Technical Report
RAL-TR-96-023

CLRC

An Evaluation of Arnoldi based Software for Sparse Nonsymmetric Eigenproblems

R B Lehoucq and J A Scott

March 1996

COUNCIL FOR THE CENTRAL LABORATORY OF THE RESEARCH COUNCILS

ISSN 1358-6254

RAL-TR-96-023

An evaluation of Arnoldi based software for sparse nonsymmetric eigenproblems.*

by

R. B. Lehoucq[1] and J. A. Scott[2]

Abstract

In recent years, high quality software for computing selected eigenvalues of large sparse nonsymmetric matrices has started to become publicly available. In this study we consider software which implements algorithms based on the original method of Arnoldi. We briefly describe the software which is available. We look at the key features of the codes and the important differences between them. Then, using a wide range of practical problems, we compare the performance of the codes in terms of storage requirements, execution times, accuracy, and reliability, and consider their suitability for solving large-scale industrial problems. Finally, we point to possible future directions of research for improving Arnoldi based software.

---

* Current reports available by anonymous ftp from `seamus.cc.rl.ac.uk` (internet 130.246.8.32) in the directory `pub/reports`.

[1]Argonne National Laboratory,
Mathematics and Computer Science Division,
Argonne, IL 60439, U.S.A.

[2]Computing and Information Systems Department,
Atlas Centre, Rutherford Appleton Laboratory,
Didcot, Oxfordshire OX11 0QX, England.

March 18, 1996.

Contents

1 Introduction

The development of efficient numerical methods for solving large sparse nonsymmetric eigenvalue problems has been the subject of much interest and research effort during the last decade. Several classes of methods have received attention and support. These include subspace (or simultaneous) iteration methods, Arnoldi's method and its variants, the (nonsymmetric) Lanczos method, and, recently, the Jacobi-Davidson method (Sleijpen and Van der Vorst, 1995). Software is gradually becoming available and we feel that it is now time to evaluate this software to see how robust it is and how suitable it is for solving today's large scale nonsymmetric eigenvalue problems. In this report, we focus attention on software which implements Arnoldi type methods. In a separate report, we examine subspace iteration software (Lehoucq and Scott, 1996*b*). The results of our studies of subspace iteration and Arnoldi software are brought together, summarised, and compared in Lehoucq and Scott (1996*a*). In the future, we plan to extend our study to software which uses the Lanczos method and, once it becomes available, to software for the Jacobi-Davidson method.

In addition to numerous research codes, several library-quality packages which employ Arnoldi iteration techniques have been developed for the standard eigenvalue problem $\mathbf{A}\mathbf{z} = \lambda\mathbf{z}$. We are interested in codes which are written either in the C programming language or in FORTRAN. In addition, for inclusion in our study, the codes must be available either in the public domain or under licence. There are currently (to the authors' knowledge) three such packages (we apologise if there are any other packages which meet our criteria but which we are not aware of). These are the ARNCHEB package of Braconnier (1993), the ARPACK package (Lehoucq, Sorensen, Vu and Yang, 1995) and the Harwell Subroutine Library code EB13 (Scott, 1995). The reports and papers which accompany each of these codes provide limited numerical results illustrating their use but results comparing their performances have not been published. Our aims are to review, compare, and evaluate the codes, to look at their limitations, and to highlight problems for which more sophisticated software is still needed.

This report is organised as follows. We briefly review Arnoldi iteration in Section 2 then in Section 3 we look at the software packages ARNCHEB, ARPACK, and EB13. We outline the algorithms used and discuss the main features of each of the codes. We examine the main differences between the codes in Section 4. In Section 5 we discuss the design of our experiments to compare the performance of the software, we explain how we verify the computed results, and present numerical results for a set of test problems. Based on our findings, in Section 6 we propose possible future developments in Arnoldi based software. Details of the availability of the software packages are given in Section 7.

We end this section by introducing notation which we use throughout this report.

- $\mathbf{A}$ is a large sparse real nonsymmetric matrix of order n.

- The eigenvalues of $\mathbf{A}$ are denoted by $\lambda_1, \lambda_2, \ldots, \lambda_n$, with associated eigenvectors $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n$. The eigenvalues are assumed to be ordered according to which are being sought. For example, if the eigenvalues of largest absolute value are required, the eigenvalues are ordered in decreasing order of their absolute values. Subscripts are dropped when doing so causes no confusion.

- r denotes the number of sought-after eigenvalues of $\mathbf{A}$.
- m denotes the dimension of the subspace used in the Arnoldi algorithm.
- $\mathbf{X}_m$ denotes the matrix representation of this subspace.
- $(\mathbf{s}, \theta)$ denotes an eigenpair of the projection matrix $\mathbf{H}_m$ of order m of $\mathbf{A}$ onto the column space of $\mathbf{X}_m$.
- The approximate eigenpairs for $\mathbf{A}$ are called *Ritz* pairs if $\mathbf{A}\mathbf{y} \approx \mathbf{y}\theta$, where $\mathbf{y} = \mathbf{X}_m\mathbf{s}$.
- $\mathbf{T}_m$ denotes the quasi-triangular Schur matrix associated with the projection of $\mathbf{A}$.
- $\mathbf{X}_m^T\mathbf{A}\mathbf{X}_m \approx \mathbf{T}_m$ is an approximate real partial Schur form if $\mathbf{X}_m^T\mathbf{X}_m \approx \mathbf{I}_m$.
- u denotes the relative machine precision (that is, the smallest machine number such that $1 + u > 1$).
- ϵ denotes the user-prescribed convergence tolerance.

In this study we are concerned with the case $r < m \ll n$.

2 Arnoldi iteration

Arnoldi's method (1951) is an orthogonal projection method for approximating a subset of the eigensystem of a general square matrix. Starting with a vector $\mathbf{x}_1$, the method builds, step by step, an orthogonal basis for the *Krylov* space of $\mathbf{A}$:

$$\mathcal{K}_m(\mathbf{A}, \mathbf{x}_1) \equiv \mathrm{Span}\{\mathbf{x}_1, \mathbf{A}\mathbf{x}_1, \ldots, \mathbf{A}^{m-1}\mathbf{x}_1\}.$$

It is a generalization of the power method in that a sequence of iterates are used to approximate eigenvalues of $\mathbf{A}$. The original algorithm was designed to reduce a dense matrix to upper Hessenberg form. However, because the method requires knowledge only of $\mathbf{A}$ through matrix-vector products, its value as a technique for approximating a few eigenvalues of a large sparse matrix was soon realised. When the matrix $\mathbf{A}$ is symmetric, the procedure reduces to the method of Lanczos (1950).

Over a decade of research was devoted to understanding and overcoming the numerical difficulties of the method for the case when $\mathbf{A}$ is symmetric (see, for example, Parlett, 1980, and Grimes, Lewis and Simon, 1994). Development of the Arnoldi method for nonsymmetric matrices lagged behind because of the inordinate computational and storage requirements if a large number of steps are required for convergence. Not only is more storage needed when $\mathbf{A}$ is nonsymmetric, but, in general, more steps are required to compute the desired eigenvalue approximations.

An explicitly restarted Arnoldi iteration (ERA-iteration) was introduced by Saad (1980) in an attempt to overcome these difficulties. Besides the storage considerations, restarted Arnoldi methods are motivated by attempting to replace the

Table 2.1: Basic restarted Arnoldi iteration

- *Start:* Choose an initial normalised vector $\mathbf{x}_1$.
- *Iteration:* Until convergence **do**
  1. Compute the Arnoldi reduction $\mathbf{AX}_m = \mathbf{X}_m\mathbf{H}_m + \mathbf{f}_m\mathbf{e}_m^T$ of length m with starting vector $\mathbf{X}_m\mathbf{e}_1 \equiv \mathbf{x}_1$.
  2. Using the length m Arnoldi factorisation, select a new starting vector $\mathbf{x}_1$.

starting vector with one that is an element of the nvariant subspace associated with the r eigenvalues of interest. If this is accomplished, then it may be shown that, in exact arithmetic, $\mathbf{f}_r = 0$.

The basic restarted Arnoldi method is summarised in Table 2.1. $\mathbf{H}_m$ is an $m \times m$ upper Hessenberg matrix, $\mathbf{X}_m^T\mathbf{X}_m = \mathbf{I}_m$, and the residual vector $\mathbf{f}_m$ is orthogonal to the columns of $\mathbf{X}_m$. The matrix $\mathbf{H}_m = \mathbf{X}_m^T\mathbf{AX}_m$ is the orthogonal projection of $\mathbf{A}$ onto the column space of $\mathbf{X}_m \equiv \mathcal{K}_m(\mathbf{A}, \mathbf{x}_1)$.

The idea of restarting is based on similar approaches used for the Lanczos process by Paige (1971), Cullum and Donath (1974), and Golub and Underwood (1977). The first example of a restarted iteration is attributed to Karush (1951). A relatively recent variant was developed by Sorensen (1992) as a more efficient and numerically stable way to implement restarting. One of the benefits of this implicitly restarted Arnoldi iteration (IRA-iteration) is that it avoids the need to restart the reduction from scratch at each iteration.

3 Arnoldi iteration software

In this section we briefly review the software packages ARNCHEB, ARPACK, and EB13, which implement restarted Arnoldi iterations.

