## 1 Introduction

We are interested in computing a few eigenvalues and the corresponding eigenvectors of a large sparse real unsymmetric matrix. 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 an 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), Sorensen (1992), and Braconnier (1993). These are 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 any of the standard numerical analysis software libraries, including NAG (NAG 1993), the Harwell Subroutine Library (Anon 1993), and LAPACK (Anderson *et. al.* 1992).

The recent 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 modulus 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 and the specification sheets (write-up) for `EB13` are given in the Appendix.

## 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 **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 **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 **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 **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 **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) has shown 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 **A** are ordered with $|\lambda_1| > |\lambda_2| \geq ... \geq |\lambda_n|$ and that $\lambda_1$ is required. The main steps in the basic Arnoldi algorithm for finding $\lambda_1$ 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 **T** is a block triangular matrix and 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 column $\mathbf{x}_1$ of **X** satisfies the convergence criteria then accept $\lambda_1$ 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 $\lambda_1$ has been found, the corresponding (approximate) eigenvector $\mathbf{y}_1$ of **A** can be determined. The eigenvector $\mathbf{w}_1$ of the matrix **T** corresponding to $\lambda_1$ is first found using a backsubstitution process (Peters and Wilkinson 1970) and then the eigenvector of **A** is taken to be $\mathbf{y}_1 = \mathbf{X}\mathbf{w}_1$ (see, for example, Duff and Scott 1993 for further details).

When more than one eigenvalue (or more than one complex conjugate pair of eigenvalues) is required, instead of restarting the method with $\mathbf{x}_1$ equal to the latest approximation to the basis vector associated with the sought-after eigenvalue, the restart vector may be taken to be a linear combination of basis vectors of the form $\sum_{i=1}^{r} \alpha_i \mathbf{x}_i$, where *r* is the number of eigenvalues sought. The main drawback with this approach is deciding how to choose the coefficients $\alpha_i$ systematically; for hard problems it can be difficult to achieve convergence. Braconnier (1993) chooses $\alpha_i$ to be the residual $\mathbf{A}\mathbf{y}_i - \lambda_i \mathbf{y}_i$. An alternative approach is to combine algorithm A1 with deflation. Deflation techniques have been discussed by Saad (1989) (see also Duff and Scott 1993). In this study we will only use implicit deflation techniques. For the basic Arnoldi method, implicit deflation proceeds as follows. Suppose 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 ($k-1 < r$) and $\mathbf{x}_k$ is orthogonal to each of the vectors $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{k-1}$. On the next iteration $m-k$ Arnoldi steps are performed in which the orthogonality of $\mathbf{x}_j$, $j = k, k+1, ..., m$ against $\mathbf{x}_i$, including $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{k-1}$ is enforced. Since the vectors $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{k-1}$ are "locked", so is the corresponding part of the Schur matrix **T** (that is, the leading $(k-1)\times(k-1)$ submatrix of **T**) and it is necessary only to reduce a Hessenberg matrix of order $m-k$ to real Schur form and compute its eigenvalues.

### 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 **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 2a and 2b 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 is often advantageous to choose a block size $n_b > r$. Some results which illustrate this are included in Section 4 (Tables 4.4a and 4.4b). 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. If $n_b < r$, some work can be saved by locking a block of vectors as soon as all the vectors in the block have converged.

### 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 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 $\lambda_1, \lambda_2, ..., \lambda_r$, whilst 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

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

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 $\lambda_r$ 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 $\max_{\lambda_j \, \varepsilon S} \mathbf{C}_j(d, c)$ 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 **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 **T** is a block triangular matrix and 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 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 **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 **X**. Set $\mathbf{x}_1 \Leftarrow \hat{\mathbf{x}} / \|\hat{\mathbf{x}}\|_2$ and **go to 2**.

As with algorithm A1, the difficulties associated with choosing an appropriate restart vector may be avoided by employing implicit deflation. In this case, once $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{k-1}$ have converged ($k-1 < r$), on the next iteration the restart vector is taken to be the first unconverged column $\mathbf{x}_k$ of **X**. Then $p_l(\mathbf{A})\mathbf{x}_k / \|p_l(\mathbf{A})\mathbf{x}_k\|_2$ is formed and orthogonalised against each of the converged vectors $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{k-1}$. An $(m-k) \times (m-k)$ Hessenberg matrix must be reduced to real Schur form.

By replacing **A** by **–A**, algorithm A2 may be used to compute the left-most eigenvalues of **A**.

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 **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 $2 \times 2$ 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. If the first $k$–1 vectors have converged ($k-1 < n_b$), then on the next iteration, the restart matrix may be taken to be $\mathbf{X}_1 = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{k-1}, p_l(\mathbf{A})\mathbf{x}_k, ..., p_l(\mathbf{A})\mathbf{x}_{n_b}]$. If $n_b < r$, once a block of vectors has converged, the whole block may be locked.

### 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 $\lambda_1$ 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 value $\lambda_i(\mathbf{C})$ 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 **C**.

The Chebychev preconditioned Arnoldi method for finding the *r* right-most eigenvalues of **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 **A** replaced by $p_l(\mathbf{A})$ in (i).

3. *Eigenvalue computation:* Compute $\mathbf{B} = \mathbf{X}^{\mathrm{T}}\mathbf{A}\mathbf{X}$. Reduce **B** to real Schur form $\mathbf{T} = \mathbf{Z}^{\mathrm{T}}\mathbf{B}\mathbf{Z}$ , 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 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 **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 $p_l(\lambda)$. Let $\mathbf{x}_1$ be the first column of **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 $\mathbf{B} = \mathbf{X}^{\mathrm{T}}\mathbf{A}\mathbf{X}$ and **B** has first to be reduced to upper Hessenberg form before it is 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 4.3a and 4.3b).

Some savings in the computational costs can be realised by employing implicit deflation 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 **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 **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 $p_l(\lambda)$. Let $\mathbf{X}_1$ be the $n_b$ columns of **X** and **go to 2**.

## 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. Finally in this section, we consider the exploitation of BLAS in 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.

