult to obtain convergence with the requested accuracy for a particular problem.

3.7 The use of high level BLAS

Within the code `EB13`, only dense linear algebra operations are performed. For efficiency we exploit Levels 2 and 3 BLAS when performing these operations (Dongarra, Du Croz, Hammarling, and Hanson 1988 and Dongarra, Du Croz, Duff, and Hammarling 1990). There are two main places in `EB13A` where we use BLAS. The first is in the Gram-Schmidt orthogonalisation process and the second is during the test for convergence. For the unblocked algorithms (A1, A2, and A3), the Gram-Schmidt process requires matrix-vector products of the form

$$\mathbf{h} \leftarrow \mathbf{X}_j^{\mathrm{T}}\mathbf{w} \tag{5.1}$$

and

$$\mathbf{w} \leftarrow \mathbf{w} - \mathbf{X}_j\mathbf{h}, \tag{5.2}$$

where $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_j] \in R^{n \times j}$, $\mathbf{w} \in R^{n \times 1}$, and $\mathbf{h} \in R^{n \times 1}$ ($j = 1, 2, ..., m$). In `EB13`, the Level 2 BLAS routine _GEMV is used to perform each of these matrix-vector products. To orthonormalise the vector $\mathbf{x}_{j+1}$ with respect to each of the vectors $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_j$ using the Gram-Schmidt process with one step of iterative refinement requires four calls to _GEMV. For the block algorithms (AB1, AB2, and AB3), if n_b is the blocksize, in (5.1) and (5.2) we have $\mathbf{X} = [\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_j] \in R^{n \times jn_b}$, $\mathbf{w} \in R^{n \times n_b}$, and $\mathbf{h} \in R^{n \times n_b}$. In this case the Level 3 routine _GEMM is used in place of _GEMV.

When testing the *i*th basis vector for convergence it is necessary to compute $\|(\mathbf{AX} - \mathbf{XT})_i\|_2$. In `EB13A` the Level 2 BLAS kernel _GEMV is used to compute $(\mathbf{AX} - \mathbf{XT})_i$ and then the Level 1 kernel _NRM2 is used to compute res_i (or R_i). Only once $\mathbf{x}_i$ (respectively, R_i) has been accepted is res_{i+1} (respectively, R_{i+1}) computed.

4 Numerical experiments

In this section we report the results of using `EB13` to compute selected eigenvalues of some matrices arising from practical problems. The performance of `EB13` is compared with that of the subspace iteration code `EB12`. `EB12` offers two algorithms: simple subspace iteration for computing the eigenvalues of largest absolute value (algorithm S1) and subspace iteration combined with Chebychev acceleration for computing the right-most (or left-most) eigenvalues (algorithm S2). For ease of reference, all the algorithms implemented in this study are summarised in Table 4.1.

Most of the problems we have used for performance testing were taken from the the Harwell-Boeing sparse matrix collection (Duff, Grimes, and Lewis 1992). This collection is now widely accepted and used in testing and evaluating sparse matrix algorithms. The matrices we selected from the collection are unsymmetric assembled matrices with the values of the nonzeros supplied, although some of them

Table 4.1. Summary of the algorithms used in this study.

Algorithm	Description
S1	Simple subspace iteration
S2	Subspace iteration with Chebychev acceleration
A1	Basic Arnoldi
AB1	Block Arnoldi
A2	Arnoldi with Chebychev acceleration
AB2	Block Arnoldi with Chebychev acceleration
A3	Chebychev preconditioned Arnoldi
AB3	Block Chebychev preconditioned Arnoldi

(namely PORES2 and PORES3) have a symmetric pattern. The problems and some of their characteristics are listed in Table 4.2. For more details of the origins of these problems the user is referred to Duff *et al.* (1992). The only test problems used which are not currently included in the Harwell-Boeing set are GRT200, GRT400, and TOLSA. GRT200 and GRT400 were supplied by Garratt (1990, 1991). These matrices arise from the detection of Hopf bifurcation points and the main concern for these problems is not the exact values of the eigenvalues but whether the right-most eigenvalue lies in the left or right-hand plane. As already noted, for the TOLOSA matrix, the eigenvalues of interest are those of largest imaginary parts. The TOLSA matrix is of order $n = 90 + 5k$ with 1746 + 9k nonzeros, where k is an integer. Bai (1993) illustrates the eigenvalue distribution of the TOLSA matrix for $n = 90$, 140, 240, 340, and 2000. We have chosen $n = 1000$ and 2000 for our tests.

We ran each of the Arnoldi based algorithms described in Section 2 on a subset of the test matrices (the entries marked with $^+$ in Table 4.2) then, based on our findings for this subset, we chose default values for our control parameters. We then performed further tests using the remaining test matrices and the algorithms A3, AB2, and AB3 only. The subset we used for the comprehensive testing comprised HOR131, GRE1107, PORES3, GRT200, and GRT400. The matrices HOR131 and GRE1107 were chosen since they provide examples of matrices for which the right-most eigenvalues are not favourably separated but are clustered. For the matrix PORES3, $\lambda_1 \cong -0.35$, $\lambda_n \cong -0.15*10^6$, and $|\lambda_i - \lambda_1|/|\lambda_n|$ is small for i small (here we are assuming the eigenvalues are ordered in descending order of their real parts). The matrices GRT200 and GRT400 have similar properties, that is, the left-most eigenvalues are large and negative while the right-most eigenvalues (which are those of practical importance) are close to the imaginary axis. The matrix IMPCOLA is rather different since for this matrix $\lambda_1 \cong 0.58*10^2$, $\lambda_2 \cong 0.12*10^1$, and $\lambda_n \cong -0.13*10^1$. Hence λ_1 is well separated and should be easy to compute using any of the algorithms we have discussed, but we anticipate that computing more than one eigenvalue will be harder.

The numerical experiments were performed on an IBM RISC System/6000 Model 550 using double precision arithmetic and on a single processor of a Cray Y-MP using single precision arithmetic. The convergence tolerance `CNTL(1)` was taken to be the default $u*10^3$, where u is the machine precision ($u = 2.2204*10^{-16}$ for the RS/6000 and $u = 7.1054*10^{-15}$ for the Y-MP). In the tables of results (which are presented together at the end of this section), an entry which is marked with an † indicates the requested accuracy was not achieved before the stopping criteria terminated the computation and an

Table 4.2. Characteristics of the matrices used for performance testing.

