An Evaluation of Software for Computing Eigenvalues of Sparse Nonsymmetric Matrices

R. B. Lehoucq* J. A. Scott†

January 22, 1996

Abstract

The past few years have seen a significant increase in research into numerical methods for computing selected eigenvalues of large sparse nonsymmetric matrices. This research has begun to lead to the development of high-quality mathematical software. The software includes codes that implement subspace iteration methods, Arnoldi-based algorithms, and nonsymmetric Lanczos methods. The aim of the current study is to evaluate this state-of-the-art software. In this study we consider subspace iteration and Arnoldi codes. We look at the key features of the codes and their ease of use. Then, using a wide range of test problems, we compare the performance of the codes in terms of storage requirements, execution times, accuracy, and reliability. We also consider their suitability for solving large-scale industrial problems. Based on our findings, we suggest how improved software should be designed.

AMS classification: Primary 65F15; Secondary 65G05

Key Words : Eigenvalues, Arnoldi method, Lanczos method, subspace iteration.

*Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439 lehoucq@mcs.anl.gov. This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38. Support was also provided by the National Science Foundation cooperative agreement CCR-9120008, and by ARPA contract number DAAL03-91-C-0047 administered by the U.S. Army Research Office.

†Computing and Information Systems Department, Rutherford Appleton Laboratory, Didcot, Oxon OX11 0QX, England sct@letterbox.rl.ac.uk.

1 Introduction

In this paper we are concerned with the standard eigenvalue problem

$$\mathbf{A}\mathbf{z} = \lambda\mathbf{z}, \tag{1.1}$$

where $\mathbf{A}$ is a large sparse real nonsymmetric matrix. The past few years have seen considerable research into numerical methods for computing selected eigenvalues and eigenvectors of (1.1). This has led to the development of new software and to papers and reports describing the usefulness of the software for solving practical problems (see, for example, Duff and Scott, 1993; Bai and Stewart, 1992; Braconnier, 1993; Sorensen, 1995; Scott, 1995). However; the published numerical results are extremely limited, and, in general, authors of software have provided few results comparing the performance of their software with that of rival software. The recent books by Saad (1992) and Chatelin (1993) consider the state-of-the-art of large eigenproblem techniques, and the progress report by Bai (1995) provides a useful review of the origins of eigenvalue problems and algorithmic techniques. In the midst of all this activity, there remains a lack of comparative numerical results for current software, and this motivates our current study. We aim to review, compare, and evaluate software for sparse nonsymmetric eigenvalue problems in terms of the following criteria:

- the user interface,
- storage requirements,
- performance,
- accuracy and stability, and
- reliability and robustness.

To keep the current study to a reasonable length, we restrict our attention to algorithms that require only matrix-vector products with $\mathbf{A}$; we do not examine methods that require linear equations involving $\mathbf{A}$ to be solved. This focus allows us to examine the quality of the underlying algorithm and associated software. We refer the reader to the survey by Meerbergen and Roose (1994) for a review of methods that use spectral transformations that require the solution of linear equations involving $\mathbf{A}$.

Three methods have received significant attention by the numerical analysis community. These are subspace (or simultaneous) iteration, Arnoldi's

method, and the (nonsymmetric) Lanczos method. Our intention is to provide a comprehensive comparative study of these three methods, along with the recent Jacobi-Davidson method of Sleijpen and Van der Vorst (1995).

In this paper we consider only subspace iteration and Arnoldi methods, for which several high-quality codes have been written. The scope of our study is restricted to software that is available either in the public domain or under licence. For the Lanzcos method, there is currently only a very limited amount of such software. As far as we are aware, the only code that falls within the criteria for inclusion in this study is the code `EIGLAL` of Freund, Gutknecht and Nachtigal (1993). Recent work on Lanzcos algorithms has been published by Cullum (1994), and a new Lanzcos code, `ABLE`, is currently under development (Bai, Day and Ye, 1995*b*). At present, there is no software implementing the Jacobi-Davidson method that meets our criteria. As software for these methods becomes available, however, we plan to evaluate and compare it with subspace iteration and Arnoldi-based software.

This paper is organized as follows. We briefly review subspace iteration in Section 2 and Arnoldi's method in Section 3. The main features of the available software are discussed in Section 4. In Section 5 we compare the software in terms of how matrix-vector products are performed, the use of BLAS and LAPACK, the stopping criteria, storage requirements, and user interfaces. The design of our experiments to compare the performance of the software is the subject of Section 6. We explain how we verified the computed results and present numerical results for our test matrices. Based on our experiences with the different codes, we consider in Section 7 the features that we would like to see incorporated in new software to compute eigenvalues of nonsymmetric matrices. General conclusions are also drawn.

We end this section by introducing the notation that is used throughout this paper.

- 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 eigenvalues are sought. For example, if the rightmost eigenvalues are required, the eigenvalues are ordered in decreasing order of their real parts. Subscripts are dropped when no confusion will result.