(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 **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 **A**, and of vectorisation and parallelism. It also gives the user greater freedom in choosing how to store the matrix **A** and allows for cases where the matrix is not explicitly available but only the action of **A** on vectors is known. In addition, it allows flexibility in using the code to compute other parts of the eigenspectrum using a shift-and-invert technique or to work with $\phi(\mathbf{A})$ in place of **A**. $\phi(\lambda)$ could be a rational function. For example, $\phi(\lambda)$ could be the generalized Cayley transformation used by Garratt (1991). Furthermore, it allows the user employ explicit Wielandt deflation techniques (see, for example, Saad 1989).

### 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 level of printing, the stream numbers for error and warning messages, and the convergence tolerance. 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 Appendix).

### 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 **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 ≥ N is required.

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 $\lambda_1$ and $\lambda_2$ 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 4.3a and 4.3b).

Conversely, if the required eigenvalues are multiple eigenvalues or are clustered, there will be 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 4.4a and 4.4b ). 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 $(\mathtt{NBLOCK}*\mathtt{NSTEPS})^2$ and, at each iteration, the cost of the Arnoldi steps is proportional to $(\mathtt{NBLOCK}*\mathtt{NSTEPS})^2*\mathtt{N}$, and computing the eigenvalues of the Hesenberg matrix **H** is proportional to $(\mathtt{NBLOCK}*\mathtt{NSTEPS})^3$. 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 a large number of iterations can be needed to achieve convergence. In practice, if NSTEPS is chosen too small, the stopping criteria used by EB13 will terminate the computation. 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 Appendix).

### 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). Following the recent recommendations of Chatelin and Fraysée (1993), we have chosen the stopping criteria used by EB13 to be based on the backward error

$$res_i = \frac{\|(\mathbf{AX}-\mathbf{XT})_i\|_2}{\|\mathbf{A}\|_2}. \tag{3.1}$$

If $\|\mathbf{A}\|_2$ is known, it should be passed to EB13A on the initial call (for details, see Appendix). If $\|\mathbf{A}\|_2$ is not known but an estimate of the norm of **A** for any appropriate norm (such as the 1-norm, the infinity norm, or the Frobenius norm) is available, this should be passed to EB13A. This norm will replace $\|\mathbf{A}\|_2$ in (3.1). Supplying the code with the norm of **A** saves computational effort since, otherwise, the Frobenius norm of **A** ($\|\mathbf{A}\|_F = (\sum_i \sum_j a_{ij}^2)^{\frac{1}{2}}$) is computed by EB13A at the start of the computation. This involves *n* matrix-vector products of the form **Ax** (*n* is the order of **A**). Control is

returned to the user by EB13A $n$ times for the necessary products to be performed. EB13A accumulates the results and the computed norm $\|\mathbf{A}\|_F$ is stored and returned to the user in the information variable RINFO(5) (see Appendix). If the user wishes to perform more than one computation with the same matrix (for example, both the right-most and the left-most eigenvalues may be of interest), work can be saved on the second and subsequent runs by passing the norm of **A** computed on the first run.

We remark that the stopping criteria used by Duff and Scott (1993) in the code EB12 was not based on the residual (3.1) but on the residual $R_i$ given by

$$R_i = \frac{\|(\mathbf{AX} - \mathbf{XT})_i\|_2}{\|(\mathbf{AX})_i\|_2}. \tag{3.2}$$

However, Chatelin and Fraysée have found that it is advisable to use $\|\mathbf{A}\|$ in the stopping criteria to allow for matrices with large norms. This is the case, for example, when a matrix is highly non-normal (recall that $\nu = \|\mathbf{A}^T\mathbf{A} - \mathbf{A}\mathbf{A}^T\|$ quantifies the nonnormality of a matrix **A** and, since $\nu \le 2\|\mathbf{A}\|^2$, as $\nu$ increases so does $\|\mathbf{A}\|$).

For convergence of the $i$th basis vector we require $res_i \le$ CNTL(1), where the control parameter CNTL(1) is the convergence tolerance. The default value for CNTL(1) is $\sqrt{u}$, where $u$ is the machine precision. This is value assigned to CNTL(1) by the call to the initialization subroutine EB13I (see Appendix). The code EB13 only accepts the basis vectors in the order $i = 1, 2, ..., r$ so that $res_j$ is only computed and tested for convergence once $res_i$, $i = 1, 2, ..., j-1$ have all satisfied the convergence criteria.

EB13 monitors for lack of convergence and slow convergence. Let $\lambda_j(k)$ and $res_j(k)$ denote, respectively, the computed approximation to the $j$ th eigenvalue and the corresponding scaled residual on the $k$ th iteration. For all $k$ sufficiently large, the scaled residuals should satisfy

$$res_j(k+1) \le res_j(k).$$

However, if

$$res_j(k+1) > res_j(k) > \text{CNTL(1)}$$

and

$$|\lambda_j(k+1) - \lambda_j(k)| \le \text{CNTL(1)} * 10^2 * \max(|\lambda_j(k+1)|, |\lambda_j(k)|),$$

EB13 accepts $\lambda_j(k+1)$ and issues a warning that the convergence tolerance requested by the user has not been achieved. Convergence of the $j$ th eigenvalue is considered to be intolerably slow if, for some $k$, one of the following sets of conditions holds.

(i) $res_j(k+1) \le \text{CNTL(1)} * 10^2$, $res_j(k+1) \le \max_{l=0,1,2,3} res_j(k-l)$, $res_j(k+1) \ge 0.9 * \max_{l=0,1,2,3} res_j(k-l)$.

(ii) $res_j(k+1-l) \ge 0.8 * res_j(k-l)$, $l = 0, 1, 2, 3$.

(iii) $res_j(k+1) < res_j(k)$, $res_j(k) > res_j(k-1)$, $res_j(k-1) < res_j(k-2)$, $res_j(k-2) > res_j(k-3)$,

In this case, EB13 again accepts $\lambda_j(k+1)$ and issues a warning. Note that condition (iii) is designed to terminate the computation if the residuals start to oscillate, which we found could happen in practice.

In the event of a problem failing to converge, we ensure a finite termination of the computation by

imposing a limit on the maximum number of matrix-vector products allowed. This limit is determined by the control parameter `ICNTL(5)`, which has a default value of 10000. 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. For details, see Appendix.