3.1 ARNCHEB

The ARNCHEB package of Braconnier (1993) provides subroutine ARNOL that implements an explicitly restarted Arnoldi method. The code is based on the algorithms of Saad (1980, 1984) and may be used to compute either the eigenvalues of largest or smallest real parts, or those of largest imaginary part.

In ARNCHEB, the computation of the restart vector is a two-step process. First, a linear combination of the r Ritz vectors associated with the r Ritz values of interest is formed. Then, a fixed-degree Chebychev polynomial $p_l(\mathbf{A})$ on an ellipse containing the unwanted Ritz values is applied to the linear combination. The evaluation of $p_l(\mathbf{A})\mathbf{x}$ is carried out by using the three-term recurrence relation for Chebychev polynomials. The polynomial is fixed in the sense that the degree is chosen by the user and is not varied from iteration to iteration.

An iterated classical Gram–Schmidt algorithm is used to maintain orthogonality of the Arnoldi basis vectors. ARNCHEB requires the user to supply a subroutine ATQ to perform matrix products $\mathbf{AX}_m$. The subroutine PRMAV does not include the matrix

$\mathbf{A}$ in its argument list so $\mathbf{A}$ need not be held explicitly—only the action of $\mathbf{A}$ on vectors is needed.

The package ARNCHEB offers the user the option of using a variant of Wielandt deflation (see Wilkinson, 1965, and Saad, 1992, for further details). Let $\mathbf{A}_0 \equiv \mathbf{A}$ and denote by $\mathbf{\Theta}_j$ the diagonal matrix of converged Ritz values. As the individual Ritz values converge, the Arnoldi iteration builds factorisations with the rank j modification

$$\mathbf{A}_j = \mathbf{A} - \mathbf{U}_j \mathbf{\Theta}_j \mathbf{U}_j^T, \tag{3.1}$$

of $\mathbf{A}$ where $\mathbf{A}_{i-1}\mathbf{U}_i\mathbf{e}_i \approx \mathbf{U}_i\mathbf{e}_i\theta_i$, for $i = 1, \ldots, j$ and θ_i is the i-th diagonal element of $\mathbf{\Theta}_j$. The deflation's goal is that the converged Ritz values are no longer extremal for $\mathbf{A}_j$ and the remaining eigenvalues of $\mathbf{A}$ are computed.

In order to use real arithmetic, ARNCHEB chooses $\mathbf{\Theta}_j$ to be a quasi-diagonal matrix. The quasi-diagonal matrix contains the real eigenvalues on the diagonal and the real and imaginary portions of the complex conjugate pairs on diagonal blocks of order two. For the blocks of order two on the diagonal of $\mathbf{\Theta}_j$ the corresponding complex eigenvector is stored in two consecutive columns of $\mathbf{U}_j$, the first holding the real part, and the second the imaginary part.

Unfortunately, although the column space of $\mathbf{U}_j$ is approximately invariant for $\mathbf{A}$ and $\mathbf{A}_j$, the eigenvectors are not the same. The extra computation involves computing the projection of $\mathbf{A}$ onto the column space of $\mathbf{U}_j$. Although not complicated, the computation must be carried out by the user. Moreover, there is no documentation to guide the user on how this could be done nor are the converged Ritz values saved.[1] We further discuss this in Section 4.2.

3.2 EB13

The Harwell Subroutine Library code EB13 (Scott, 1995) also implements an explicitly restarted Arnoldi method. It allows the user to compute the eigenvalues of $\mathbf{A}$ that are right-most, of largest modulus, or of largest imaginary parts. By working with $-\mathbf{A}$ in place of $\mathbf{A}$, the code may also be used to compute the left-most eigenvalues.

EB13 incrementally computes a partial Schur form for $\mathbf{A}$, locking approximate Schur vectors corresponding to Ritz values that converge. At each iteration, the best approximating Ritz vector of the unlocked portion of the Arnoldi reduction is used to restart. For example, if the eigenvalues of largest magnitude are desired, then the best approximating Ritz vector is the one associated with the Ritz value largest in magnitude.

A Chebychev polynomial $p_l(\mathbf{A})$ on an ellipse containing the unwanted Ritz values is applied to the restart vector in an attempt to accelerate convergence. The code adaptively selects the degree l of the Chebychev polynomial on each iteration (although the user may override this value).

An iterated classical Gram-Schmidt algorithm is used to orthogonalise the Arnoldi basis vectors. The code EB13 uses reverse communication. Each time a set of vectors is required to be multiplied by $\mathbf{A}$, control is returned to the user. This allows

[1]In fact, the author of ARNCHEB was not aware that this was possible.

Table 3.2: The Arnoldi iteration used by EB13.

- Build a length m Arnoldi reduction. Set $j = 0$ define $\mathbf{Q}_0 \equiv 0$.
- *Iteration:*
  1. Compute the ordered Schur decomposition $\mathbf{H}_{m-j}\mathbf{Z}_{m-j} = \mathbf{Z}_{m-j}\mathbf{T}_{m-j}$.
  2. Check the Ritz vector $\mathbf{X}_{m-j}\mathbf{Z}_{m-j}\mathbf{e}_1$ for convergence. If it satisfies the convergence criterion, increment j and set $\mathbf{Q}_j = \begin{bmatrix} \mathbf{Q}_{j-1} & \mathbf{X}_{m-j}\mathbf{Z}_{m-j}\mathbf{e}_1 \end{bmatrix}$. If $j = r$, exit the *Iteration.*
  3. Compute the Arnoldi reduction $\mathbf{A}\begin{bmatrix} \mathbf{Q}_j & \mathbf{X}_{m-j} \end{bmatrix} = \begin{bmatrix} \mathbf{Q}_j & \mathbf{X}_{m-j} \end{bmatrix}\begin{bmatrix} \mathbf{T}_j & \mathbf{M}_j \\ & \mathbf{H}_{m-j} \end{bmatrix} + \mathbf{f}_m\mathbf{e}_m^T$ of length m with starting vector $p_l(\mathbf{A})\mathbf{X}_{m-j}\mathbf{e}_1$ orthogonal to the Range$(\mathbf{Q}_j)$.

full advantage to be taken of the sparsity and structure of $\mathbf{A}$ and of vectorisation or parallelism. It also gives the user greater freedom in cases where the matrix $\mathbf{A}$ is not held explicitly and only the product of $\mathbf{A}$ with vectors is known. Reverse communication is discussed further in Section 4.3.

Unlike any of the other Arnoldi codes tested, EB13 optionally computes a block Arnoldi reduction. This option is designed for problems where the wanted eigenvalues are multiple or closely clustered. Another option is available to perform Chebychev polynomial preconditioning on $\mathbf{A}$.

Finally, once the required eigenvalues of $\mathbf{A}$ are computed, subroutine EB13B may be used to compute the corresponding (normalised) eigenvectors and, optionally, the scaled eigenvector residuals $\|\mathbf{Ay} - \theta\mathbf{y}\|_2/\|\mathbf{Ay}\|_2$.

Table 3.2 summarises the default procedure used by EB13.

3.3 ARPACK

The ARPACK software package (Lehoucq et al., 1995) provides subroutine DNAUPD that implements an implicitly restarted Arnoldi method. The scheme is called *implicit* because the starting vector is updated with an implicitly shifted QR algorithm on the Hessenberg matrix $\mathbf{H}_m$.

The method is motivated by the following observation. Let $\mathbf{AX}_m = \mathbf{X}_m\mathbf{H}_m + \mathbf{f}_m\mathbf{e}_m^T$ be a length m Arnoldi factorisation. Suppose that ψ is a polynomial of degree $m - r$. A simple but tedious derivation shows that

$$\psi(\mathbf{A})\mathbf{X}_r \;=\; \mathbf{X}_m\psi(\mathbf{H}_m)\begin{bmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \cdots & \mathbf{e}_r \end{bmatrix}. \tag{3.2}$$

Compute the QR factorisation $\psi(\mathbf{H}_m)\begin{bmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \cdots & \mathbf{e}_r \end{bmatrix} = \mathbf{Q}_r\mathbf{R}_r$. Equation (3.2) may then be rewritten as $\psi(\mathbf{A})\mathbf{X}_r = \mathbf{X}_m\mathbf{Q}_r\mathbf{R}_r$. The column space of $\mathbf{X}_m\mathbf{Q}_r$ is an orthogonal basis for $\psi(\mathbf{A})\mathbf{X}_r$.

Restarting the iteration involves post-multiplying the length m Arnoldi factorisation with $\mathbf{Q}_r$ and thus obtaining a length r factorisation. Thus, an IRA-iteration may be viewed as a truncated QR algorithm (see Lehoucq, 1995, and Sorensen, 1995, for further details).

DNAUPD computes the eigenvalues of $\mathbf{A}$ that are right-most, left-most, of largest or smallest modulus, or of largest or smallest imaginary parts. It uses approximate Schur vectors to restart. An iterated classical Gram-Schmidt algorithm is used to orthogonalise the Arnoldi basis vectors. The standard deflation rules used by the QR algorithm are employed on $\mathbf{H}_m$. Thus, if a subdiagonal element of $\mathbf{H}_m$ becomes small enough, it is set to zero, and the corresponding columns of $\mathbf{X}_m$ are locked. As in EB13, reverse communication is used when computing matrix-vector products with $\mathbf{A}$. An option allows the user to define a polynomial preconditioner on $\mathbf{A}$ through its roots via the implicitly shifted QR iteration on $\mathbf{H}_m$ performed during each iteration. Spectral transformations are also available, as well as the ability to solve the generalised eigenvalue problem, $\mathbf{Az} = \lambda\mathbf{Bz}$, when $\mathbf{B}$ is a symmetric positive semi-definite matrix.