Matrix	Order	Number of entries	Henrici number	$\|\|\mathbf{A}\|\|_F$	Description/Discipline
GRE1107$^+$	1107	5664	7.7E-01	1.8E+01	Simulation of computer systems
GRT200$^+$	200	796	2.3E+00	6.8E+04	Hopf bifurcation problem
GRT400$^+$	400	1596	1.5E+00	3.9E+05	Hopf bifurcation problem
HOR131$^+$	434	4710	2.8E-02	2.1E+00	Flow network problem
IMPCOLA$^+$	207	572	5.3E+00	2.4E+03	Heat exchanger network
IMPCOLB	59	312	1.8E+00	1.4E+01	Cavett's process
IMPCOLC	137	411	1.5E+01	1.4E+02	Ethylene plant model
IMPCOLD	425	1339	1.7E+00	1.4E+02	Nitric acid plant model
IMPCOLE	225	1308	9.5E+01	1.6E+04	Hydrocarbon separation problem
NNC261	261	1500	1.1E+00	3.9E+03	Nuclear reactor core modelling
NNC666	666	4044	1.0E+00	6.5E+03	Nuclear reactor core modelling
NNC1374	1374	8606	1.0E+00	9.6E+03	Nuclear reactor core modelling
PORES2	1224	9613	4.3E+00	1.5E+08	Reservoir modelling
PORES3$^+$	532	3474	1.4E+00	6.6E+05	Reservoir modelling
TOLSA	1000	3384	3.0E+03	1.0E+07	Aerospace engineering
TOLSA	2000	5184	5.8E+03	5.4E+07	Aerospace engineering
WEST0156	156	371	1.6E+07	1.9E+07	Chemical engineering
WEST0167	167	507	2.0E+02	6.4E+05	Chemical engineering
WEST0381	381	2157	2.1E+02	3.0E+03	Chemical engineering
WEST2021	2021	7353	3.5E+01	1.8E+06	Chemical engineering

entry marked by ** indicates the default number of matrix-vector products or default number of iterations was insufficient. On each machine, the implementations of the BLAS provided by the manufacturer were employed.

In order to check the accuracy of the computed eigenvalues, all the eigenvalues of each of the test matrices were computed using the QR algorithm. To do this, the Harwell Subroutine Library code `EB07` was employed. It is interesting to note that on the Cray Y-MP it took 37.6 CPU seconds to compute all the eigenvalues of the TOLSA matrix of order 1000 and 606.7 seconds for the TOLSA matrix of order 2000. If only the eigenvalue of largest imaginary part is required then the CPU times required using algorithm A3 in `EB13` were 3.2 and 12.3 seconds, respectively (see Table 4.6a). In the tables of results, $FE_{\max}$ denotes the maximum relative forward error, that is,

$$FE_{\max} = \max_{1 \le i \le r} \frac{|\lambda_i - \lambda_i^*|}{|\lambda_i|},$$

where λ_i and λ_i^* are the exact and computed i th eigenvalue of $\mathbf{A}$, respectively.

Throughout this section, r denotes the number of eigenpairs sought and n_b denotes the blocksize. For `EB13` (algorithms A1, AB1, A2, AB2, A3, AB3), m is the number of Arnoldi steps used on each iteration (so that the order of the Hessenberg matrix is mn_b), and for `EB12` (algorithms S1, S2), m is the subspace dimension. In each test, the implicit deflation techniques outlined in Section 2 are used wherever appropriate. All CPU timings are in seconds. For `EB13`, $Res_{\max}$ denotes the maximum of the scaled eigenvector residuals given by (3.16a), and for `EB12`, $Res_{\max}$ is the maximum of the residuals given by (3.16b). We have this difference since the current version of `EB12` does not offer the option of using (3.16a), although on the basis of our current findings it may be necessary to include this option in future releases of `EB12`.

In Tables 4.3a and 4.3b we present a comparison of the subspace iteration and Arnoldi methods for computing the right-most eigenvalue of each of the test matrices marked with $^{+}$ in Table 4.2. Table 4.3a gives results for the Cray Y-MP and Table 4.3b for the IBM RS/6000. For the matrices GRT200, GRT400, and PORES3 the right-most eigenvalues are not those of largest absolute value, so the simple subspace iteration algorithm S1 and the basic Arnoldi algorithms A1 and AB1 are not appropriate. In these initial tests we are only interested in computing a single eigenvalue (or one complex conjugate pair of eigenvalues), but we did perform some runs using the block algorithms (with $n_b > r$). From the tables we observe that, as expected, each of the algorithms converged to the right-most eigenvalue of the matrix IMPCOLA in a single iteration. For the other examples, the block methods generally required more matrix-vector products than the unblocked methods, but a smaller number of iterations. On the Y-MP, this resulted in the block methods AB2 and AB3 requiring less total CPU time than the unblocked methods A2 and A3 but, on the RS/6000, the difference was less clear-cut and algorithm A2 generally performed well. However, for the matrix PORES3, A2 did not achieve the requested accuracy before the stopping criteria for slow convergence stopped the computation. Algorithm S2 failed to achieve the requested accuracy for this problem within the default number of matrix-vector products. We did try other values of m but for each of the values tried, algorithm S2 did not achieve the required accuracy within the default limit.

In Tables 4.4a and 4.4b we compare the performances of the subspace iteration and Arnoldi methods for computing the eight right-most eigenvalues of the test matrices HOR131, GRE1107, IMPCOLA, and PORES3. For the unblocked algorithms, for HOR131, GRE1107, and IMPCOLA we give results for $m=24$ and $m=40$; for PORES3 results for $m=40$ and $m=64$ are given since $m=24$ gave much poorer results. For the block algorithms, we took the block size to be $n_b = r+2$ (for IMPCOLA, $n_b = r+3$ since λ_r and λ_{r+1} are a complex conjugate pair of eigenvalues), and with up to 8 Arnoldi steps on each iteration. These choices were made for m since experience showed that, for the block methods, the best computational times are achieved by taking a much smaller number of Arnoldi steps on each iteration than for the unblocked methods. Recall that the storage requirements are proportional to mn_b and the Hessenberg matrix is of order mn_b, so with the values of m and n_b we used the storage requirements and work involved in computing the eigenvalues of the Hessenberg matrix were generally similar for the block and unblocked methods.