### 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 $\lambda_r$ to be real (see (2.3)), when the reference eigenvalue is complex, Saad replaces $\lambda_r$ by the point $\gamma$ on the real line having the same convergence ratio as $\lambda_r$ 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 $\lambda_r$. 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 $b = \max_i \dfrac{|aIm(\lambda_i)|}{\sqrt{a^2 - (Re(\lambda_i) - d)^2}}$ 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 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, $\lambda_r$). The ellipse is then 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 Appendix 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 $k$th 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 $res_r \le$ `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 kernels 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 $res_i$. Only if $\mathbf{x}_i$ is accepted is $res_{i+1}$ computed.

## 4 Numerical experiments

In this section we report the results of using `EB13` to compute the right-most 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 modulus (algorithm S1) and subspace iteration combined with Chebychev acceleration for computing the right-most (or left-most) eigenvalues (algorithm S2).

Most of the test problems were taken from the the Harwell-Boeing sparse matrix collection (Duff, Grimes, and Lewis 1992), which is widely used in testing and evaluating sparse matrix algorithms. The problems we selected and some of their characteristics are given in Table 4.1. The test problems GRT200 and GRT400 were supplied by Garratt (1990). We tested each of the Arnoldi based algorithms described in Section 2 on a subset of the test matrices (the entries marked with $^+$ in Table 4.1) then, based on our findings for this subset, we 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 with eigenvalues which are not favourably separated. 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 $\lambda_1$ 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.

Table 4.1. Characteristics of the matrices used for performance testing.

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

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.

Throughout this section, $r$ denotes the number of eigenpairs sought and $n_b$ denotes the blocksize. For `EB13`, $m$ is the number of Arnoldi steps used on each iteration (so that the order of the Hessenberg matrix is $mn_b$), and for `EB12`, $m$ is the subspace dimension. For `EB13`, $Res_{\max}$ denotes the maximum of the scaled eigenvector residuals given by (3.3), and for `EB12`, $Res_{\max}$ is given by

$$Res_{\max} = \max_{1 \le i \le r} \frac{\|(\mathbf{A}\mathbf{y}_i - \lambda_i \mathbf{y}_i)\|_2}{\|(\mathbf{A}\mathbf{y})_i\|_2}. \tag{4.1}$$

We have this difference since the code `EB12` uses (3.2) in place of (3.1). In each test, the implicit deflation techniques outlined in Section 2 are used wherever appropriate. The convergence tolerance `CNTL(1)` is taken to be $u*10^k$, where $u$ is the machine precision, and $k=3$ for the RS/6000 and $k=4$ Cray 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 entry marked by ** indicates the default number of matrix-vector products was insufficient. All CPU timings are in seconds. On each machine, the implementations of the BLAS provided by the manufacturer are employed.

In Tables 4.2a and 4.2b 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 4.1. Table 4.2a gives results for the Cray Y-MP and Table 4.2b for the IBM RS/6000. For the matrices GRT200, GRT400, and PORES3 the right-most eigenvalues are not those of largest modulus, 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. The additional matrix-vector products used by the Arnoldi based algorithms for this problem are the products to compute $\|\mathbf{A}\|_F$. Compared with the unblocked methods, the block methods generally required more matrix-vector products but a smaller number of iterations for convergence. 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. On both the Y-MP and the RS/6000, algorithm A2 terminated in less time than algorithm A3, although 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 4.3a and 4.3b 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 $\lambda_r$ and $\lambda_{r+1}$ are a complex conjugate pair of eigenvalues), and the number of Arnoldi steps to be $m=4$, 6, or 8. These choices were made for $m$ since experience has shown 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 AB2 with $m=4$ did not converge within the default number of matrix-vector products and, for a range of values of $m$, 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 4.4a and 4.4b. 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 since convergence was too slow. Algorithm AB3 appeared to be less sensitive to the choice of $n_b$ but using $n_b>r$ in place of $n_b=r$ generally resulted in smaller maximum residuals. 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 4.1 which were not used in Tables 4.2 and 4.3. The results are in Tables 4.5a and 4.5b. They 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. This 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 will converge 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 4.2a. 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 | $Res_{\max}$ |
|---|---|---|---|---|---|---|---|---|
| HOR131 | S1 | 10 | | 6 | 509 | 0.19 | 0.24 | 5.47E-13 |
| | S2 | 10 | | 4 | 449 | 0.17 | 0.21 | 5.10E-13 |
| | A1 | 10 | | 5 | 485 | 1.32 | 1.36 | 5.69E-13 |
| | AB1 | 8 | 2 | 5 | 516 | 1.25 | 1.33 | 4.68E-12 |
| | A2 | 10 | | 2 | 495 | 1.35 | 1.38 | 2.86E-14 |
| | AB2 | 8 | 2 | 2 | 548 | 1.30 | 1.35 | 3.01E-14 |
| | A3 | 10 | | 2 | 815 | 2.19 | 2.25 | 3.85E-14 |
| | AB3 | 8 | 2 | 2 | 1028 | 1.91 | 2.00 | 1.03E-13 |
| GRE1107 | S1 | 8 | | 19 | 2991 | 1.44 | 1.77 | 3.90E-12 |
| | S2 | 8 | | 28 | 5079 | 2.39 | 3.06 | 2.21E-12 |
| | A1 | 12 | | 15 | 1289 | 4.43 | 4.63 | 1.66E-05† |
| | AB1 | 6 | 4 | 58 | 2503 | 4.93 | 7.27 | 5.16E-12 |
| | A2 | 12 | | 7 | 2050 | 7.04 | 7.30 | 9.57E-15 |
| | AB2 | 6 | 4 | 5 | 2387 | 4.83 | 5.30 | 7.08E-12 |
| | A3 | 12 | | 2 | 1572 | 5.36 | 5.50 | 6.32E-12 |
| | AB3 | 6 | 4 | 2 | 1959 | 4.44 | 4.75 | 1.23E-12 |
| IMPCOLA | S1 | 10 | | 1 | 29 | 0.01 | 0.02 | 1.42E-13 |
| | S2 | 10 | | 1 | 29 | 0.01 | 0.02 | 1.42E-13 |
| | A1 | 10 | | 1 | 218 | 0.08 | 0.10 | 2.96E-14 |
| | A2 | 10 | | 1 | 218 | 0.08 | 0.10 | 2.96E-14 |
| | A3 | 10 | | 1 | 218 | 0.08 | 0.10 | 2.96E-14 |
| GRT200 | S2 | 8 | | 12 | 6535 | 2.03 | 2.21 | 3.17E-10† |
| | A2 | 8 | | 8 | 1802 | 0.88 | 0.99 | 1.42E-14 |
| | AB2 | 4 | 4 | 6 | 2276 | 0.36 | 0.51 | 4.35E-15 |
| | A3 | 8 | | 5 | 2269 | 1.13 | 1.65 | 6.67E-15 |
| | AB3 | 4 | 4 | 5 | 3656 | 0.54 | 0.73 | 5.11E-15 |
| GRT400 | S2 | 8 | | 25 | 13703 | 8.40 | 8.86 | 7.02E-09† |
| | A2 | 8 | | 8 | 2812 | 2.78 | 3.00 | 9.34E-14 |
| | AB2 | 4 | 4 | 7 | 3948 | 1.29 | 1.61 | 1.63E-14 |
| | A3 | 8 | | 6 | 3914 | 3.80 | 4.10 | 5.46E-15 |
| | AB3 | 4 | 4 | 6 | 6392 | 1.98 | 2.39 | 5.89E-15 |
| PORES3 | S2 | 12 | | ** | ** | ** | ** | ** |
| | A2 | 12 | | 13 | 6218 | 12.82 | 13.39 | 5.21E-09† |
| | AB2 | 6 | 4 | 10 | 13292 | 7.70 | 8.86 | 3.37E-12 |
| | A3 | 12 | | 7 | 7899 | 16.25 | 16.99 | 3.86E-13 |
| | AB3 | 6 | 4 | 6 | 10560 | 6.22 | 7.11 | 5.52E-12 |