Finally, analogous to the approach of EB13, once the desired Ritz values have converged, subroutine DNEUPD optionally computes associated approximate Ritz or Schur vectors. Moreover, if a spectral transformation is employed, DNEUPD maps the computed Ritz values to those of the original system.

4 Software comparison

ARNCHEB, ARPACK, and EB13 are all Arnoldi based codes written in FORTRAN 77, but it is clear from the above descriptions that the algorithms they use differ from one another in a number of important ways. Furthermore, implementation details differ. The aim of this section is to examine some of the main differences. This should help us to understand the difference in performance of the codes when used to solve a set of test problems (see Section 5).

4.1 Restart mechanism

A principal difference between all three codes is the manner in which each algorithm restarts the iteration. As discussed in Section 2, restarting mechanisms attempt to alleviate storage and replace the starting vector with one that is "rich" in the direction of the desired invariant subspace.

ARNCHEB and EB13 explicitly restart with Ritz vectors. However, EB13 carefully deflates the converged Ritz vectors, using them to incrementally build a partial Schur decomposition. Moreover, ARNCHEB uses a linear combination of r Ritz vectors while EB13 uses the best approximating Ritz vector when restarting. Mathematically, r linearly independent eigenvectors may not exist and thus ARNCHEB may not be able to compute the r eigenvalues of interest. In practical computation, the r linearly independent eigenvectors may form a poorly conditioned basis for the wanted eigenspace.

It may be shown that ARPACK's restarting procedure implicitly replaces the starting vector by a linear combination of r approximate Schur vectors associated with the r wanted Ritz values. Thus the Arnoldi factorisation is restarted by using numerically orthogonal matrices of order m and an explicit restart is avoided.

4.2 Deflation

Each of the codes attempts to use deflation in order to reduce the size of the active Arnoldi factorisation and for robustness. This section is adapted from similar ideas discussed in Lehoucq and Sorensen, 1995. For ease of discussion, we suppose that converged quantities are known exactly.

We first examine the deflation strategy used by EB13. ¿From line 3 of the algorithm in Table 3.2, the length m Arnoldi factorisation is equivalent to

$$\hat{\mathbf{A}}_j \mathbf{X}_{m-j} = \mathbf{X}_{m-j}\mathbf{H}_{m-j} + \mathbf{f}_m \mathbf{e}_{m-j}^T,$$

where $\hat{\mathbf{A}}_j \equiv (\mathbf{I} - \mathbf{Q}_j\mathbf{Q}_j^T)\mathbf{A}(\mathbf{I} - \mathbf{Q}_j\mathbf{Q}_j^T)$. If we assume that $\mathbf{A}\mathbf{Q}_j = \mathbf{Q}_j\mathbf{T}_j$, then it is easily shown that any vector in the range of $\mathbf{Q}_j$ is annihilated by $\hat{\mathbf{A}}_j$ and that $\mathcal{K}_{m-j}(\hat{\mathbf{A}}_j, \mathbf{X}_{m-j}\mathbf{e}_1)$ is orthogonal to the range of $\mathbf{Q}_j$. The locking process used by EB13 thus allows convergence to the remaining eigenvalues of $\mathbf{A}$. Of course, EB13 never needs to explicitly apply $\hat{\mathbf{A}}_j$. During the orthogonalisation phase of the Arnoldi iteration, the columns are orthogonalised against the $\mathbf{X}_{m-j}$ locked vectors.

Section 3.3 explains the manner in which ARPACK uses deflation. It sets to zero any subdiagonal elements of $\mathbf{H}_m$ that are negligible according to the criteria used by the standard QR algorithm. If the i-th subdiagonal element of $\mathbf{H}_m$ is set to zero, then the first i columns of $\mathbf{X}_m$ are locked. We remark, that unlike the standard QR algorithm, ARPACK rarely encounters the opportunity to deflate. Moreover, there is no guarantee that all the associated vectors locked are desired ones. Upon approximation of the r desired eigenvalues, a postprocessing step is performed so that the first r columns of $\mathbf{X}_m$ are a Schur basis for these r eigenvalues.

Section 3.1 briefly introduced the Wielandt deflation used by ARNCHEB. We assume complex arithmetic in order to avoid being overwhelmed by technical details. An inductive proof shows that

$$\mathbf{A}\mathbf{U}_j = \mathbf{U}_j\mathbf{C}_j \tag{4.3}$$

where $\mathbf{C}_j$ is the upper triangular portion of the matrix $\Theta_j\mathbf{U}_j^H\mathbf{U}_j$. ¿From equation (4.3), a direct computation reveals that

$$\mathbf{A}_j\mathbf{U}_j = \mathbf{U}_j(\mathbf{C}_j - \Theta_j\mathbf{U}_j^H\mathbf{U}_j),$$

and thus $\mathbf{C}_j - \Theta_j\mathbf{U}_j^H\mathbf{U}_j$ is a strictly lower triangular matrix. The converged Ritz values are now zero eigenvalues of $\mathbf{A}_j$. However, $\mathcal{K}_i(\mathbf{A}_j, \mathbf{X}_m\mathbf{e}_1)$ is not orthogonal to the column space of $\mathbf{U}_j$ unless two further conditions are met. The first requires $\mathbf{X}_m\mathbf{e}_1$ to be orthogonal to the column space of $\mathbf{U}_j$. Assuming the previous condition, $\mathbf{A}\mathbf{X}_m\mathbf{e}_1$ must also be orthogonal to the column space of $\mathbf{U}_j$. For example, this would occur if $\mathbf{A}$ were a symmetric matrix. In general, the Wielandt deflation adopted by ARNCHEB does not prevent locked vectors from emerging in subsequent factorisations computed with $\mathbf{A}_j$. Moreover, careful inspection of ARNCHEB reveals that the starting vector selected is not orthogonal to the column space of $\mathbf{X}_m$.

4.3 Matrix products Ax

A large part of the total cost of computing eigenvalues of a sparse nonsymmetric matrix using Arnoldi's method is the cost of performing matrix-vector products $\mathbf{Ax}$.

Some timings which illustrate this are given in Scott (1995). It is therefore very important for the overall performance of the software that matrix-vector products are computed as efficiently as possible. If the software is to be general purpose, it is also important that the matrix $\mathbf{A}$ is not required to be held in a prescribed fixed format. In many practical situations, the matrix is not known explicitly, but only the action of $\mathbf{A}$ on vectors is available, so the software needs to be able to cope with this. Each of the three codes in our study recognises the need to allow the user to exploit the sparsity and structure of the matrix but one of the major implementation differences between ARNCHEB and the other two codes is the way in which matrix-vector products are carried out.

ARNCHEB requires the user to supply a subroutine to perform matrix-vector products. The matrix $\mathbf{A}$ is not an argument to the subroutine so the user is not required to hold the matrix explicitly. Nevertheless, for some problems it can be inconvenient to pass the matrix into this subroutine. For example, since ARPACK is written in FORTRAN 77, the number of subroutine arguments is fixed. Consequently, if a user needs additional descriptors to perform matrix products, they must be passed using a COMMON block.

The reverse communication approach adopted by ARPACK and EB13 provides flexibility and gives the user a greater degree of control. By avoiding passing the matrix through a COMMON block, the user is able to take full advantage of parallelism and/or vectorisation. Reverse communication also allows the user to incorporate different preconditioning techniques in a very straightforward way. For example, the user may wish to use a shift-and-invert transformation, in which $(\mathbf{A} - \sigma\mathbf{I})^{-1}$ is used in place of $\mathbf{A}$. The eigenvalues close to the shift σ will tend to converge most rapidly since under the transformation they become dominant. In this case, linear systems of the form $(\mathbf{A} - \sigma\mathbf{I})\mathbf{w} = \mathbf{x}$ are solved in place of the matrix products $\mathbf{w} = \mathbf{A}\mathbf{x}$. If a direct method of solution is used, the LU factorisation of $(\mathbf{A} - \sigma\mathbf{I})$ need only be performed once. However, since reverse communication allows progress to be monitored, the user may choose to update σ as the computation progresses, and a new factorisation will be required for each shift.

4.4 Exploiting BLAS

Apart from the matrix-vector products with $\mathbf{A}$, Arnoldi's method only requires dense linear algebra operations to be performed on matrices of order m. One way of achieving an efficient implementation and assisting with robustness, portability, readability, and maintance of the software is through the use of BLAS (Basic Linear Algebra Subprograms) kernels (Lawson, Hanson, Kincaid and Krogh, 1979, Dongarra, DuCroz, Hammarling and Hanson, 1988, and Dongarra, DuCroz, Duff and Hammarling, 1990). Highly efficient machine-specific implementations of the BLAS are available for many modern high-performance computers. By exploiting the BLAS, software can achieve high performance and be portable.