For the relatively straightforward test examples HOR131 and GRE1107, the subspace iteration algorithm S1 performs well (when the computational times for convergence are considered) compared with the Arnoldi algorithms. As expected, for these problems with clustered eigenvalues, the basic Arnoldi methods (A1 and AB1) require a large number of iterations and a large number of Arnoldi steps per iteration and, in general, these methods are seen not to be competitive with either the Arnoldi methods with Chebychev acceleration or the preconditioned Arnoldi methods. For the matrices IMPCOLA and PORES3, the subspace iteration algorithm performed poorly, failing to give the requested accuracy. For the matrix IMPCOLA (which does not have clustered eigenvalues), the block methods AB2 and AB3 performed less well than their unblocked counterparts A2 and A3. For PORES3, algorithm A2 failed to achieve the required accuracy. For most of the problems tried, algorithm A2 performed less well than algorithm A3 when more than one eigenvalue was required. We found that, for A2, the eigenvalues converged one at a time so that the number of iterations required increased with the number of eigenvalues sought. However, for A3, several eigenvalues converged at a single step so that the number of iterations required was often no more than for the convergence of a single eigenvalue.

We performed some experiments using different block sizes for algorithms AB2 and AB3. We summarize our findings in Tables 4.5a and 4.5b. In each of the reported tests the number of Arnoldi steps per iteration was taken to be 8. For problems IMPCOLA and GRE1107, with $n_b = r$, algorithm AB2 did not give the requested accuracy but algorithm AB3 appeared to be less sensitive to the choice of n_b. On the basis of our experience, we would recommend using a block size which is at least as large as the largest cluster of wanted eigenvalues.

We experimented further with algorithms A3, AB2 and AB3 using the test problems given in Table 4.2 which were not used in Tables 4.2 and 4.3. The results are in Tables 4.6a and 4.6b. For the TOLSA matrices the eigenvalue of largest imaginary part was computed and for the other matrices the right-most eigenvalues were computed. Results for the TOLSA matrices on the RS/6000 are not included since it was found to be prohibitively expensive to run several experiments with these matrices on the RS/6000 because of the time taken for the matrix-vector products. For example, using algorithm A3 on the RS/6000 to compute one eigenvalue of the TOLSA matrix of order 1000 required only 4 iterations but a CPU time of 521 seconds, of which 518 seconds was for the matrix-vector products. The results in Tables 4.6a and 4.6b broadly confirm our previous findings. In general we see that algorithm AB2 requires a greater number of iterations for convergence than algorithms A3 and AB3. However, since each iteration for AB3 involves $(m-1)ln_b$ matrix-vector products while for AB2 each iteration involves only ln_b products (l is the degree of the Chebychev polynomial), the total number of products for AB2 (and the time taken to compute them) can be significantly less than for AB3. Moreover, for a range of values of m and n_b, AB3 failed to give convergence with the requested accuracy for the TOLOSA matrices. The conflict between a small number of iterations and a small number of products makes it impossible (without some prior knowledge of the problem) to predict which of the algorithms A3, AB2, and AB3 is most likely to converge with the required accuracy in the shortest time. For this reason we allow the user of `EB13` to choose which method is employed. The default offered by `EB13` is AB3, which reduces to A3 if the user sets the blocksize to 1.

Table 4.3a. A comparison of subspace iteration and Arnoldi methods for computing the right-most eigenvalue (Cray Y-MP).

Problem	Algorithm	m	n_b	Iterations	Matrix-vector products	CPU Time for products	Total CPU Time	$Res_{\max}$	$FE_{\max}$
HOR131	S1	10		6	399	0.14	0.18	4.70E-10	3.98E-13
	S2	10		4	289	0.11	0.17	1.54E-08	4.51E-13
	A1	10		5	51	0.14	0.16	2.75E-13	3.01E-13
	AB1	8	2	6	98	0.13	0.20	9.34E-14	2.68E-13
	A2	10		2	61	0.16	0.18	7.24E-15	4.73E-13
	AB2	8	2	2	114	0.15	0.18	6.89E-15	4.73E-13
	A3	10		2	381	0.99	1.03	7.22E-14	2.36E-13
	AB3	8	2	2	594	0.78	0.85	1.17E-13	9.68E-14
GRE1107	S1	8		15	1983	0.93	1.07	4.08E-08	9.15E-10
	S2	8		19	3359	1.58	1.92	1.60E-08	5.53E-09
	A1	12		13	158	0.53	0.67	1.04E-04†	4.91E-04
	AB1	6	4	13	316	0.29	0.82	4.38E-04†	9.84E-04
	A2	12		6	597	1.93	2.06	4.89E-13	5.15E-13
	AB2	6	4	5	1280	1.13	1.52	7.32E-15	1.03E-13
	A3	12		2	465	1.54	1.64	5.68E-14	1.26E-12
	AB3	6	4	4	3440	2.99	3.67	4.64E-14	9.27E-13
IMPCOLA	S1	10		1	29	0.01	0.02	1.42E-13	5.02E-14
	S2	10		1	29	0.01	0.02	1.42E-13	5.02E-14
	A1	10		1	11	0.01	0.01	2.26E-12	1.25E-13
	A2	10		1	11	0.01	0.01	2.26E-12	1.25E-13
	A3	10		1	11	0.01	0.01	2.26E-12	1.25E-13
GRT200	S2	8		13	5079	1.55	1.66	1.51E-08	5.10E-09
	A2	8		8	1603	0.78	0.88	1.65E-10	4.71E-09
	AB2	6	4	6	2124	0.27	0.45	2.01E-10	4.65E-09
	A3	8		5	2069	1.01	1.13	4.90E-10	4.71E-09
	AB3	6	4	4	3460	0.44	0.64	2.11E-10	4.66E-09
GRT400	S2	10		19	14239	8.79	9.19	7.03E-08†	3.67E-08
	A2	10		11	4029	4.08	4.40	6.08E-14	9.16E-09
	AB2	6	4	7	3548	0.92	1.21	7.75E-14	9.15E-09
	A3	10		6	3345	3.36	3.64	9.60E-15	9.04E-09
	AB3	6	4	6	6028	1.58	2.02	4.84E-15	8.96E-09
PORES3	S2	12		**	**	**	**	**	
	A2	12		10	3250	6.64	6.66	3.25E-07†	1.08E-02
	AB2	6	4	10	12760	6.75	7.78	4.87E-15	1.04E-11
	A3	12		7	7279	14.24	15.13	8.71E-13	1.92E-13
	AB3	6	4	7	11472	6.01	7.03	3.24E-13	3.63E-10

Table 4.3b. A comparison of subspace iteration and Arnoldi methods for computing the right-most eigenvalue (IBM RS/6000).