Table 4.2b. 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 | $Res_{\max}$ |
|---|---|---|---|---|---|---|---|---|
| HOR131 | S1 | 10 | | 7 | 749 | 0.56 | 0.75 | 1.08E-16 |
| | S2 | 10 | | 4 | 429 | 0.28 | 0.49 | 1.87E-12 |
| | A1 | 10 | | 5 | 485 | 0.81 | 0.89 | 5.69E-13 |
| | AB1 | 8 | 2 | 6 | 532 | 0.92 | 1.19 | 2.79E-14 |
| | A2 | 10 | | 2 | 495 | 0.95 | 0.98 | 2.41E-16 |
| | AB2 | 8 | 2 | 2 | 548 | 0.98 | 1.05 | 3.33E-16 |
| | A3 | 10 | | 2 | 815 | 1.50 | 1.75 | 2.82E-16 |
| | AB3 | 8 | 2 | 2 | 1028 | 1.45 | 1.73 | 6.55E-16 |
| GRE1107 | S1 | 8 | | 11 | 2935 | 2.86 | 3.47 | 5.84E-13 |
| | S2 | 8 | | 16 | 5127 | 4.83 | 7.35 | 8.97E-13 |
| | A1 | 12 | | 15 | 1289 | 3.67 | 4.63 | 4.46E-05† |
| | AB1 | 6 | 4 | 62 | 2599 | 4.99 | 16.95 | 1.74E-12 |
| | A2 | 12 | | 7 | 2050 | 5.36 | 6.73 | 3.18E-16 |
| | AB2 | 6 | 4 | 5 | 2387 | 4.71 | 7.37 | 4.50E-16 |
| | A3 | 12 | | 3 | 2156 | 5.81 | 7.17 | 1.09E-11† |
| | AB3 | 6 | 4 | 2 | 1959 | 4.68 | 6.26 | 1.22E-12 |
| IMPCOLA | S1 | 10 | | 1 | 29 | 0.01 | 0.02 | 1.42E-13 |
| | S2 | 10 | | 1 | 29 | 0.01 | 0.02 | 1.42E-13 |
| | A1 | 10 | | 1 | 218 | 0.08 | 0.10 | 2.94E-14 |
| | A2 | 10 | | 1 | 218 | 0.08 | 0.10 | 2.94E-14 |
| | A3 | 10 | | 1 | 218 | 0.08 | 0.10 | 2.94E-14 |
| GRT200 | S2 | 8 | | 16 | 9079 | 1.74 | 2.45 | 3.87E-12† |
| | A2 | 8 | | 8 | 1802 | 0.66 | 1.02 | 4.73E-16 |
| | AB2 | 4 | 4 | 6 | 2276 | 0.36 | 0.83 | 3.90E-16 |
| | A3 | 8 | | 5 | 2269 | 0.86 | 1.25 | 2.11E-16 |
| | AB3 | 4 | 4 | 5 | 3752 | 0.61 | 1.35 | 1.20E-16 |
| GRT400 | S2 | 8 | | 14 | 9271 | 3.54 | 5.11 | 2.29E-11† |
| | A2 | 8 | | 10 | 3621 | 2.76 | 3.90 | 9.30E-14 |
| | AB2 | 4 | 4 | 8 | 6648 | 1.93 | 4.29 | 2.51E-17 |
| | A3 | 8 | | 6 | 3914 | 2.63 | 3.92 | 3.36E-16 |
| | AB3 | 4 | 4 | 6 | 6428 | 1.99 | 4.20 | 2.16E-16 |
| PORES3 | S2 | 12 | | ** | ** | ** | ** | ** |
| | A2 | 12 | | 13 | 6218 | 9.62 | 12.55 | 5.20E-09† |
| | AB2 | 6 | 4 | 10 | 16516 | 11.34 | 20.54 | 1.71E-14 |
| | A3 | 12 | | 7 | 8295 | 12.14 | 15.72 | 1.24E-13 |
| | AB3 | 6 | 4 | 7 | 12524 | 8.38 | 14.55 | 8.59E-14 |