- n denotes the order of $\mathbf{A}$.

- r denotes the number of sought-after eigenvalues of $\mathbf{A}$.

- m denotes the dimension of the subspace used in the subspace iteration and Arnoldi algorithms.
- $\mathbf{X}_m = \left[\begin{array}{cc} \mathbf{X}_r & \bar{\mathbf{X}}_{m-r} \end{array} \right]$ denotes the matrix representation of this subspace.
- $(\mathbf{s}, \theta)$ denotes an eigenpair of the projection matrix 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 relative machine precision.
- ϵ denotes the user-prescribed convergence tolerance.
- $\mathbf{e}_j$ denotes the j-th canonical basis vector.

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

2 Subspace Iteration

We briefly recall the main ideas behind the subspace iteration algorithm. Subspace iteration was originally introduced by Bauer (1957), who called the method *Treppeniteration* (staircase iteration). It is a straightforward method for computing the eigenvalues of largest modulus of a real nonsymmetric matrix and is a generalization of the power method. It has been widely used and remains particularly popular in structural engineering. Starting with an initial $n \times m$ matrix $\mathbf{X}_m$ with linearly independent normalized columns $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m$ (called the "trial" vectors), the subspace iteration method generates a sequence of $n \times m$ matrices as follows:

1. *Start:* Choose an initial set of normalized vectors $\mathbf{X}_m \leftarrow \left[\begin{array}{cccc} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_m \end{array} \right]$.
2. *Iteration:* Until convergence **do**

- Choose $l > 1$ and compute $\mathbf{X}_m \leftarrow \mathbf{A}^l \mathbf{X}_m$.
- QR factorization: $\mathbf{Q}_m \mathbf{R}_m = \mathbf{X}_m$ and set $\mathbf{X}_m \leftarrow \mathbf{Q}_m$.
- Projection: form $\mathbf{B}_m = \mathbf{X}_m^T \mathbf{A} \mathbf{X}_m$.
- Schur form: compute the real Schur form $\mathbf{T}_m = \mathbf{V}_m^T \mathbf{B}_m \mathbf{V}_m$, and set $\mathbf{X}_m \leftarrow \mathbf{X}_m \mathbf{V}_m$.

In some variations of the algorithm, the QR factorization is not performed (see, for example, Stewart and Jennings, 1981*b*). In place of the above projection, $\mathbf{B}_m = (\mathbf{X}_m^T \mathbf{X}_m)^{-1} \mathbf{X}_m^T \mathbf{A} \mathbf{X}_m$ is computed.

If one assumes the eigenvalues are ordered so that $|\lambda_1| \geq |\lambda_2| \geq \ldots \geq |\lambda_n|$, if $|\lambda_m| > |\lambda_{m+1}|$, it may be shown (under mild restrictions on the initial set of trial vectors) that the columns of $\mathbf{X}_m$ converge to a basis for the invariant subspace of $\mathbf{A}$ corresponding to the m dominant eigenvalues. Convergence is linear with the rate $|\lambda_{m+1}/\lambda_i|$ for the ith column of $\mathbf{X}_m$.

Further details and discussion of subspace iteration may be found, for example, in Stewart (1976*b*), in Watkins and Elsner (1991), and in the recent books by Chatelin (1993) and Saad (1992).

3 Arnoldi's Method

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 $\mathbf{A}$:

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

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 realized. 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. The restarted Arnoldi method may be summarized as follows:

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

$\mathbf{H}_m$ 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 $\mathbf{H}_m = \mathbf{X}_m^T\mathbf{A}\mathbf{X}_m$ 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.

4 Available Software

Numerous research codes for solving the sparse nonsymmetric eigenproblem have been developed over the years. In addition, there are implementations of the subspace iteration and Arnoldi methods embedded within much larger codes, for example, within the `MSC` engineering application software (Komzsik, 1995). However, very little library-quality software has been developed. As discussed in Section 1, in this study we aim to evaluate software that is available through the public domain or under commercial license. The software must also have been designed by its authors for use by a nonexpert. Specifically, it must not demand that the user have a detailed knowledge of

the underlying numerical algorithm. In addition, the software must be supplied with documentation and must be implemented by using either the C or `Fortran` programming languages.

In Table 1 we list all the subspace iteration and Arnoldi codes we are aware of that appear to meet the above criteria. These are the codes that will be evaluated in our current study. (We apologize if we have failed to include in our evaluation study any other codes that meet our criteria.)

Table 1: Subspace iteration and Arnoldi software for the sparse nonsymmetric eigenproblem (ERA denotes explicitly restarted Arnoldi, and IRA denotes implicitly restarted Arnoldi)

Code	Method	Authors	Year	Availability
`LOPSI`	Subspace	Stewart and Jennings	1981	TOMS
`SRRIT`	Subspace	Stewart and Bai	1993	ftp
`EB12`	Subspace	Duff and Scott	1991	HSL
`ARNCHEB`	ERA	Braconnier	1993	ftp
`EB13`	ERA	Scott	1993	HSL
`ARPACK`	IRA	Sorensen, Lehoucq, and Vu	1995	netlib

In Table 1, TOMS denotes the ACM Transactions on Mathematical Software; ftp indicates that the code is available by anonymous ftp; HSL denotes the Harwell Subroutine Library (1996); and, finally, netlib indicates that the code may be obtained through a software repository (Dongarra and Grosse, 1987) on the Internet. Full details of how to obtain the codes are given in Section 8.