Problem	Algorithm	m	n_b	Iterations	Matrix-vector products	CPU Time for products	Total CPU Time	$Res_{\max}$	$FE_{\max}$
HOR131	S1	10		6	439	0.26	0.42	2.73E-11	4.71E-13
	S2	10		4	429	0.25	0.40	1.87E-12	4.71E-13
	A1	10		6	61	0.15	0.25	9.30E-15	4.71E-13
	AB1	8	2	7	114	0.09	0.44	6.22E-16	4.72E-13
	A2	10		2	61	0.14	0.20	1.43E-16	4.71E-13
	AB2	8	2	2	114	0.11	0.26	2.22E-16	4.72E-13
	A3	10		2	381	0.76	0.98	9.35E-17	4.70E-13
	AB3	8	2	2	594	0.73	1.12	2.55E-16	4.72E-13
GRE1107	S1	8		10	2319	1.96	2.47	22.8E-10	5.19E-11
	S2	8		14	3375	2.45	4.29	1.42E-08	3.85E-09
	A1	12		13	158	0.48	1.40	1.83E-04†	8.61E-04
	AB1	6	4	13	316	0.28	3.04	4.30E-04†	9.34E-04
	A2	12		7	943	2.66	4.18	3.35E-17	7.69E-14
	AB2	6	4	5	1280	1.51	4.37	3.72E-16	7.50E-14
	A3	12		4	1743	4.88	7.67	1.31E-16	7.01E-14
	AB3	6	4	4	3460	4.23	10.33	5.01E-17	7.67E-14
IMPCOLA	S1	10		1	29	0.01	0.03	1.19E-13	1.57E-15
	S2	10		1	29	0.01	0.03	1.19E-13	1.57E-15
	A1	10		1	11	0.08	0.02	1.98E-14	0.00E+00
	A2	10		1	11	0.08	0.02	1.98E-14	0.00E+00
	A3	10		1	11	0.08	0.02	1.98E-14	0.00E+00
GRT200	S2	8		12	6175	1.21	1.79	1.23E-09	5.75E-09
	A2	8		8	1603	0.66	1.12	3.66E-16	4.71E-09
	AB2	4	4	7	3548	0.59	1.60	5.32E-17	4.71E-09
	A3	8		5	2070	0.86	1.55	2.11E-16	4.71E-09
	AB3	4	4	5	3552	0.51	1.61	1.90E-16	4.71E-09
GRT400	S2	8		12	7255	2.66	3.81	8.93E-09	9.23E-09
	A2	8		11	4029	3.26	5.40	5.94E-14	1.02E-08
	AB2	4	4	8	6248	1.83	5.29	2.82E-17	9.04E-09
	A3	8		6	3514	2.93	4.92	1.60E-16	9.03E-09
	AB3	4	4	6	6028	1.96	5.20	5.50E-16	9.03E-09
PORES3	S2	12		**	**	**	**	**	**
	A2	12		10	3250	5.47	7.93	3.25E-07†	1.08E-02
	AB2	6	4	9	9536	6.84	14.45	5.93E-08†	1.91E-03
	A3	12		7	9523	16.32	23.32	1.42E-17	1.41E-10
	AB3	6	4	7	13452	9.50	21.06	8.93E-15	1.07E-10

Table 4.4a. A comparison of subspace iteration and Arnoldi methods for computing the eight right-most eigenvalues (CRAY Y-MP).

Problem	Algorithm	*m*	n_b	Iterations	Matrix-vector products	CPU Time for products	Total CPU Time	$Res_{\max}$	$FE_{\max}$
HOR131	S1	40		5	505	0.39	0.56	3.17E-08	6.76E-12
		24		9	910	0.21	0.38	7.28E-08	1.28E-11
	S2	40		11	1693	0.31	0.82	5.00E-08	8.68E-12
		24		12	1919	0.41	0.65	5.30E-08	8.20E-12
	A1	40		5	202	0.50	0.66	6.64E-12	5.64E-13
		24		22	497	1.17	1.42	3.78E-12	4.92E-13
	AB1	4	10	16	650	0.20	0.99	4.06E-12	4.41E-13
		6	10	6	370	0.11	0.78	1.63E-12	4.41E-13
	A2	40		9	999	2.56	2.92	1.68E-12	8.17E-13
		24		11	439	1.08	1.23	2.35E-13	4.31E-13
	AB2	4	10	3	414	0.16	0.34	2.27E-12	4.73E-13
		6	10	3	444	0.17	0.53	4.48E-13	5.69E-13
	A3	40		2	720	1.80	1.95	3.03E-12	3.36E-13
		24		2	477	1.17	1.25	1.79E-12	3.33E-13
	AB3	4	10	3	1360	0.39	0.66	2.61E-12	4.80E-13
		6	10	2	1080	0.31	0.62	3.11E-12	4.41E-13
GRE1107	S1	40		32	8651	1.44	4.36	5.01E-08	1.14E-07
		24		27	6013	1.30	2.32	4.60E-08	1.61E-08
	S2	40		40	9712	1.68	5.97	6.12E-08	1.02E-07
		24		36	6540	1.43	3.10	5.45E-08	1.66E-07
	A1	40		46	1949	5.95	9.06	7.83E-06†	7.06E-03
		24		74	2001	5.55	7.50	6.70E-05†	7.46E-03
	AB1	4	10	**	**	**	**	**	**
		6	10	51	3070	1.21	12.84	1.38E-13	1.20E-12
	A2	40		42	8060	27.17	30.73	8.40E-09†	8.00E-08
		24		54	7832	26.16	28.20	1.51E-07†	1.09E-06
	AB2	4	10	5	2260	1.14	1.99	1.66E-12	2.19E-11
		6	10	4	1855	0.73	1.92	7.95E-13	2.28E-11
	A3	40		2	1656	5.45	5.92	8.81E-14	3.15E-12
		24		2	984	3.20	3.44	9.11E-14	2.94E-12
	AB3	4	10	4	4370	1.63	2.78	9.74E-14	3.43E-12
		6	10	2	2130	0.80	1.64	7.20E-14	2.64E-12
IMPCOLA	S2	40		19	1471	0.04	0.95	6.56E-08†	3.24E-06
		24		52	2553	0.07	1.07	1.22E-06†	1.41E-05
	A2	40		7	306	0.11	0.36	4.05E-13	9.31E-09
		24		21	555	0.19	0.43	2.68E-12	1.04E-08
	AB2	6	11	7	679	0.03	1.04	1.25E-13	2.57E-09
		8	11	4	468	0.02	1.18	8.06E-14	1.59E-09
	A3	40		2	284	0.08	0.20	8.69E-14	2.77E-10
		24		4	315	0.09	0.17	1.00E-12	2.15E-09
	AB3	6	11	4	935	0.03	0.69	3.47E-14	9.31E-10
		8	11	3	891	0.03	0.97	7.67E-15	9.17E-13
PORES3	S2	40		25	84503	7.71	10.99	7.84E-08	1.70E-09
	A2	40		48	17026	34.34	37.47	7.98E-07†	1.29E-02
		64		36	16895	34.20	39.49	4.67E-07†	0.12E+00
	AB2	4	10	10	25308	7.05	9.37	1.84E-12	2.97E-08
		6	10	10	23910	7.09	9.96	9.37E-13	2.37E-08
	A3	40		7	18508	37.30	39.37	1.51E-12	5.93E-08
		64		4	10713	21.47	23.11	2.10E-13	2.67E-09
	AB3	4	10	9	23260	5.27	7.58	1.11E-12	2.23E-09
		6	10	6	19570	4.44	6.62	2.64E-12	3.22E-09