Table 4.3a. 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 | $Res_{\max}$ |
|---|---|---|---|---|---|---|---|---|
| HOR131 | S1 | 40 | | 7 | 835 | 0.22 | 0.53 | 5.29E-12 |
| | | 24 | | 11 | 1197 | 0.25 | 0.48 | 5.77E-13 |
| | S2 | 40 | | 18 | 4073 | 0.62 | 1.49 | 3.29E-12 |
| | | 24 | | 18 | 3311 | 0.64 | 1.02 | 1.77E-12 |
| | A1 | 40 | | 6 | 673 | 1.78 | 1.99 | 9.09E-13 |
| | | 24 | | 23 | 948 | 2.41 | 2.71 | 1.79E-12 |
| | AB1 | 4 | 10 | 14 | 1004 | 1.38 | 2.25 | 5.60E-12 |
| | | 6 | 10 | 7 | 864 | 1.33 | 2.31 | 3.17E-13 |
| | A2 | 40 | | 3 | 590 | 1.57 | 1.70 | 1.15E-12 |
| | | 24 | | 11 | 871 | 2.28 | 2.57 | 2.22E-13 |
| | AB2 | 4 | 10 | 3 | 845 | 1.33 | 1.56 | 0.89E-13 |
| | | 6 | 10 | 4 | 1002 | 1.41 | 1.90 | 3.08E-13 |
| | A3 | 40 | | 2 | 1117 | 2.91 | 3.06 | 1.84E-12 |
| | | 24 | | 2 | 888 | 2.32 | 2.40 | 9.11E-13 |
| | AB3 | 4 | 10 | 3 | 1764 | 1.56 | 1.86 | 2.78E-13 |
| | | 6 | 10 | 2 | 1514 | 1.49 | 1.84 | 2.85E-12 |
| GRE1107 | S1 | 40 | | 38 | 10593 | 1.74 | 5.26 | 4.66E-12 |
| | | 24 | | 34 | 7512 | 1.60 | 2.86 | 5.39E-12 |
| | S2 | 40 | | 50 | 12728 | 2.08 | 7.69 | 5.91E-12 |
| | | 24 | | 49 | 9397 | 2.02 | 4.40 | 5.85E-12 |
| | A1 | 40 | | 74 | 4280 | 15.51 | 25.61 | 1.84E-04† |
| | | 24 | | 72 | 3074 | 10.38 | 15.08 | 1.20E-03† |
| | AB1 | 4 | 10 | 101 | 5157 | 5.78 | 22.07 | 1.74E-12 |
| | | 6 | 10 | 51 | 4177 | 5.13 | 23.01 | 1.84E-12 |
| | A2 | 40 | | 38 | 8056 | 25.25 | 30.88 | 5.69E-08† |
| | | 24 | | 47 | 7510 | 27.27 | 27.35 | 4.90E-08† |
| | AB2 | 4 | 10 | 9 | 6036 | 7.77 | 9.65 | 1.11E-09† |
| | | 6 | 10 | 4 | 2812 | 4.45 | 6.32 | 1.36E-13 |
| | A3 | 40 | | 2 | 2762 | 9.30 | 9.75 | 1.04E-14 |
| | | 24 | | 2 | 2091 | 7.05 | 7.31 | 8.26E-14 |
| | AB3 | 4 | 10 | 4 | 6317 | 5.71 | 7.02 | 1.05E-13 |
| | | 6 | 10 | 2 | 3237 | 4.57 | 5.55 | 5.99E-14 |
| IMPCOLA | S2 | 40 | | 33 | 2561 | 0.06 | 1.85 | 1.48E-10† |
| | | 24 | | 92 | 3801 | 0.10 | 1.87 | 5.18E-10† |
| | A2 | 40 | | 6 | 469 | 0.17 | 0.41 | 1.06E-12 |
| | | 24 | | 22 | 797 | 0.28 | 0.62 | 2.61E-06† |
| | AB2 | 6 | 11 | 6 | 786 | 0.10 | 0.99 | 3.28E-13 |
| | | 8 | 11 | 4 | 675 | 0.10 | 1.26 | 2.23E-14 |
| | A3 | 40 | | 2 | 454 | 0.15 | 0.27 | 1.43E-14 |
| | | 24 | | 3 | 461 | 0.16 | 0.22 | 5.87E-13 |
| | AB3 | 6 | 11 | 4 | 1142 | 0.12 | 1.06 | 1.08E-13 |
| | | 8 | 11 | 3 | 1098 | 0.10 | 1.02 | 6.07E-15 |
| PORES3 | S2 | 40 | | ** | ** | ** | ** | ** |
| | A2 | 40 | | 47 | 14257 | 28.83 | 31.83 | 4.39E-06† |
| | | 64 | | 35 | 15610 | 31.79 | 37.00 | 1.27E-06† |
| | AB2 | 4 | 10 | 10 | 25041 | 8.06 | 10.33 | 1.61E-12 |
| | | 6 | 10 | 9 | 22774 | 6.43 | 8.99 | 3.69E-12 |
| | A3 | 40 | | 5 | 11793 | 22.72 | 24.08 | 1.73E-12 |
| | | 64 | | 4 | 11089 | 21.29 | 22.91 | 1.09E-12 |
| | AB3 | 4 | 10 | 9 | 25412 | 6.65 | 9.16 | 4.70E-12 |
| | | 6 | 10 | 7 | 24062 | 6.35 | 9.38 | 3.99E-14 |