Level 1 BLAS perform basic vector operations, such as $\mathbf{y} \leftarrow \alpha\mathbf{x}+\mathbf{y}$. They are not able to achieve high efficiency on most modern supercomputers but they do assist with the clarity and portability. Level 2 BLAS perform matrix-vector products, such as $\mathbf{y} \leftarrow \alpha\mathbf{A}\mathbf{x} + \beta\mathbf{y}$. For machines having a memory hierarchy, the Level 2 BLAS do not have a ratio of floating-point operations to data movement that is high enough to make efficient use of data that reside in cache or local memory. For these architectures, it is often preferable to partition matrices into blocks and to perform the computation using matrix-matrix operations on the blocks. The Level 3 BLAS are targeted at the matrix-matrix operations required for these purposes.

Each of the codes in our study uses BLAS routines. ARNCHEB and EB13 employ mainly Level 1 and Level 2 routines and, in addition, they use EISPACK routines (Smith, Boyle, Garbow, Ikebe, Klema and Moler, 1976). EISPACK has for many years provided high-quality portable software for eigenvalue problems; but on modern high-performance computers EISPACK routines often achieve only a small fraction of the peak performance of the machines. LAPACK (Anderson, Bai, Bischof, Demmel, Dongarra, Croz, Greenbaum, Hammarling, McKenney, Ostrouchov and Sorensen, 1992) was designed to supersede EISPACK. The authors of LAPACK developed new routines and restructured the EISPACK software with the aim of achieving much greater efficiency. This was accompolished by writing routines using the BLAS as building blocks. ARPACK makes extensive use of both the BLAS and LAPACK routines and we anticipate that this will be reflected in its efficiency and robustness.

4.5 The stopping criteria

Each of the codes uses different stopping criteria, which adds to the difficulties associated with trying to compare their performance (see Section 5). Helpful discussions of stopping criteria for iterative eigensolvers are given by Bennani and Braconnier (1994) and Scott (1995). Throughout this section, ϵ denotes a user-defined tolerance.

EB13 follows Stewart (1978) and bases its stopping criterion on demanding that

$$\mathbf{AX}_r \approx \mathbf{X}_r\mathbf{T}_r.$$

The difficulty of choosing appropriate stopping criteria was recognised during the development of EB13 (Scott, 1995) and as a result, EB13 offers a choice of stopping criteria. The user can require that the jth column of $\mathbf{X}$ satisfies $||(\mathbf{AX}_m - \mathbf{X}_m\mathbf{T}_m)_j||_2 < ||\mathbf{A}||\epsilon$, or $< ||(\mathbf{AX}_m)_j||_2\epsilon$, or, finally, $< \epsilon$. The advantage of using the norm of $\mathbf{A}$ is that the stopping criterion is based on the backward error. This is discussed by Chatelin and Fraysée (1993). The idea of backward error is to measure the shortest distance between the original problem with computed solution $\mathbf{z}$ and a perturbed problem with exact solution $\mathbf{z}$. The normwise backward error associated with $(\mathbf{T}_m, \mathbf{X}_m)$ is defined by

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

In the 2-norm, it can be shown that $\eta = ||\mathbf{AX}_m - \mathbf{X}_m\mathbf{T}_m||_2/||\mathbf{A}||_2$.

A disadvantage of using the backward error is that it requires $||\mathbf{A}||$ to be known. If the user wants to use the stopping criteria involving $||\mathbf{A}||$, EB13 asks for $||\mathbf{A}||$ (or an estimate of $||\mathbf{A}||$) to be provided. If the user is unable to do this, the code will compute the Frobenius norm of $\mathbf{A}$ but at the cost of n matrix-vector products.

To save work, the residual $||(\mathbf{AX}_m - \mathbf{X}_m\mathbf{T}_m)_j||_2$ is computed only if all the basis vectors $\mathbf{X}_i$ with $0 < i < j$ have already been accepted. EB13 monitors the residuals for unacceptably slow convergence and, if necessary, terminates the computation with a warning that the requested accuracy was not achieved. In this event, the user is advised on how to modify the input parameters to try and obtain the requested accuracy and facilities are included for restarting the computation from the point at which the warning was issued.

The most compelling reason for possibly not wishing to use a stopping criterion that involves the norm of **A** is that it can lead to accepting Ritz values that have no digits of accuracy. 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. Since high precision in the computed eigenvalues may not be necessary, the user may be tempted to set the convergence tolerance ϵ to be, for instance, 10^{-4}. But if the norm of **A** is of order 10^5, the stopping criterion may lead to a computed θ being accepted as converged when it actually has no accuracy. Thus, wrong conclusions concerning the stability of the system may be drawn. Clearly, if the norm of **A** is to be used, the user should take its size into account when selecting the convergence tolerance.

For Arnoldi's method, an inexpensive estimate of the norm of the eigenvector residual is available. Let $\mathbf{AX}_m = \mathbf{X}_m\mathbf{H}_m + \mathbf{f}_m\mathbf{e}_m^T$ be an Arnoldi factorisation of length m. If $\mathbf{s}$ is an eigenvector of $\mathbf{H}_m$ and $\mathbf{y} = \mathbf{X}_m\mathbf{s}$, it follows that

$$||\mathbf{Ay} - \mathbf{y}\theta|| = ||\mathbf{AX}_m\mathbf{s} - \mathbf{X}_m\mathbf{H}_m\mathbf{s}|| = ||\mathbf{f}_m||\ |\mathbf{e}_m^T\mathbf{s}|.$$

The benefit of using the *Ritz estimate* $||\mathbf{f}_m||\ |\mathbf{e}_m^T\mathbf{s}|$ is that it avoids explicit formation of the direct residual $||\mathbf{AX}_m\mathbf{s} - \mathbf{X}_m\mathbf{s}\theta||$. ARPACK bases its stopping criterion on the Ritz estimate. Moreover, since only the last component of **s** is needed, ARPACK does not compute the full eigenvectors of $\mathbf{H}_m$ at each iteration. The computation is terminated on the first iteration that r Ritz values all satisfy $||\mathbf{f}_m||\ |\mathbf{e}_m^T\mathbf{s}| < |\theta|\epsilon$.

Recent work by Chatelin (1993) and Bennani and Braconnier (1994) suggests that when **A** is highly non-normal, there can be a significant difference between the Ritz estimate and the eigenvector residual. Because of this potential difference, ARNCHEB computes both the scaled Ritz estimate and the direct backward error given by

$$\frac{||\mathbf{f}_m||\ |\mathbf{e}_m^T\mathbf{s}|}{||\mathbf{A}||_F||\mathbf{y}||_2} \quad \text{and} \quad \frac{||\mathbf{Ay} - \lambda\mathbf{y}||_2}{||\mathbf{A}||_F||\mathbf{y}||_2},$$

respectively, where $||\mathbf{A}||_F$ denotes the Frobenius norm of **A**. The current version of the code tests the direct backward errors for convergence. ARNCHEB does not offer the user the option of supplying $||\mathbf{A}||_F$ but computes $||\mathbf{A}||_F$ with n matrix-vector products. Our numerical results show that this can make ARNCHEB uncompetitive with the other codes (see Appendix).

4.6 Storage requirements

When the order n of the matrix **A** is large, the amount of storage needed can be an important consideration when choosing software. In Table 4.3 we compare the storage requirements of the codes in our study. We observe that, for a given subspace dimension m, ARPACK uses the least amount of storage. We also see that, for a block size $n_b = 1$, EB13 needs three arrays of length nm. There are two reasons why

EB13 demands an extra array. First, to use the three-term recurrence relation for Chebychev polynomials to compute $p_l(\mathbf{A})\mathbf{X}$, three arrays of length nm are needed. Second, as already discussed, EB13 uses reverse communication. To try to ensure against the user's overwriting the latest approximation $\mathbf{X}_m$ to the Schur vectors, the user forms matrix-vector products by using two arrays $\mathbf{U}$ and $\mathbf{W}$ of dimension nm and then, within the code, copying into the appropriate part of the third array $\mathbf{X}_m$ is performed. Thus, even if Chebychev acceleration is not employed, EB13 demands three arrays of length nm.

Table 4.3: Storage requirements (n_b denotes the block size for EB13, and † denotes that ARNCHEB is used with deflation)

Code	Storage
ARNCHEB	$2n \times (m+5) + 2m^2 + O(m)$
ARNCHEB†	$3n \times (m+5) + 2m^2 + O(m)$
EB13	$3n \times m \times n_b + 2m^2 + O(m)$
ARPACK	$n \times (m+4) + 3m^2 + O(m)$

4.7 User interface

An important feature of any code written for general use is that it should be accompanied by straightforward but comprehensive documentation which allows the code to be used with a minimum of effort. The documentation should also assist the user in the event of the computation failing for his or her problem. Our numerical experiments have provided us with a feel for how easy the software is to use and in this section we comment briefly on our experiences.

- EB13 and ARPACK came with a well-commented parameter list which allowed us to use them without any difficulties. ARNCHEB does not document all input and output parameters fully, and the code itself does not include comments to explain each of the parameters.
- A particularly helpful feature of the documentation provided with EB13 was that it included a simple sample program. This would be of particular value to users who are unfamiliar with using reverse communication.
- The codes all provide sample programs that illustrate their use. In particular, ARPACK has an extensive set of programs illustrating the use of reverse communication and all its options.
- EB13 and ARPACK have error flags and check the parameters supplied by the user for errors. If an error is detected, EB13 optionally prints a message indicating what the error is. ARPACK sets a flag and provides documentation for

interpreting the flag. ARNCHEB has no error flag, performs no error checking, and offers no assistance in the event of an error.