Table 4.4b. A comparison of subspace iteration and Arnoldi methods for computing the eight right-most eigenvalues (IBM RS/6000).

Problem	Algorithm	m	n_b	Iterations	Matrix-vector products	CPU Time for products	Total CPU Time	$Res_{\max}$	$FE_{\max}$
HOR131	S1	40		5	611	0.42	1.30	9.48E-10	4.71E-12
		24		9	1007	0.54	1.27	1.24E-08	4.58E-13
	S2	40		8	2255	1.62	3.45	5.19E-09	3.20E-12
		24		12	2476	1.44	2.71	9.78E-10	4.72E-12
	A1	40		8	321	0.71	2.06	3.37E-14	4.67E-13
		24		28	635	1.25	2.87	4.09E-15	4.71E-13
	AB1	4	10	23	930	0.64	5.38	1.24E-14	4.71E-13
		6	10	11	670	0.42	5.98	1.33E-14	4.71E-13
	A2	40		5	280	0.56	1.37	3.75E-15	4.71E-13
		24		12	485	1.04	1.81	1.09E-15	4.71E-13
	AB2	4	10	3	456	0.30	1.14	9.77E-15	4.71E-13
		6	10	4	640	0.44	2.72	6.79E-16	4.71E-13
	A3	40		2	777	1.64	2.60	8.74E-15	4.72E-13
		24		3	844	1.10	1.53	5.52E-16	4.73E-13
	AB3	4	10	3	1480	1.22	2.81	6.68E-16	4.71E-13
		6	10	3	2290	2.01	5.22	1.43E-16	4.71E-13
GRE1107	S1	40		19	9044	9.88	18.81	3.96E-09	2.23E-09
		24		23	6651	4.87	9.24	8.28E-09	1.14E-09
	S2	40		24	11256	12.02	26.07	1.54E-08†	1.56E-08
		24		20	7691	5.60	11.65	2.82E-10	3.85E-10
	A1	40		**	**	**	**	**	**
		24		**	**	**	**	**	**
	AB1	4	10	**	**	**	**	**	**
		6	10	63	3790	3.64	76.19	3.04E-15	1.56E-12
	A2	40		53	11291	32.14	64.54	2.16E-11†	5.12E-09
		24		72	11832	34.42	58.20	1.08E-09†	1.09E-08
	AB2	4	10	19	12899	16.06	44.28	1.15E-10†	3.04E-11
		6	10	10	6202	7.54	27.30	5.71E-11†	1.01E-09
	A3	40		2	1656	5.07	8.51	1.59E-16	1.56E-12
		24		2	984	2.66	4.53	1.61E-15	1.56E-11
	AB3	4	10	3	4940	5.20	15.58	8.55E-16	1.57E-12
		6	10	2	2130	2.27	8.03	6.87E-16	1.56E-12
IMPCOLA	S2	40		18	1529	0.14	3.58	3.89E-08†	2.72E-06
		24		50	2195	0.22	3.30	1.22E-06†	8.63E-06
	A2	40		7	315	0.08	1.22	2.08E-16	8.23E-12
		24		22	616	0.13	1.06	3.85E-06†	2.42E-02
	AB2	6	11	7	723	0.03	4.39	4.23E-16	4.95E-12
		8	11	5	607	0.04	6.05	9.54E-16	1.51E-12
	A3	40		2	284	0.09	0.63	4.94E-15	1.60E-11
		24		5	482	0.23	0.80	3.81E-14	1.08E-10
	AB3	6	11	4	1100	0.21	3.25	4.65E-16	9.35E-12
		8	11	3	1045	0.21	4.51	2.81E-16	5.79E-12
PORES3	S2	40		**	**	**	**	**	
	A2	40		27	9965	17.55	27.61	4.00E-05†	1.86E+00
		64		33	17781	29.33	56.05	1.97E-06†	6.01E-02
	AB2	4	10	11	30951	15.41	40 30	4.00E-15	2.00E-09
		6	10	9	22774	11.84	29.77	2.10E-12	1.98E-09
	A3	40		5	11916	20.42	28.54	4.33E-13	1.98E-09
		64		4	11245	19.87	27.37	4.32E-13	1.99E-09
	AB3	4	10	8	26030	12.82	34.43	2.35E-14	1.99E-09
		6	10	6	25070	12.77	35.03	2.57E-15	1.99E-09

Table 4.5a. The effect of the blocksize n_b (CRAY Y-MP).

Problem	Algorithm	m	n_b	Iterations	Matrix-vector products	CPU Time for products	Total CPU Time	$Res_{\max}$	$FE_{\max}$
HOR131	AB2	4	8	4	416	0.20	0.36	3.75E-12	4.41E-13
		6	10	3	444	0.17	0.53	4.48E-13	5.69E-13
	AB3	4	8	3	1040	0.36	0.56	2.43E-13	4.62E-13
		6	10	2	1080	0.31	0.62	3.11E-12	4.41E-13
GRE1107	AB2	6	8	22	7276	13.13	17.54	7.91E-10	4.49E-09
		6	10	4	1855	0.73	1.92	7.95E-13	2.28E-11
	AB3	6	8	4	6928	3.17	4.95	1.20E-13	3.37E-12
		6	10	2	2130	0.80	1.64	7.20E-14	2.64E-12
IMPCOLA	AB2	8	8	15	1270	0.08	2.43	1.85E-06†	0.62E+00
		8	11	4	468	0.02	1.18	8.06E-14	1.59E-09
	AB3	8	8	3	664	0.03	0.55	2.45E-14	5.46E-10
		8	11	3	891	0.03	0.97	7.67E-15	9.17E-13
PORES3	AB2	6	8	11	21589	8.65	11.12	3.65E-12	3.14E-09
		6	10	10	23910	7.09	9.96	9.37E-13	2.37E-08
	AB3	6	8	7	20744	5.72	8.00	2.02E-12	2.20E-09
		6	10	6	19570	4.44	6.62	2.64E-12	3.22E-09

AB2 with block size 8 failed to give any accuracy..we could say failed to converge within default number of iterations? However, converged with required accuracy by increasing number of steps to 10.