Table 4.3b. 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 | $Res_{\max}$ |
|---|---|---|---|---|---|---|---|---|
| HOR131 | S1 | 40 | | 6 | 918 | 0.77 | 2.18 | 5.29E-12† |
| | | 24 | | 10 | 1136 | 0.80 | 1.78 | 1.23E-13 |
| | S2 | 40 | | 13 | 4543 | 3.62 | 6.91 | 4.77E-13 |
| | | 24 | | 15 | 3046 | 2.30 | 3.85 | 6.79E-13 |
| | A1 | 40 | | 6 | 675 | 1.22 | 2.28 | 1.39E-12 |
| | | 24 | | 23 | 948 | 1.72 | 3.02 | 2.13E-12 |
| | AB1 | 4 | 10 | 17 | 1124 | 1.58 | 5.48 | 4.29E-13 |
| | | 6 | 10 | 7 | 864 | 1.28 | 5.39 | 1.39E-13 |
| | A2 | 40 | | 3 | 594 | 1.18 | 1.74 | 6.35E-13 |
| | | 24 | | 11 | 880 | 1.88 | 2.72 | 2.67E-14 |
| | AB2 | 4 | 10 | 3 | 871 | 1.26 | 2.19 | 9.40E-15 |
| | | 6 | 10 | 4 | 1034 | 1.35 | 3.83 | 1.68E-15 |
| | A3 | 40 | | 2 | 1191 | 2.30 | 3.13 | 6.36E-13 |
| | | 24 | | 2 | 934 | 1.79 | 2.20 | 9.23E-14 |
| | AB3 | 4 | 10 | 3 | 1884 | 1.95 | 3.33 | 8.92E-16 |
| | | 6 | 10 | 2 | 1614 | 1.94 | 3.44 | 2.64E-13 |
| GRE1107 | S1 | 40 | | 22 | 10640 | 13.42 | 25.26 | 1.05E-12 |
| | | 24 | | 26 | 8127 | 7.73 | 13.07 | 1.98E-12 |
| | S2 | 40 | | 25 | 12671 | 15.87 | 32.85 | 1.36E-12 |
| | | 24 | | 25 | 8933 | 8.31 | 15.93 | 2.50E-12† |
| | A1 | 40 | | 55 | 3421 | 9.61 | 31.55 | 3.42E-07† |
| | | 24 | | 59 | 2651 | 7.01 | 15.38 | 1.14E-04† |
| | AB1 | 4 | 10 | 121 | 5197 | 6.84 | 57.11 | 6.64E-11† |
| | | 6 | 10 | 63 | 4897 | 6.62 | 80.90 | 1.23E-12 |
| | A2 | 40 | | 30 | 6776 | 19.29 | 35.44 | 1.86E-06† |
| | | 24 | | 42 | 7376 | 21.16 | 32.55 | 6.04E-08† |
| | AB2 | 4 | 10 | 5 | 3367 | 5.47 | 10.49 | 1.23E-13 |
| | | 6 | 10 | 4 | 2962 | 4.98 | 11.73 | 3.02E-14 |
| | A3 | 40 | | 2 | 2762 | 8.08 | 10.90 | 2.50E-15 |
| | | 24 | | 2 | 2091 | 5.95 | 7.32 | 5.86E-15 |
| | AB3 | 4 | 10 | 4 | 6317 | 8.38 | 16.21 | 8.88E-16 |
| | | 6 | 10 | 2 | 3237 | 5.32 | 10.23 | 4.22E-15 |
| IMPCOLA | S2 | 40 | | 39 | 2965 | 0.31 | 7.71 | 7.93E-12† |
| | | 24 | | 100 | 4169 | 0.50 | 6.39 | 4.68E-11† |
| | A2 | 40 | | 13 | 757 | 0.24 | 2.30 | 1.58E-08† |
| | | 24 | | 20 | 758 | 0.27 | 1.32 | 9.50E-14 |
| | AB2 | 6 | 11 | 5 | 732 | 0.14 | 3.75 | 3.98E-13 |
| | | 8 | 11 | 4 | 600 | 0.19 | 5.65 | 3.09E-14 |
| | A3 | 40 | | 2 | 492 | 0.16 | 0.69 | 1.47E-14 |
| | | 24 | | 3 | 477 | 0.13 | 0.41 | 6.67E-14 |
| | AB3 | 6 | 11 | 4 | 1307 | 0.23 | 3.22 | 2.13E-16 |
| | | 8 | 11 | 3 | 1252 | 0.32 | 4.64 | 6.52E-17 |
| PORES3 | S2 | 40 | | ** | ** | ** | ** | ** |
| | A2 | 40 | | 27 | 9965 | 17.55 | 27.61 | 4.00E-05† |
| | | 64 | | 33 | 17781 | 29.33 | 56.05 | 1.97E-06† |
| | AB2 | 4 | 10 | ** | ** | ** | ** | ** |
| | | 6 | 10 | 9 | 22774 | 11.84 | 29.77 | 2.10E-12 |
| | A3 | 40 | | 5 | 11916 | 20.42 | 28.54 | 4.33E-13 |
| | | 64 | | 4 | 11245 | 19.87 | 27.37 | 4.32E-13 |
| | AB3 | 4 | 10 | 10 | 28632 | 14.75 | 32.43 | 8.35E-13 |
| | | 6 | 10 | 7 | 23612 | 11.77 | 30.03 | 6.97E-14 |

Table 4.4a. 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 | $Res_{\max}$ |
|---|---|---|---|---|---|---|---|---|
| HOR131 | AB2 | 4 | 8 | 3 | 730 | 1.30 | 1.54 | 9.38E-13 |
| | | 4 | 10 | 3 | 845 | 1.33 | 1.56 | 0.89E-13 |
| | AB3 | 4 | 8 | 3 | 1450 | 1.51 | 1.79 | 2.58E-13 |
| | | 4 | 10 | 3 | 1764 | 1.56 | 1.86 | 2.78E-13 |
| GRE1107 | AB2 | 6 | 8 | 11 | 4591 | 9.12 | 12.40 | 6.48E-10† |
| | | 6 | 10 | 4 | 2812 | 4.45 | 6.32 | 1.36E-13 |
| | AB3 | 6 | 8 | 2 | 2811 | 4.50 | 5.42 | 1.64E-12 |
| | | 6 | 10 | 2 | 3237 | 4.57 | 5.55 | 5.99E-14 |
| IMPCOLA | AB2 | 8 | 8 | 32 | 2904 | 0.30 | 8.69 | 6.85E-07† |
| | | 8 | 11 | 4 | 675 | 0.10 | 1.26 | 2.23E-14 |
| | AB3 | 8 | 8 | 3 | 879 | 0.11 | 0.92 | 7.302E-14 |
| | | 8 | 11 | 3 | 1098 | 0.10 | 1.02 | 6.07E-15 |
| PORES3 | AB2 | 6 | 8 | 11 | 22129 | 9.65 | 12.63 | 4.38E-12 |
| | | 6 | 10 | 9 | 22774 | 6.43 | 8.99 | 3.69E-12 |
| | AB3 | 6 | 8 | 7 | 24076 | 7.55 | 10.37 | 1.23E-14 |
| | | 6 | 10 | 7 | 24062 | 6.35 | 9.38 | 3.99E-14 |