- The codes all have monitoring printing; that is, at each iteration they print values of, for example, the computed eigenvalues and the corresponding residuals. This information allows the user to follow the convergence. It is particularly useful for the reverse communication codes EB13 and ARPACK because, if the convergence is not proceeding satisfactorily, the user is able to intervene. For EB13 and ARPACK the monitoring printing is optional.
- Our experience suggests that ARNCHEB was not comprehensively tested. The code was found to contain bugs.[2]
- The FORTRAN programming within ARNCHEB could be considerably improved. The code uses nonstandard FORTRAN 77 (such as REAL*8 declarations), which caused some of the compilers we used for testing to return error messages. When the code was checked with a FORTRAN code analyzer,[3] a large number of errors messages were returned. The analyzer passed the other two codes as conforming to the FORTRAN 77 standard and they were found to be portable when tested on a range of computers (including SUN workstations, a Cray Y-MP, and IBM RS/6000's).

5 Numerical experiments

In this section and the Appendix we present the results of using the software discussed in this report to compute a small number of eigenvalues and eigenvectors of a set of test problems.

5.1 The test matrices

The test problems all arise from real scientific and industrial applications. A brief description of the test problems is given in Table 5.4. The problems are drawn from the Harwell-Boeing collection of sparse matrices (Duff, Grimes and Lewis, 1992) and the recent collection of large eigenvalue problems of Bai, Barrett, Day and Dongarra (1995). Further details of the problems may be found in the above two references. We have attempted to select a range of problems with different properties and applications. We employ the same set of test problems in our evaluation of subspace iteration software (Lehoucq and Scott, 1996*b*). This study has again highlighted the need to have a set of practical problems available as a testbed for eigenvalue algorithms and we would welcome further problems which could also be used to test the software.

5.2 Verification

It is important when testing software that an attempt is made to check the correctness of the computed results. For example, an important consideration is whether

[2]The author of the code was contacted with our findings.

[3]pfort, ISTLA - Toolpack Static Analyser, Version 1.2

Identifier	Order	Number of entries	Description/discipline
PORES2	1224	9613	Oil reservoir simulation.
PORES3	532	3474	Oil reservoir simulation.
GRE1107	1107	5664	Simulation studies in computer systems.
HOR131	434	4710	Flow network problem.
IMPCOLC	137	411	Ethylene plant model.
IMPCOLD	425	1339	Nitric acid plant model.
NNC666	666	4044	Nuclear reactor core modelling.
NNC1374	1374	8606	Nuclear reactor core modelling.
WEST0156	156	371	Chemical engineering plant model.
WEST0167	167	507	Chemical engineering plant model.
WEST2021	2021	7353	Chemical engineering plant model.
*CK400	400	2860	Not available.
*CK656	656	3884	Not available.
*RWK5151	5151	20199	Markov chain modelling: random walk.
*CDDE			2-D convection diffusion problem.
*TOLOSA			Stability of aircraft in flight.
*BW2000	2000	7996	Chemical engineering model.
* PDE2961	2961	14585	Model PDE eigenvalue problem

Table 5.4: The matrices used for performance testing (* indicates matrix from the collection of Bai, Barratt, Day and Dongarra, 1995).

any of the sought-after eigenvalues have been missed. In the symmetric case, if a factorisation is performed, an inertia count can then be used to provide a check for missing eigenvalues (see Grimes et al., 1994 and Parlett, 1980 for details). There is no analogous procedure for nonsymmetric matrices.

For the purposes our study, we may determine the reliability of the codes using the exact eigenvalues. The forward error is defined to be

$$FE_{max} = \max_{1\leq i\leq r} \frac{|\lambda_i - \theta_i|}{|\lambda_i|}, \tag{5.4}$$

where λ_i and θ_i are the exact and computed eigenvalues, respectively, of $\mathbf{A}$. This tests the forward stability of the software. For the test problems for which the exact eigenvalues are not known, we compare the computed eigenvalues with those found using the QR algorithm.

We also check results by computing the r eigenvector residuals

$$\|\mathbf{A}\mathbf{y} - \theta\mathbf{y}\|_2, \tag{5.5}$$

and the real and imaginary portions of the Rayleigh Quotient errors

$$\|\mathbf{y}^T\mathbf{A}\mathbf{y} - \theta\mathbf{y}^T\mathbf{y}\|_2. \tag{5.6}$$

For ARPACK and EB13, we check the orthogonality of the computed Schur basis and quality of the Schur projection by computing

$$\|\mathbf{X}_r^T\mathbf{X}_r - \mathbf{I}_r\|_\infty \quad \text{and} \quad \|\mathbf{X}_r^T\mathbf{A}\mathbf{X}_r - \mathbf{T}_r\|_\infty, \tag{5.7}$$

respectively.

The checks (5.5)-(5.7) are designed to test the backward stability of the software.

5.3 The test environment

The numerical experiments were performed on an IBM RS/6000 3BT using double precision arithmetic, and the vendor-supplied BLAS. As we have already seen, the software in our study employ different stopping criteria. Therefore, even if we supply each code with the same convergence tolerance and the computations all terminate successfully, the eigenvalues computed by each code may differ. For the results reported in this section and in the Appendix, the codes each used a convergence tolerance that gave eigenvalues with an accuracy of at least $\sqrt{u}$ (for some of the test examples, different codes used different convergence tolerances). The convergence tolerances used were all in the range $10u$ to 10^{-4}.

In designing their software, the authors have all attempted to produce software which can be used as a black box while at the same recognising that in doing so they have had to make a number of *ad hoc* decisions and there may be problems for which the choices that have been made are either poor or completely unsuitable. To assess the usefulness of the choices, in our numerical experiments we only use the default values (or values recommended by the authors in their documentation).

Good general-purpose software should make most decisions automatically and not require the user to have a detailed understanding of the algorithm being implemented. Each of the codes in our study requires the user to choose the number r of eigenvalues required, the subspace dimension m, and the convergence tolerance ϵ. In addition, the codes require the user to decide which portion of the spectrum is to be computed.

Table 5.5: Input from the user

Code	Required Input
ARNCHEB	Matrix-vector product routine Type of ellipse Degree of Chebychev polynomial Amount of orthogonalisation Whether to perform deflation
ARPACK	Maximum number of iterations
EB13	Block size

Requiring the user to choose these parameters may appear reasonable because the user is likely to know how many eigenvalues are required and how much accuracy is wanted. However, as discussed in Section 4.5, in order to select an appropriate value for ϵ, the user generally needs some knowledge of the problem, such as the norm of **A** or the size of the sought-after eigenvalues. Furthermore, our experience with the codes has shown that selecting r to be greater than the number of eigenvalues actually required can sometimes yield more rapid convergence. This can happen if the sought-after eigenvalues are not well separated from the remaining ones and better separation is achieved by increasing r. Moreover, the efficiency of the software depends strongly on the choice of m. For small m, convergence may not be possible. On the other hand, if m is large, the amount of work per iteration and the storage requirements may be prohibitively high.

When using `ARNCHEB`, the user has a number of decisions to make. He or she must decide which of the routines provided for computing an ellipse is to be used and whether or not to use reorthogonalisation and/or deflation. Making these decisions requires an understanding of Arnoldi's method and its implementation. The ability to experiment with different options is of considerable value, and thus `EB13` also offers different ellipse routines. The difference is that `EB13` has a default routine that is used unless the user selects one of the alternatives. The use of default settings helps make `EB13` user friendly while at the same time providing flexibility.

The code `ARNCHEB` requires the degree of the Chebychev polynomial to be specified by the user but provides no advice on how to do this. We performed some preliminary experiments with the code and, on the basis of these experiments, selected a degree of $m - r$ for all our reported results. In our tests, doubling or even tripling this value generally increased the total time required for convergence.

In our tests, a limit of $4000m$ was imposed on the number of matrix-vector products allowed.

The results of our numerical experiments are lengthy so in the next two subsections we present detailed results for the TOLOSA matrix and the two-dimensional convection-diffusion problem (CDDE), and we then summarise our findings for the remaining test problems. The complete results for computing a single eigenpair are given in Appendix A and for computing several eigenpairs the results are in Appendix B.

5.4 Results for the TOLOSA Matrix

The TOLOSA matrix arises from the stability analysis of a model of an airplane in flight. Its eigenvalues lie on a parabola in the left-half plane that opens to the left. The eigenvalues of interest are the eigenvalues of largest imaginary part, which are also those of largest modulus. The matrix is non-normal, and its departure from normality increases with the order of the matrix.

We employed each of the codes to compute the complex conjugate pair of eigenvalues of largest imaginary part and the corresponding eigenvectors of the TOLOSA matrix with orders up to 2000. Since the The eigenvalues of interest are the eigenvalues of largest imaginary part are also those of largest modulus, we ran the codes `ARPACK` and `EB13` with `WHICH = LM` (largest modulus) and tt WHICH = LI (largest imaginary part). Our findings for $n = 1000$ and $n = 2000$ are summarised in Tables 5.6 and 5.7.