Table 4.5b. The effect of the blocksize n_b (IBM RS/6000).

Problem	Algorithm	m	n_b	Iterations	Matrix-vector products	CPU Time for products	Total CPU Time	$Res_{\max}$	$FE_{\max}$
HOR131	AB2	4	8	6	572	0.48	1.49	9.72E-15	4.72E-13
		6	10	4	640	0.44	2.72	6.79E-16	4.71E-13
	AB3	4	8	3	1112	0.93	2.05	6.21E-16	4.72E-13
		6	10	3	2290	2.01	5.22	1.43E-16	4.71E-13
GRE1107	AB2	6	8	17	7230	11.23	33.39	7.64E-11†	9.87E-11
		6	10	10	6202	7.54	27.30	5.71E-11†	1.01E-09
	AB3	6	8	4	6888	6.90	21.19	1.27E-15	1.55E-12
		6	10	2	2130	2.27	8.03	6.87E-16	1.56E-12
IMPCOLA	AB2	8	8	9	996	0.18	5.52	1.36E-06†	6.21E-01
		8	11	5	607	0.04	6.05	9.54E-16	1.51E-12
	AB3	8	8	4	1128	0.20	3.12	8.87E-17	6.63E-13
		8	11	3	1045	0.21	4.51	2.81E-16	5.79E-12
PORES3	AB2	6	8	9	18273	9.04	26.08	3.10E-08†	2.29E-06
		6	10	9	22774	11.84	29.77	2.10E-12	1.98E-09
	AB3	6	8	7	24268	13.44	34.42	5.19E-14	1.99E-09
		6	10	6	25070	12.77	35.03	2.57E-15	1.99E-09

Table 4.6a. Further results for algorithms A3, AB2, and AB3 (Cray Y-MP).

Problem	Algorithm	r	m	n_b	Iterations	Matrix-vector products	CPU Time for products	Total CPU Time	$Res_{\max}$	$FE_{\max}$
IMPCOLB	A3	8	40		2	1656	0.31	0.45	5.60E-14	4.11E-13
	AB2	8	5	8	4	1437	0.04	0.22	2.34E-12	1.20E-11
	AB3	8	5	8	3	3080	0.08	0.26	3.57E-14	5.81E-13
IMPCOLC	A3	14	56		2	1811	0.47	0.71	8.58E-14	7.85E-13
	AB2	14	4	14	4	2529	0.06	0.48	4.92E-13	5.87E-13
	AB3	14	4	14	3	4074	0.09	0.48	2.60E-14	4.99E-13
IMPCOLD	A3	8	40		2	1656	1.38	1.62	1.17E-13	2.05E-12
	AB2	8	6	10	6	4290	0.51	1.65	7.10E-14	1.75E-12
	AB3	8	6	10	3	4790	0.20	0.66	3.62E-14	2.48E-12
IMPCOLE	A3	8	40		2	1655	1.26	1.44	1.02E-14	7.28E-12
	AB2	8	6	10	2	530	0.04	0.26	4.87E-15	1.46E-12
	AB3	8	6	10	2	2130	0.19	0.47	9.34E-15	1.48E-12
NNC261	A3	8	24		2	523	0.43	0.50	7.53E-14	6.48E-13
	AB2	8	4	10	2	490	0.04	0.16	2.00E-13	5.65E-13
	AB3	8	4	10	2	1290	0.13	0.28	3.31E-12	6.16E-13
NNC666	A3	8	24		2	615	1.37	1.50	1.20E-13	9.28E-13
	AB2	8	4	10	3	795	0.28	0.58	1.06E-12	1.97E-12
	AB3	8	4	10	3	2890	0.76	1.27	1.05E-12	1.25E-12
NNC1374	A3	8	24		2	684	3.28	3.49	1.68E-12	1.46E-12
	AB2	8	4	10	4	1316	1.00	1.68	1.30E-12	1.35E-12
	AB3	8	4	10	3	2890	1.66	2.57	1.01E-12	2.02E-12
PORES2	A3	4	20		2	808	4.47	4.71	5.09E-13	9.43E-09
	AB2	4	10	4	4	553	1.17	2.26	1.18E-14	3.81E-13
	AB3	4	10	6	3	6729	11.48	15.67	6.68E-13	6.28E-09
WEST0156	A3	8	48		2	1697	0.42	0.63	1.29E-12	1.64E-04
	AB2	8	6	10	10	1407	0.07	1.10	4.23E-12	1.72E-04
	AB3	8	6	10	7	8630	0.24	1.21	5.91E-13	1.77E-05
WEST0167	A3	8	60		1	68	0.02	0.09	9.44E-16	2.55E-08
	AB2	8	8	10	3	545	0.03	0.74	2.06E-13	2.52E-05
	AB3	8	8	10	2	1640	0.06	0.60	7.38E-14	5.10E-06
WEST0381	A3	8	24		2	985	1.22	1.33	1.57E-14	1.05E-11
	AB2	8	4	10	4	852	0.28	0.54	2.78E-12	5.87E-09
	AB3	8	4	10	2	1540	0.50	0.71	1.82E-14	2.52E-11
WEST2021	A3	4	28		2	1145	5.12	5.58	1.32E-16	1.42E-10
	AB2	4	11	5	4	939	1.05	2.46	1.13E-14	3.26E-05
	AB3	4	11	5	2	2115	2.01	3.26	1.19E-14	2.24E-05
TOLSA1000	A3	1	24		3	2192	3.03	3.49	5.87E-16	1.69E-09
	AB2	1	10	4	5	1360	1.94	2.81	2.50E-15	2.11E-10
	AB3	1	10	4	5	10608	15.20	17.87	4.67E-05†	0.56E+00
TOLSA2000	A3	1	24		3	2192	11.38	12.26	6.37E-15	9.34E-10
	AB2	1	10	4	6	2220	12.94	14.91	1.50E-15	3.57E-10
	AB3	1	10	4	5	10608	61.67	66.87	8.99E-05†	0.64E+00

Table 4.6b. Further results for algorithms A3, AB2, and AB3 (IBM RS/6000).