Table 4.4b. 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 | $Res_{\max}$ |
|---|---|---|---|---|---|---|---|---|
| HOR131 | AB2 | 4 | 8 | 3 | 756 | 1.13 | 1.76 | 8.80E-14 |
| | | 4 | 10 | 3 | 871 | 1.26 | 2.19 | 9.40E-15 |
| | AB3 | 4 | 8 | 3 | 1570 | 1.70 | 2.66 | 1.14E-15 |
| | | 4 | 10 | 3 | 1884 | 1.95 | 3.33 | 8.92E-16 |
| GRE1107 | AB2 | 6 | 8 | 8 | 3877 | 7.52 | 16.39 | 7.25E-11† |
| | | 6 | 10 | 4 | 2962 | 4.98 | 11.73 | 3.02E-14 |
| | AB3 | 6 | 8 | 3 | 4939 | 7.06 | 14.02 | 2.69E-12† |
| | | 6 | 10 | 2 | 3237 | 5.32 | 10.23 | 4.22E-15 |
| IMPCOLA | AB2 | 8 | 8 | 9 | 996 | 0.18 | 5.52 | 1.36E-06† |
| | | 8 | 11 | 4 | 600 | 0.19 | 5.65 | 3.09E-14 |
| | AB3 | 8 | 8 | 3 | 991 | 0.20 | 2.12 | 1.22E-12 |
| | | 8 | 11 | 3 | 1252 | 0.32 | 4.64 | 6.52E-17 |
| PORES3 | AB2 | 6 | 8 | 11 | 22129 | 13.03 | 28.21 | 1.61E-13 |
| | | 6 | 10 | 9 | 22774 | 11.84 | 29.77 | 2.10E-12 |
| | AB3 | 6 | 8 | 7 | 22676 | 12.23 | 26.78 | 4.89E-13 |
| | | 6 | 10 | 7 | 23612 | 11.77 | 30.03 | 6.97E-14 |

Table 4.5a. 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 | $Res_{\max}$ |
|---|---|---|---|---|---|---|---|---|---|
| IMPCOLB | A3 | 8 | 40 | | 2 | 1715 | 0.32 | 0.46 | 4.02E-14 |
| | AB2 | 8 | 5 | 8 | 4 | 1346 | 0.05 | 0.25 | 1.32E-12 |
| | AB3 | 8 | 5 | 8 | 2 | 1427 | 0.05 | 0.16 | 8.48E-14 |
| IMPCOLC | A3 | 14 | 56 | | 2 | 2164 | 0.56 | 0.81 | 1.64E-14 |
| | AB2 | 14 | 4 | 14 | 4 | 1752 | 0.09 | 0.53 | 2.32E-14 |
| | AB3 | 14 | 4 | 14 | 2 | 1943 | 0.08 | 0.34 | 4.96E-13 |
| IMPCOLD | A3 | 8 | 40 | | 2 | 2081 | 1.73 | 1.99 | 2.64E-14 |
| | AB2 | 8 | 6 | 10 | 5 | 3265 | 0.65 | 1.59 | 2.42E-12 |
| | AB3 | 8 | 6 | 10 | 3 | 5215 | 0.81 | 1.58 | 2.70E-14 |
| IMPCOLE | A3 | 8 | 40 | | 2 | 1880 | 1.41 | 1.59 | 1.28E-14 |
| | AB2 | 8 | 6 | 10 | 2 | 755 | 0.22 | 0.48 | 1.35E-14 |
| | AB3 | 8 | 6 | 10 | 2 | 2355 | 0.35 | 0.68 | 7.67E-15 |
| NNC261 | A3 | 8 | 24 | | 2 | 876 | 0.80 | 0.90 | 7.03E-14 |
| | AB2 | 8 | 4 | 10 | 2 | 751 | 0.29 | 0.50 | 4.87E-13 |
| | AB3 | 8 | 4 | 10 | 2 | 1551 | 0.37 | 0.61 | 6.51E-13 |
| NNC666 | A3 | 8 | 24 | | 2 | 1327 | 3.17 | 3.33 | 1.13E-13 |
| | AB2 | 8 | 4 | 10 | 3 | 1461 | 1.89 | 2.24 | 1.21E-12 |
| | AB3 | 8 | 4 | 10 | 3 | 3556 | 2.32 | 2.93 | 1.01E-12 |
| NNC1374 | A3 | 8 | 24 | | 2 | 2058 | 10.39 | 10.80 | 1.00E-12 |
| | AB2 | 8 | 4 | 10 | 4 | 2690 | 8.05 | 9.25 | 6.10E-13 |
| | AB3 | 8 | 4 | 10 | 3 | 4264 | 8.66 | 10.11 | 1.61E-13 |
| PORES2 | A3 | 4 | 16 | | 2 | 1863 | 10.13 | 10.33 | 1.21E-12 |
| | AB2 | 4 | 10 | 6 | 8 | 4203 | 11.71 | 15.36 | 2.11E-13 |
| | AB3 | 4 | 10 | 6 | 3 | 6378 | 11.90 | 14.28 | 7.61E-13 |
| WEST0156 | A3 | 8 | 48 | | 4 | 2415 | 0.58 | 0.93 | 1.78E-12 |
| | AB2 | 8 | 6 | 10 | 8 | 1387 | 0.09 | 0.98 | 2.67E-13 |
| | AB3 | 8 | 6 | 10 | 5 | 4516 | 0.16 | 0.83 | 1.87E-13 |
| WEST0167 | A3 | 8 | 48 | | 2 | 1064 | 0.33 | 0.50 | 4.95E-13 |
| | AB2 | 8 | 8 | 10 | 3 | 645 | 0.09 | 1.20 | 8.64E-13 |
| | AB3 | 8 | 8 | 10 | 2 | 1737 | 0.12 | 0.91 | 8.80E-14 |
| WEST0381 | A3 | 8 | 24 | | 2 | 1366 | 1.72 | 1.84 | 9.25E-15 |
| | AB2 | 8 | 4 | 10 | 3 | 1227 | 0.64 | 0.95 | 2.07E-13 |
| | AB3 | 8 | 4 | 10 | 3 | 3271 | 0.90 | 1.31 | 5.59E-15 |
| WEST2021 | A3 | 4 | 28 | | 5 | 9781 | 44.81 | 47.34 | 4.00E-17 |
| | AB2 | 4 | 11 | 5 | 5 | 3500 | 11.07 | 13.71 | 2.47E-15 |
| | AB3 | 4 | 11 | 5 | 2 | 4136 | 11.40 | 13.12 | 7.16E-14 |