We see that for this problem the value of m giving the best `ARPACK` results is larger than that giving the best `ARNCHEB` and `EB13` results. This suggests that the

Table 5.6: CPU times (in seconds) and matrix-vector products for the TOLOSA matrix of order 1000

		Subspace dimension m		
Algorithm	WHICH	8	16	32
ARNCHEB	LI	1.0/1867	2.4/2583	7.3/4294
ARPACK	LM	12/3992	4.5/1080	3.2/482
ARPACK	LI	15/5120	3.1/744	3.0/452
EB13	LM	0.8/405	2.2/749	2.6/496
EB13	LI	5.6/5545	0.8/625	8.2/3917

Table 5.7: CPU times (in seconds) and matrix-vector products for the TOLOSA matrix of order 2000

		Subspace dimension m		
Algorithm	WHICH	8	16	32
ARNCHEB	LI	5.0/4201	6.3/4100	15/5484
ARPACK	LM	16/2528	3.4/422	12/1052
ARPACK	LI	108/16844	13/1612	7.1/602
EB13	LM	3.3/793	8.7/1429	8.8/892
EB13	LI	2.8/1451	6.4/2468	7.1/1657

figures given in Table 4.3 for the storage requirements of the different codes should be taken into consideration. We also see in this example that none of the codes clearly performs better than the others, and each code is sensitive to the value of m and, in the case of ARPACK and EB13, to which eigenvalues the code has been asked to compute.

5.5 Results for a Convection-Diffusion Problem

The second problem for which we present results is a two-dimensional model convection-diffusion problem

$$-\Delta \mathbf{u}(x,y) + \rho\nabla \cdot \nabla \mathbf{u}(x,y) = \lambda \mathbf{u}(x,y),$$

on the unit square $[0,1] \times [0,1]$, with zero boundary data and ρ a real number. The problem is discretised using centered finite differences. The eigenvalues and eigenvectors of the resulting matrix are known explicitly (see, for example, Bai et al., 1995). We have chosen this example because it has the following interesting properties:

- Many of the eigenvalues have multiplicity two. It may be shown that, if $|\rho| \leq \sqrt{n}$, the eigenvalues are all real and the matrix is diagonalisable.
- As the mesh size decreases, the relative separation of all the eigenvalues decreases. All the eigenvalues are contained within the interval (0, 8).
- As ρ increases, so does the non-normality of the matrix.

We computed $r = 6$ eigenpairs of largest real part for a range of values of ρ and for orders up to $n = 10,000$. The eigenvalues of largest real part are also those of largest modulus. We found that, as n was increased, EB13 and ARPACK required small a convergence tolerance to avoid missing multiple eigenvalues.

Table 5.8: CPU times (in seconds) and matrix-vector products for the 2-D Laplacian ($\rho = 0$) matrix of order 2500 († denotes that one or more of the requested eigenvalues was missed)

		Subspace dimension m	
Algorithm	WHICH	18	36
ARNCHEB	LR	†	†
ARPACK	LM	8.7/630	9.1/532
ARPACK	LR	9.2/616	9.6/557
EB13	LM	35/3707	32/2375
EB13	LR	7.2/2116	8.2/1358

In addition to the tests reported in the tables, we ran the case $\rho = 40$ with $n = 10,000$. Since $|\rho| \leq \sqrt{n}$, the exact eigenvalues are all real. All the codes experienced difficulties for this example. Using convergence tolerances of $\sqrt{u}$ and 10^3u, ARPACK ran without an error flag being set, but complex eigenvalues were returned. When

Table 5.9: CPU times (in seconds) and matrix-vector products for the 2-D Laplacian ($\rho = 0$) matrix of order 10,000. († denotes that one or more of the requested eigenvalues was missed. $*$ denotes that code did not converge within $4000m$ matrix-vector products).

		Subspace dimension m	
Algorithm	WHICH	18	36
ARNCHEB	LR	$*$	$*$
ARPACK	LM	171/2625	85/1153
ARPACK	LR	185/2827	79/1081
EB13	LM	†	342/5687
EB13	LR	81/4781	115/4263

Table 5.10: CPU times (in seconds) and matrix-vector products for the CDDE matrix with $\rho = 10$ of order 2500. († denotes that one or more of the requested eigenvalues was missed. $*$ denotes that code did not converge within $4000m$ matrix-vector products).

		Subspace dimension m	
Algorithm	WHICH	18	36
ARNCHEB	LR	†	†
ARPACK	LM	8.6/620	12/694
ARPACK	LR	8.3/602	11/613
EB13	LM	†	46/3383
EB13	LR	41/12178	$*$

Table 5.11: CPU times (in seconds) and matrix-vector products for the CDDE matrix with $\rho = 15$ of order 10000 ($*$ denotes that code did not converge within $4000m$ matrix-vector products)

		Subspace dimension m	
Algorithm	WHICH	18	36
ARNCHEB	LR	$*$	$*$
ARPACK	LM	71/1123	103/1398
ARPACK	LR	61/991	80/1095
EB13	LM	727/20004	436/7263
EB13	LR	1251/74107	$*$

the accuracy of the computed eigenvalues was tested, the forward error was found to be $0(10^{-2})$. However, the computed results were those of a nearby matrix: the four residuals (5.5)–(5.7) were suitably small. With the same convergence tolerances, EB13 returned real eigenvalues but missed the multiplicities.

In each of our tests on this problem, ARNCHEB failed to compute the required eigenvalues with the requested accuracy. The code was successful in computing the leading eigenpair. As mentioned in Section 3.1, if deflation is used, ARNCHEB does not return the Ritz values associated with the locked vectors nor the Ritz vectors for $\mathbf{A}$. As an experiment, we computed the projection of $\mathbf{A}$ onto the orthogonal span of the column space of $\mathbf{U}_j$ and computed the Ritz values. For the smaller problems, this allowed ARNCHEB to compute more of the required eigenvalues, however, for $n \geq 2,500$, ARNCHEB was not successful. Moreover, it often returned spurious Ritz values. Section 4.2 demonstrated how Wielandt deflation does not, in general, prevent the locked vectors from re-appearing in subsequent Arnoldi iterations. We are lead to conclude that the re-appearing vectors contaminated the overall iteration.

5.6 General Findings

Our general findings from the numerical experiments may be summarised as follows:

- ARPACK is generally the fastest and most dependable of the codes studied, especially for small convergence tolerances and large departures from normality.

- An attractive feature of ARPACK is that it displays monotonic consistency, that is, as the convergence tolerance decreases so does the size of the computed residuals. The stopping criteria used by EB13 which attempts to detect stagnating convergence means that EB13 does not always display this property.

- There is tendency for Arnoldi codes to miss multiplicities and to miss some of the sought-after eigenvalues when they are not well-separated from the unwanted ones. We found that EB13 and ARPACK were, in general, able to detect multiple eigenvalues but to do so it was necessary to use double precision arithmetic and to choose a very small convergence tolerance (for example, 10^4u). If a larger tolerance (such as $\sqrt{u}$) was used, multiple eigenvalues could be missed.

- The code EB13 offers many options and, as these options are fully documented, they enable the user to experiment with different choices of the parameters which define the underlying algorithm. Further work needs to be done on selecting appropriate choices and on automatically changing and modifying parameters as the computation proceeds and knowledge of the eigenvalue distribution is gained.

- For some of the examples (in particular, when several eigenvalues were requested), ARPACK uses dramatically fewer matrix-vector products than the other codes (see, for example, problems GRE1107, IMPCOLD, and WEST2021 with $r = 5$). This can result in substantial savings in computation time when application of the matrix-vector product is expensive. However, the restarting strategy used by ARPACK can be more expensive than that used by the other codes. This is typically the case when the cost of a matrix-vector product is inexpensive. The CDDE examples show this, although the ratio of the number of matrix-vector products to the total time taken decreased as the problem sized decreased.

- For problems for which several eigenvalues are required, EB13's blocking option can give worthwhile reductions in the number of iterations and in the computation time needed for convergence. For example, for problem GRE1107, which has clustered eigenvalues, using EB13 with Chebychev acceleration, the computation times for $r = 5$ are 4.9 and 2.7 seconds for the unblocked and blocked versions, respectively. However, for some problems, the blocking option gives disappointing results and further work appears to be needed. Results can be particularly poor if an inappropriate blocksize is chosen. For problem IMPCOLD, if 5 eigenvalues are wanted and the blocksize is also taken to be 5, then using 4 Arnoldi steps per iteration, EB13 requires 64 iterations to achieve convergence. However, if the blocksize is increased to 6, the number of iterations is cut to 19. This is because λ_6 is close to λ_5 and better separation is achieved by increasing the block size. We observe that the results for the block Arnoldi scheme given in Sadkane (1993) reflect a much larger subspace being constructed than was attempted in our study.

- A major disadvantage of all the codes is that they are sensitive to the choice of the input parameters. Moreover, we have found that the performance of Arnoldi based software is extremely sensitive to implementation details.

6 Future directions

We end our study of Arnoldi based software by briefly discussing how the quality of existing software might be improved in terms of efficiency, reliability, and ease of use.