Problem	Algorithm	r	m	n_b	Iterations	Matrix-vector products	CPU Time for products	Total CPU Time	$Res_{\max}$	$FE_{\max}$
IMPCOLB	A3	8	40		2	1656	0.27	0.72	4.41E-16	4.81E-12
	AB2	8	5	8	5	2160	0.09	0.92	1.01E-14	4.81E-13
	AB3	8	5	8	3	3080	0.07	0.89	3.34E-16	4.79E-13
IMPCOLC	A3	14	56		2	1937	0.52	1.51	1.05E-15	1.80E-13
	AB2	14	4	14	5	2794	0.25	1.89	2.30E-15	1.76E-13
	AB3	14	4	14	3	4074	0.27	2.09	2.02E-14	1.80E-13
IMPCOLD	A3	8	40		3	3408	2.81	5.61	2.10E-14	1.61E-12
	AB2	8	6	10	6	4698	0.87	7.86	1.91E-14	1.61E-12
	AB3	8	6	10	3	4790	1.06	6.27	4.50E-15	1.61E-12
IMPCOLE	A3	8	40		2	1655	1.08	2.10	1.95E-16	1.51E-12
	AB2	8	6	10	2	530	0.05	1.04	1.95E-16	1.34E-12
	AB3	8	6	10	2	2130	0.40	2.13	2.03E-16	1.37E-12
NNC261	A3	8	24		2	592	0.43	0.78	3.50E-16	2.71E-13
	AB2	8	4	10	2	490	0.12	0.57	2.41E-14	2.73E-13
	AB3	8	4	10	2	1290	0.25	0.97	5.83E-14	2.80E-12
NNC666	A3	8	24		2	684	1.19	1.95	5.61E-16	1.96E-12
	AB2	8	4	10	3	795	0.43	1.90	1.81E-13	1.96E-12
	AB3	8	4	10	3	2890	1.67	5.20	3.61E-13	1.96E-12
NNC1374	A3	8	24		2	753	2.87	4.63	5.58E-14	1.34E-12
	AB2	8	4	10	4	1475	1.71	6.25	9.91E-14	1.34E-12
	AB3	8	4	10	4	5210	6.39	18.38	5.11E-14	1.34E-12
PORES2	A3	4	16		3	1657	7.37	11.11	3.93E-13	5.24E-10
	AB2	4	10	6	7	1596	3.09	13.71	1.16E-14	9.97E-12
	AB3	4	10	6	5	15210	25.58	53.19	1.33E-14	1.69E-12
WEST0156	A3	8	48		4	6941	1.56	4.01	7.26E-15	9.91E-07
	AB2	8	6	10	10	1559	0.11	4.29	1.65E-13	8.60E-06
	AB3	8	6	10	6	6270	0.13	1.87	4.68E-15	2.95E-07
WEST0167	A3	8	60		1	68	0.02	0.33	1.14E-17	4.64E-10
	AB2	8	8	10	4	728	0.06	3.24	3.40E-16	2.55E-08
	AB3	8	8	10	2	2620	0.21	2.40	11.1E-14	6.13E-07
WEST0381	A3	8	24		2	984	1.01	1.67	2.34E-16	1.37E-12
	AB2	8	4	10	10	5558	2.15	6.92	4.25E-14	7.78E-11
	AB3	8	4	10	2	1540	0.65	2.07	2.17E-16	1.26E-12
WEST2021	A3	4	28		2	1145	4.70	9.11	2.28E-18	1.38E-10
	AB2	4	11	5	5	1530	2.51	14.41	8.62E-18	2.48E-08
	AB3	4	11	5	3	4770	8.93	27.89	2.19E-18	9.80E-10

5 Conclusion

Although there is considerable interest in solving unsymmetric eigenvalue problems, there has been a lack of library software implementing Arnoldi methods. The purpose of this study was to efficiently implement variations of Arnoldi's method for computing the dominant eigenvalues of large unsymmetric matrices, to compare their performance on a range of test problems and, on the basis of the test results, to develop an Arnoldi based library code. The code we have designed and developed is `EB13`. This code will be included in the Harwell Subroutine Library. `EB13` has been tried and tested on a set of matrices arising from practical problems. The performance of `EB13` has also been compared with that of the Harwell Subroutine Library code `EB12`. For the more difficult test problems, `EB13` performed significantly better than `EB12`.

A key feature of `EB13` is that the user has the option of using a basic Arnoldi method, an Arnoldi method with Chebychev acceleration, or a Chebychev preconditioned Arnoldi method. A block version of each of these methods, with the blocksize chosen by the user, is also offered. It is not always possible to advise a user which method will give the best performance for a particular problem on a given machine hence this need for flexibility. When choosing which method to employ, the user should consider the following questions.

(1) Are several eigenvalues required?

(2) Are the sought-after eigenvalues known to be clustered?

(3) Can blocks of matrix-vector products be performed efficiently on the machine to be used?

(4) Does the machine to be used offer efficient implementations of the Level 3 BLAS kernel _GEMM?

If the answer to each of these questions is yes, a block algorithm AB2 or AB3 should be tried. However, if only one eigenvalue is wanted, algorithm A2 may give a better performance. Furthermore, for computing one or more eigenvalue, the number of matrix-vector products required by A3 will often be considerably less than for AB3. Even if the user makes a poor choice of method for the problem of interest, `EB13` has been designed to terminate the computation leaving the user with information which should be helpful in resetting the input parameters to achieve convergence on a subsequent run.

To limit as much as possible the number of input parameters which must be set by the user, at each iteration `EB13` automatically selects the degree of the Chebychev polynomial. The criteria used by `EB13` to determine the polynomial degree are designed to ensure the code is robust. For some problems this may mean that other choices of the degree will lead to more rapid convergence. However, through the use of control parameters, the user is able to overrule the choice of degree made by `EB13` at any stage of the computation. This increases the flexibility of the code and is a feature which the more experienced user may wish to exploit to enhance the rate of convergence.

6 Availability of the code `EB13`

`EB13` is written in standard FORTRAN 77. The code will be included in Release 12 of the Harwell Subroutine Library and anyone interested in using the code should contact the HSL Manager: Ms L Thick, Harwell Subroutine Library, AEA Technology, Building 8.19, Harwell, Oxfordshire, OX11 0RA, England, tel (44) 235 432688, fax (44) 235 432989, or e-mail libby.thick@aea.orgn.uk, who will provide details of price and conditions of use.