Table 4.5b. 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 | $Res_{\max}$ |
|---|---|---|---|---|---|---|---|---|---|
| IMPCOLB | A3 | 8 | 40 | | 2 | 1715 | 0.20 | 0.71 | 3.40E-16 |
| | AB2 | 8 | 5 | 8 | 4 | 1346 | 0.09 | 0.72 | 1.61E-13 |
| | AB3 | 8 | 5 | 8 | 2 | 1427 | 0.09 | 0.47 | 8.48E-14 |
| IMPCOLC | A3 | 14 | 56 | | 2 | 2326 | 0.58 | 1.49 | 1.96E-16 |
| | AB2 | 14 | 4 | 14 | 4 | 1752 | 0.08 | 1.67 | 7.85E-15 |
| | AB3 | 14 | 4 | 14 | 2 | 1943 | 0.21 | 1.12 | 5.60E-13 |
| IMPCOLD | A3 | 8 | 40 | | 2 | 2081 | 1.50 | 2.90 | 1.19E-16 |
| | AB2 | 8 | 6 | 10 | 6 | 4151 | 1.05 | 6.62 | 2.07E-15 |
| | AB3 | 8 | 6 | 10 | 3 | 5215 | 1.46 | 5.91 | 6.41E-14 |
| IMPCOLE | A3 | 8 | 40 | | 2 | 1880 | 1.17 | 1.92 | 1.86E-16 |
| | AB2 | 8 | 6 | 10 | 2 | 755 | 0.23 | 1.24 | 2.24E-16 |
| | AB3 | 8 | 6 | 10 | 2 | 2355 | 0.52 | 2.04 | 1.15E-16 |
| NNC261 | A3 | 8 | 24 | | 2 | 945 | 0.71 | 0.96 | 1.87E-15 |
| | AB2 | 8 | 4 | 10 | 2 | 751 | 0.29 | 0.77 | 1.55E-14 |
| | AB3 | 8 | 4 | 10 | 2 | 1551 | 0.42 | 1.17 | 9.86E-15 |
| NNC666 | A3 | 8 | 24 | | 2 | 1396 | 2.68 | 3.34 | 3.83E-16 |
| | AB2 | 8 | 4 | 10 | 2 | 1156 | 1.66 | 2.70 | 1.39E-12 |
| | AB3 | 8 | 4 | 10 | 3 | 3556 | 2.92 | 6.01 | 2.92E-14 |
| NNC1374 | A3 | 8 | 24 | | 2 | 2127 | 8.76 | 10.32 | 4.38E-14 |
| | AB2 | 8 | 4 | 10 | 4 | 2699 | 7.10 | 11.14 | 1.10E-13 |
| | AB3 | 8 | 4 | 10 | 3 | 4264 | 9.12 | 15.74 | 2.02E-15 |
| PORES2 | A3 | 4 | 16 | | 3 | 2500 | 10.66 | 12.68 | 9.46E-13 |
| | AB2 | 4 | 10 | 6 | 8 | 3466 | 9.54 | 21.61 | 5.71E-14 |
| | AB3 | 4 | 10 | 6 | 3 | 6378 | 14.58 | 26.19 | 4.22E-14 |
| WEST0156 | A3 | 8 | 64 | | 2 | 2525 | 0.56 | 2.07 | 2.58E-16 |
| | AB2 | 8 | 6 | 10 | 7 | 1332 | 0.07 | 3.13 | 1.98E-13 |
| | AB3 | 8 | 6 | 10 | 5 | 4916 | 0.32 | 3.31 | 2.88E-14 |
| WEST0167 | A3 | 8 | 64 | | 1 | 239 | 0.10 | 0.53 | 2.09E-17 |
| | AB2 | 8 | 8 | 10 | 4 | 843 | 0.15 | 4.07 | 1.47E-15 |
| | AB3 | 8 | 8 | 10 | 2 | 1947 | 0.21 | 2.55 | 2.67E-14 |
| WEST0381 | A3 | 8 | 24 | | 2 | 1366 | 1.31 | 1.91 | 2.88E-16 |
| | AB2 | 8 | 4 | 10 | 3 | 1227 | 0.73 | 1.84 | 7.57E-14 |
| | AB3 | 8 | 4 | 10 | 4 | 5591 | 2.13 | 5.51 | 4.06E-17 |
| WEST2021 | A3 | 4 | 28 | | 5 | 9781 | 39.85 | 57.97 | 2.94E-17 |
| | AB2 | 4 | 11 | 5 | 5 | 3500 | 10.78 | 21.29 | 1.36E-15 |
| | AB3 | 4 | 11 | 5 | 3 | 6791 | 15.75 | 33.12 | 5.73E-16 |

## 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 the Harwell Subroutine Library and anyone interested in using the code should contact the HSL Manager: Ms L Thick, Theoretical Studies Department, AEA Technology, Building 424.4, Harwell, Oxfordshire, OX11 0RA, England, tel (44) 235 432688, fax (44) 235 436579, 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.

## 8 References

Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, H., McKenney, A., Ostrouchov, S., and Sorensen, D. (1992). LAPACK User's Guide. SIAM.

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.* 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 Fraysée, 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.

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 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.

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.

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

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.

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.

Stewart, W. J. and Jennings, A. (1981). A simultaneous iteration algorithm for real matrices. *ACM Trans. Math. Softw.* **7**, 184-198.

# 9 APPENDIX: Specification sheets for the EB13 package