Each of the codes in our study was written in FORTRAN 77. As discussed in Section 4.3, for such codes, the use of reverse communication for carrying out matrix-vector products is recommended. However, it is perhaps time to consider developing software using a more modern programming language. This could remove the requirement for a reverse communication interface. A more modern programming language could also help simplify the user interface in other ways. For example, the FORTRAN 90 programming language allows the use of optional arguments. By using optional arguments, the experienced user can be given a large amount of control and freedom to experiment, while at the same time the less familiar user can have the amount of input data reduced to a minimum.

Further research is needed into the automated selection of parameters such as the dimension m of the Krylov subspace. Here again there would be advantages in using, for example, FORTRAN 90, which allows dynamic allocation of storage. Thus, the user would only be asked to specify the number r of sought-after eigenvalues, and the software would automatically choose m and then adjust it at each iteration as the computation proceeded to try and accelerate convergence.

Improved polynomial restarting methods are required. The success of the Arnoldi codes depends upon such strategies. EB13 includes mechanisms for tracking the polynomials constructed as the iteration progresses. Such mechanisms should be incorporated into new polynomial restarting methods. ARPACK allows the user to select shifts during each iteration and should prove useful for determining alternate or new polynomials. Again, the selection of the restart polynomial needs to be automated.

The whole question of the stopping criteria is another area requiring investigation. Further research and testing need to be undertaken to improve the ways in which the software decides to terminate the computation. Strategies for detecting stagnating convergence to prevent unnecessary work from being performed are also needed. Although not always successful, EB13 does attempt to do this.

Closely related to the stopping criteria is the verification of the computed eigenvalues and eigenvectors. At present, no software is publicly available for determining whether *the* given eigenvalue problem was solved. Successful convergence implies only that *an* eigenvalue problem was solved. For the Arnoldi software, which we found can experience difficulties when attempting to locate multiple and closely clustered eigenvalues, it is particularly important to have some means of checking that one or more of the sought-after eigenvalues has not been missed. Some promising work on a possible approach has been reported on by Meerbergen, Spence and Roose (1994).

In conclusion, it is clear that although the Arnoldi software situation is much more promising than it was even 5 years ago, there remains important problems and limitations with the current software which need to be addressed. Given the results of all our experiments, we believe that an implicitly restarted Arnoldi iteration is a promising way forward and that further investigation into block algorithms is also likely to be worthwhile.

7 Availability of the Software and Test Matrices

We summarise how the interested reader may obtain the test matrices and software reviewed in this study.

- The Harwell–Boeing matrices of Duff et al. are available by anonymous ftp to `seamus.cc.rl.ac.uk` in the directory `pub/harwell_boeing`.
- ARNCHEB is available by anonymous ftp to `orion.cerfacs.fr` in the directory `pub/algo/software/Qualcomp/Arncheb/Real`.
- ARPACK is available by anonymous ftp to `ftp.caam.rice.edu` in the directory `pub/people/sorensen/ARPACK`. The file README provides directions on downloading the software.
- EB13 is included in Release 12 of the Harwell Subroutine Library, and anyone interested in using the code should contact the HSL Manager: Dr. S. J. Roberts, Harwell Subroutine Library, AEA Technology, Building 552, Harwell, Oxfordshire, OX11 0RA, England, tel. +44 (0) 1235 434714, fax +44 (0)

1235 434136, or e-mail `Scott.Roberts@aeat.co.uk`, who will provide details of price and conditions of use.

8 Acknowledgements

The authors are grateful to Zhaojun Bai and David Day for supplying some of the test problems, and to Iain Duff and Danny Sorensen for their interest and support of this project.

References

E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide.* SIAM, Philadelphia, PA., second edn, 1992.

W. E. Arnoldi. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quarterly of Applied Mathematics*, **9**, 17–29, 1951.

Z. Bai, R. Barrett, D. Day, and J. Dongarra. Nonsymmetric test matrix collection. Research report, Department of Mathematics, University of Kentucky, 1995.

M. Bennani and T. Braconnier. Stopping criteria for eigensolvers. Technical report, CERFACS, November 1994.

T. Braconnier. The Arnoldi–Tchebycheff algorithm for solving large nonsymmetric eigenproblems. Technical Report TR/PA/93/25, CERFACS, Toulouse, France, 1993.

F. Chatelin. *Eigenvalues of Matrices.* Wiley, 1993.

F. Chatelin and V. Fraysée. Qualitative computing: elements of a theory for finite-precision computation. Technical report, CERFACS and THOMSON-CSF, June 1993. Lecture Notes for the Commett European Course, June 8–10, Orsay, France.

J. Cullum and W. E. Donath. A block Lanczos algorithm for computing the q algebraically largest eigenvalues and a corresponding eigenspace for large, sparse symmetric matrices. *In* 'Proceedings of the 1974 IEEE Conference on Decision and Control', pp. 505–509, New York, 1974.

J.J. Dongarra, J. DuCroz, I. S. Duff, and S. Hammarling. A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, **16**(1), 1–17, 1990.

J.J. Dongarra, J. DuCroz, S. Hammarling, and R. J. Hanson. An extended set of Fortran basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, **14**(1), 1–17, 1988.

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

G. H. Golub and R. Underwood. The block Lanczos method for computing eigenvalues. *In* J. R. Rice, ed., 'Mathematical Software III', pp. 361–377, 1977.

R. G. Grimes, J. G. Lewis, and H. D. Simon. A shifted block Lanczos algorithm for solving sparse symmetric generalized eigenproblems. *SIAM Journal on Matrix Analysis and Applications*, **15**(1), 228–272, January 1994.

W. Karush. An iterative method for finding characteristics vectors of a symmetric matrix. *Pacific Journal of Mathematics*, **1**, 233–248, 1951.

C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *Journal of Research of the National Bureau of Standards*, **45**(4), 255–282, October 1950. Research Paper 2133.

C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for Fortran usage. *ACM Transactions on Mathematical Software*, **5**(3), 308–323, 1979.

R. B. Lehoucq. *Analysis and Implementation of an Implicitly Restarted Iteration.* PhD thesis, Rice University, Houston, Texas, May 1995. Also available as Technical Report TR95-13, Dept. of Computational and Applied Mathematics.

R. B. Lehoucq and J. A. Scott. An evaluation of software for computing eigenvalues of sparse nonsymmetric matrices. Technical Report MCS P547, Argonne National Laboratory, 1996*a*. Submitted to ACM Transactions on Mathematical Software.

R. B. Lehoucq and J. A. Scott. An evaluation of subspace iteration software for sparse nonsymmetric eigenproblems. Technical Report RAL-TR-96-022, Rutherford Appleton Laboratory, Chilton, England, 1996*b*.

R. B. Lehoucq and D. C. Sorensen. Deflation techniques within an implicitly restarted iteration. *SIAM Journal on Matrix Analysis and Applications*, 1995. To appear.

R. B. Lehoucq, D. C. Sorensen, P. Vu, and C. Yang. ARPACK: *An implementation of the Implicitly Re-started Arnoldi Iteration that computes some of the eigenvalues and eigenvectors of a large sparse matrix*, 1995. Available from netlib@ornl.gov under the directory scalapack.

K. Meerbergen, A. Spence, and D. Roose. Shift-invert and Cayley transforms for the detection of rightmost eigenvalues of nonsymmetric matrices. *BIT*, **34**, 409–423, 1994.

C. C. Paige. *The computation of eigenvalues and eigenvectors of very large sparse matrices.* PhD thesis, University of London, London, England, 1971.

B. N. Parlett. *The Symmetric Eigenvalue Problem.* Prentice-Hall, 1980.

Y. Saad. Variations on Arnoldi's method for computing eigenelements of large unsymmetric matrices. *Linear Algebra and Its Applications*, **34**, 269–295, 1980.

Y. Saad. Chebyshev acceleration techniques for solving nonsymmetric eigenvalue problems. *Mathematics of Computation*, **42**, 567–588, 1984.

Y. Saad. *Numerical Methods for Large Eigenvalue Problems.* Halsted Press, 1992.

M. Sadkane. A block Arnoldi–Chebyshev method for computing the leading eigenpairs of large sparse unsymmetric matrices. *Numerische Mathematik*, **64**, 181–193, 1993.

J. A. Scott. An Arnoldi code for computing selected eigenvalues of sparse real unsymmetric matrices. *ACM Transactions on Mathematical Software*, **21**, 432–475, 1995.

G. L. G. Sleijpen and H.A. Van der Vorst. A Jacobi-Davidson iteration method for linear eigenvalue problems. Technical Report 856 (revised), University Utrecht, Department of Mathematics, 1995.

B. T. Smith, J. M. Boyle, J. J. Dongarra B. S. Garbow, Y. Ikebe, V. C. Klema, and C. B. Moler. *EISPACK Guide.* Springer–Verlag, Berlin, second edn, 1976. Volume 6 of Lecture Notes in Computer Science.

D. C. Sorensen. Implicit application of polynomial filters in a k-step Arnoldi method. *SIAM Journal on Matrix Analysis and Applications*, **13**(1), 357–385, January 1992.

D. C. Sorensen. Implicitly restarted Arnoldi/Lanczos methods for large scale eigenvalue calculations. *In* D. E. Keyes, A. Sameh and V. Venkatakrishnan, eds, 'Proceedings of the ICASE/LaRC Workshop on Parallel Numerical Algorithms, May 23-25, 1994', Kluwer, Norfolk, Va, 1995. To appear.