7 Acknowledgments

I would like to thank Miloud Sadkane and Thierry Braconnier for copies of their codes, and to thank Tony Garratt for supplying the test problems GRT200 and GRT400. I am also grateful to Iain Duff for his interest in this work and for commenting on this manuscript. I would like to thank Valerie Fraysśe for her comments on the problems of computing eigenvalues of highly nonnormal matrices and to thank an anonymous referee for some helpful observations and remarks.

8 References

Anon. (1993). Harwell Subroutine Library. A catalogue of subroutines (Release 11). Theoretical Studies Department, AEA Technology.

Arnoldi, W. E. (1959). The principle of minimized iteration in the solution of matrix eigenvalue problem. *Quat. Appl. Math.* **9**, 17-29.

Bai, Z. (1993). A collection of test matrices for the large scale nonsymmetric eigenvalue problem. Private communication.

Braconnier, T. (1993). The Arnoldi-Tchebycheff algorithm for solving large nonsymmetric eigenproblems. Tech. Report TR/PA/93/25, CERFACS, Toulouse.

Chatelin, F. and Fraysśe, V. (1993). Qualitative computing: elements of a theory for finite-precision computation. Lecture notes for the Comett European Course.

Daniel, J. W., Gragg, W. B., Kaufman, L., and Stewart, G. W. (1976). Reorthogonalization and stable algorithms for updating the Gram-Schmidt QR factorization. *Math. Comp.* **30**, 772-795.

Deif, A. (1989). A relative backward perturbation theorem for the eigenvalue problem. *Numerische Math.* **56**, 625-626.

Dongarra, J. J. and Grosse, E. (1987). Distribution of mathematical software via electronic mail. *Comm. ACM* **30**, 403-407.

Dongarra, J. J., Du Croz, J., Duff, I. S., and Hammarling, S. (1990). A set of Level 3 Basic Linear Algebra Subprograms. *ACM Trans. Math. Softw.* **16**, 1-17.

Dongarra, J. J., Du Croz, J., Hammarling, S., and Hanson, R. (1988). An extended set of Fortran Basic Linear Algebra Subprograms. *ACM Trans. Math. Softw.* **14**, 1-17.

Duff, I. S., Grimes, R. G., and Lewis, J. G. (1992). User's guide for the Harwell-Boeing sparse matrix collection (Release 1). Technical Report RAL 92-086, Rutherford Appleton Laboratory.

Duff, I. S. and Reid, J. K. (1993). MA48, a Fortran code for direct solution of sparse unsymmetric linear systems of equations. Technical Report RAL 93-072, Rutherford Appleton Laboratory.

Duff, I. S. and Scott, J. A. (1993). Computing selected eigenvalues of large sparse unsymmetric matrices using subspace iteration. *ACM Trans. Math. Softw.*, **19**,137-159.

Garratt, T. J. (1990). Private communication.

Garratt, T. J. (1991). The numerical detection of Hopf bifurcation in large systems arising in fluid mechanics. Ph.D. thesis, University of Bath.

Garratt, T. J., Moore, G., and Spence, A. (1991). Two methods for the numerical detection of Hopf bifurcations. In: Bifurcation and chaos: analysis, algorithms and applications (eds R. Seydel, F. W. Schneider, and H. Troger). Birkhauser, 119-123.

Godet-Thobie, S. (1992). Eigenvalues of large highly nonnormal matrices. PhD thesis, University Paris IX Dauphine.

Ho, D. (1990). Tchebychev acceleration technique for large scale nonsymmetric matrices. *Numerische Math.* **56**, 721-734.

Ho, D., Chatelin, F., and Bennani, M. (1990). Arnoldi-Tchebychev procedure for large scale nonsymmetric matrices. *Mathematical Modelling and Numerical Analysis* **24**, 53-65.

Kerner, W. (1989). Large-scale complex eigenvalue problems. *Comp. Phys.* 1-85.

Lehoucq, R B., Sorensen, D. C., and Vu, P. (1994). ARPACK: An implementation of the Implicitly Re-started Arnoldi Iteration that computes some of the eigenvalues and eigenvectors of a large sparse matrix. Available from netlib@ornl.gov under the directory scalapack.

Manteuffel, T. A. (1975). An iterative method for solving nonsymmetric linear systems with dynamic estimation of parameters. Ph.D. Thesis, Technical Report UIUCDCS-75-758 Department of Computer Science, University of Illinois at Urbana-Champaign, Illinois.

Manteuffel, T. A. (1977). The Tchebyshev iteration for nonsymmetric linear systems. *Numerische Math.* **28**, 307-327.

NAG (1993). NAG Fortran Library Mark 15, NAG Ltd, Oxford, England.

Parlett, B. N. and Saad, Y. (1985). Complex shift and invert strategies for real matrices. Technical Report YALEU/DCS-RR-424, Yale University Department of Computer Science.

Peters, G. and Wilkinson, J. H. (1970). Eigenvectors of real and complex matrices by LR and QR factorizations. *Numerische Math.* **16**, 181-204.

Saad, Y. (1980). Variations on Arnoldi's method for computing eigenelements of large unsymmetric matrices. *Linear Alg. and its Applics.* **34**, 269-295.

Saad, Y. (1984). Chebyshev acceleration techniques for solving nonsymmetric eigenvalue problems. *Math. Comp.* **42**, 567-588.

Saad, Y. (1989). Numerical solution of large nonsymmetric eigenvalue problems. *Computer Physics Communications* **53**, 71-90.

Sadkane, M. (1991a). A block Arnoldi-Chebychev method for computing the leading eigenpairs of large sparse unsymmetric matrices. Tech. Report TR/PA/91/46, CERFACS, Toulouse.

Sadkane, M. (1991b). On the solution of large sparse unsymmetric eigenvalue problems. Tech. Report TR/PA/91/47, CERFACS, Toulouse.

Scott, J. A. (1993). An Arnoldi code for computing selected eigenvalues of sparse real unsymmetric matrices. Rutherford Appleton Laboratory Report RAL-93-097.

Smith, R. A. (1967). The condition numbers of the matrix eigenvalue problem. *Numerische Math.* **10**, 232-240.

Sorensen, D. C. (1992). Implicit application of polynomial filters in a k-step Arnoldi method. *SIAM J. Matrix Anal. Appl.* **13**, 357-385.

Stewart, G. W. (1978). SRRIT – A FORTRAN subroutine to calculate the dominant invariant subspaces of a real matrix. Technical Report TR-514, Univ. of Maryland.