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

J. H. Wilkinson. *The Algebraic Eigenvalue Problem.* Clarendon Press, Oxford, UK, 1965.

A Appendix: Tables of results for right-most eigenpair

For examples for which the right-most eigenvalue is also the eigenvalue of largest modulus, results are given for ARPACK and EB13 when the right-most eigenvalue is requested (WHICH = LR) and the eigenvalue of largest modulus is requested (WHICH = LM).

PORES2, $r = 1, m = 8$

Code	WHICH	Matrix-vector products	Iterations	Time (secs)
ARNCHEB	LR	1401	7	3.37
ARPACK	LR	90	5	0.50
EB13	LR	119	3	0.40

PORES3, $r = 1, m = 8$

Code	WHICH	Matrix-vector products	Iterations	Time (secs)
ARNCHEB	LR	$*$	$*$	$*$
ARPACK	LM	100	24	0.23
ARPACK	LR	$*$	$*$	$*$
EB13	LM	217	27	0.40
EB13	LR	$*$	$*$	$*$

$*$ denotes that code did not converge with the requested accuracy within $4000m$ matrix-vector products.

GRE1107, $r = 1, m = 8$

Code	WHICH	Matrix-vector products	Iterations	Time (secs)
ARNCHEB	LR	2204	47	4.00
ARPACK	LM	420	119	1.94
ARPACK	LR	320	71	1.28
EB13	LM	929	116	2.85
EB13	LR	465	7	1.05

HOR131, $r = 1, m = 8$

Code	WHICH	Matrix-vector products	Iterations	Time (secs)
ARNCHEB	LR	496	2	0.53
ARPACK	LM	28	6	0.07
ARPACK	LR	28	6	0.07
EB13	LM	32	7	0.08
EB13	LR	57	2	0.09

IMPCOLC, $r = 1, m = 8$

Code	WHICH	Matrix-vector products	Iterations	Time (secs)
ARNCHEB	LR	360	9	0.06
ARPACK	LM	200	34	0.15
ARPACK	LR	59	14	0.06
EB13	LM	181	20	0.08
EB13	LR	117	3	0.02

IMPCOLD, $r = 1, m = 8$

Code	WHICH	Matrix-vector products	Iterations	Time (secs)
ARNCHEB	LR	†	†	†
ARPACK	LR	†	†	†
EB13	LR	1963	24	1.29

† indicates sought-after eigenvalue was missed.

NNC666, $r = 1, m = 8$

Code	WHICH	Matrix-vector products	Iterations	Time (secs)
ARNCHEB	LR	912	10	1.00
ARPACK	LR	132	32	0.35
EB13	LR	117	3	0.18

NNC1374, $r = 1, m = 8$

Code	WHICH	Matrix-vector products	Iterations	Time (secs)
ARNCHEB	LR	1758	16	4.11
ARPACK	LR	176	43	1.15
EB13	LR	201	4	0.66

WEST0156, $r = 1, m = 8$

Code	WHICH	Matrix-vector products	Iterations	Time (secs)
ARNCHEB	LR	195	1	0.03
ARPACK	LR	38	6	0.04
EB13	LR	59	2	0.02

WEST0167, $r = 1, m = 8$

Code	WHICH	Matrix-vector products	Iterations	Time (secs)
ARNCHEB	LR	367	8	0.08
ARPACK	LR	135	38	0.19
EB13	LR	58	2	0.01

WEST2021, $r = 1, m = 8$

Code	WHICH	Matrix-vector products	Iterations	Time (secs)
ARNCHEB	LR	3233	52	8.64
ARPACK	LR	401	103	3.73
EB13	LR	4869	57	17.34

PDE2961, $r = 1, m = 8$

Code	WHICH	Matrix-vector products	Iterations	Time (secs)
ARNCHEB	LR	3299	14	13.36
ARPACK	LR	180	30	2.29
EB13	LR	204	4	1.37

RWK5151, $r = 1, m = 8$

Code	WHICH	Matrix-vector products	Iterations	Time (secs)
ARNCHEB	LR	5167	1	19.18
ARPACK	LR	350	88	7.44
EB13	LR	905	12	7.71

B Appendix: Tables of results for several eigenvalues

PORES2, $r = 4, m = 20$

Code	WHICH	Block size	Matrix-vector products	Iterations	Time (secs)
ARNCHEB	LR	1	1712	8	4.74
ARPACK	LR	1	230	17	1.92
EB13	LR	1	1759	25	6.99
EB13	LR	4	$*$	$*$	$*$

$*$ denotes that code did not converge with
the requested accuracy within $4000m$ matrix-vector products.

PORES3, $r = 5, m = 20$

Code	WHICH	Block size	Matrix-vector products	Iterations	Time (secs)
ARNCHEB	LR	1	$*$	$*$	$*$
ARPACK	LR	1	11684	782	38.19
EB13	LR	1	$*$	$*$	$*$
EB13	LR	5	$*$	$*$	$*$

$*$ denotes that code did not converge with
the requested accuracy within $4000m$ matrix-vector products.

GRE1107, $r = 5, m = 20$

Code	WHICH	Block size	Matrix-vector products	Iterations	Time (secs)
ARNCHEB	LR	1	*	*	*
ARPACK	LM	1	227	20	1.54
ARPACK	LR	1	260	18	1.42
EB13	LM	1	1558	77	5.91
EB13	LR	1	1950	21	4.95
EB13	LM	5	3925	196	19.05
EB13	LR	5	1449	6	2.69

$*$ denotes that code did not converge with
the requested accuracy within $4000m$ matrix-vector products.

HOR131, $r = 5, m = 20$

Code	WHICH	Block size	Matrix-vector products	Iterations	Time (secs)
ARNCHEB	LR	1	749	5	0.98
ARPACK	LM	1	46	3	0.15
ARPACK	LR	1	45	3	0.14
EB13	LM	1	221	11	0.49
EB13	LR	1	237	5	0.40
EB13	LM	5	505	25	1.04
EB13	LR	5	245	2	0.23

IMPCOLC, $r = 5, m = 20$

Code	WHICH	Block size	Matrix-vector products	Iterations	Time (secs)
ARNCHEB	LR	1	†	†	†
ARPACK	LR	1	188	16	0.33
EB13	LR	1	436	6	0.16
EB13	LR	5	245	2	0.06

† indicates sought-after eigenvalues were missed.

IMPCOLD, $r = 5, m = 20$

Code	WHICH	Block size	Matrix-vector products	Iterations	Time (secs)
ARNCHEB	LR	1	$*$	$*$	$*$
ARPACK	LR	1	793	61	0.72
EB13	LR	1	2950	31	2.38
EB13	LR	5	12656	64	7.73

$*$ denotes that code did not converge with the requested accuracy within $4000m$ matrix-vector products.

NNC666, $r = 5, m = 20$

Code	WHICH	Block size	Matrix-vector products	Iterations	Time (secs)
ARNCHEB	LR	1	2246	28	3.48
ARPACK	LR	1	102	7	0.40
EB13	LR	1	419	6	0.75
EB13	LR	5	423	3	0.51

NNC1374, $r = 5, m = 20$

Code	WHICH	Block size	Matrix-vector products	Iterations	Time (secs)
ARNCHEB	LR	1	†	†	†
ARPACK	LR	1	145	10	1.24
EB13	LR	1	557	7	2.07
EB13	LR	5	700	4	1.83

† indicates sought-after eigenvalues were missed.

WEST0156, $r = 5, m = 20$

Code	WHICH	Block size	Matrix-vector products	Iterations	Time (secs)
ARNCHEB	LR	1	251	1	0.06
ARPACK	LR	1	29	2	0.05
EB13	LR	1	25	1	0.02
EB13	LR	5	286	4	0.12

WEST0167, $r = 5, m = 20$

Code	WHICH	Block size	Matrix-vector products	Iterations	Time (secs)
ARNCHEB	LR	1	317	2	0.09
ARPACK	LR	1	91	7	0.15
EB13	LR	1	26	1	0.02
EB13	LR	5	434	5	0.15

WEST2021, $r = 5, m = 20$

Code	WHICH	Block size	Matrix-vector products	Iterations	Time (secs)
ARNCHEB	LR	1	15921	252	70.67
ARPACK	LR	1	167	13	2.12
EB13	LR	1	4149	42	18.07
EB13	LR	5	6865	19	20.96

CK400, $r = 8, m = 20$

Code	WHICH	Block size	Matrix-vector products	Iterations	Time (secs)
ARNCHEB	LR	1	†	†	†
ARPACK	LM	1	71	7	0.20
ARPACK	LR	1	80	8	0.40
EB13	LM	1	223	12	0.39
EB13	LR	1	228	9	0.35
EB13	LM	4	340	17	0.58
EB13	LR	4	256	6	0.28

† indicates sought-after eigenvalues were missed.

CK656, $r = 8, m = 20$

Code	WHICH	Block size	Matrix-vector products	Iterations	Time (secs)
ARNCHEB	LR	1	†	†	†
ARPACK	LM	1	80	8	0.40
ARPACK	LR	1	80	8	0.40
EB13	LM	1	254	13	0.72
EB13	LR	1	238	9	0.58
EB13	LM	4	356	18	1.02
EB13	LR	4	224	5	0.49

† indicates sought-after eigenvalues were missed.