In the remainder of this section, we briefly discuss each of the codes listed in Table 1. Further details may be found in Lehoucq and Scott (1995*a*, 1995*b*). Note that all the codes are written in `Fortran 77`.

4.1 LOPSI

The code `LOPSI` of Stewart and Jennings (1981*a*), which is based on work done in the 1970s by Clint and Jennings (1971) and Jennings and Stewart (1975), has been available for more than a decade. It uses subspace iteration combined with a *lopsided* oblique projection to compute the eigenvalues of

$\mathbf{A}$ of largest modulus together with the corresponding eigenvectors.

After the first iteration, the number l of premultiplications by $\mathbf{A}$ between projections is chosen to avoid both the dominant eigenvector components "swamping" the predictions for the lower eigenvectors and unnecessary work being performed by carrying the iteration beyond the stage at which convergence is achieved. The restriction $1 \leq l \leq l_{max}$, where l_{max} is a user-defined parameter, is also imposed. The value of l_{max} recommended by Stewart and Jennings is 10.

To increase efficiency when more than one eigenvalue is required, `LOPSI` incorporates a deflation strategy whereby a column of $\mathbf{X}_m$ is *locked* as soon as it has converged. Locking means that no further computations are carried out with this vector.

Matrix-vector products $\mathbf{Ax}$ are performed within `LOPSI` by an internal subroutine `PREMULT`. The matrix $\mathbf{A}$ must be passed to `LOPSI` by using the coordinate storage scheme; that is, the matrix must be held as an unordered set of triples (a_{ij}, i, j) using a real array and two integer arrays, of length equal to the number of (nonzero) entries in $\mathbf{A}$. To ensure finite termination, the user is required to specify the maximum number of matrix-vector products that may be performed.

4.2 SRRIT

The recent subspace iteration code `SRRIT` of Bai and Stewart (1992) is a revised version of a code by Stewart (1978) of the same name. It computes an orthonormal basis for the invariant subspace corresponding to the eigenvalues of largest modulus.

As in the code `LOPSI`, efficiency is increased by locking columns as soon as they have converged. `SRRIT` also gives the user the option of supplying the initial basis $\mathbf{X}_m$. The user must supply a subroutine `ATQ` to perform matrix products $\mathbf{AX}_m$. A user-defined parameter limits the number of calls to this subroutine and ensures finite termination of the code. The subroutine `ATQ` 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 necessary.

An iterated modified Gram-Schmidt algorithm with possible reorthogonalization is used to maintain the orthogonality of the columns of $\mathbf{X}_m$. An internal parameter controls the maximum number of reorthogonalizations that may be performed; this is set to 5. At each iteration, the code computes the value of l so that the columns of $\mathbf{A}^l\mathbf{X}_m$ remain linearly independent and orthogonalizations are minimized. `SRRIT` also determines the

number of iterations before a Schur–Rayleigh–Ritz projection is performed. A projection is performed only when it is anticipated that one or more of the columns of $\mathbf{X}_m$ satisfy the convergence criterion (see Section 5.3).

4.3 EB12

The Harwell Subroutine Library code `EB12` (Duff and Scott, 1993) is the most general subspace iteration package considered in this study, since it is designed to calculate either the right-most or the left-most eigenvalues of $\mathbf{A}$, or the eigenvalues that are of largest modulus. `EB12` computes the right-most (or left-most) eigenvalues by replacing the power $\mathbf{A}^l$ in the subspace iteration algorithm by a Chebychev polynomial $p_l(\mathbf{A})$ on an ellipse containing the unwanted Ritz values. The idea of using a Chebychev polynomial to accelerate convergence was proposed by Saad (1984). The evaluation of $p_l(\mathbf{A})\mathbf{x}$ is carried out by using the three-term recurrence relation for Chebychev polynomials.

In `EB12`, the columns of $\mathbf{X}_m$ are orthonormalized by using the modified Gram-Schmidt algorithm. On each iteration, the degree l of the iteration polynomial is chosen to try to ensure that the columns of $\mathbf{X}_m$ remain linearly independent. If Chebychev acceleration is employed, l is also chosen to ensure the ellipse is updated sufficiently often. Furthermore, l is limited close to convergence to prevent unnecessary work from being performed.

As in the other codes, `EB12` incorporates locking techniques to reduce the computational effort if more than one eigenvalue has been requested. `EB12` also allows the user to supply the initial basis vectors.

The code `EB12` uses reverse communication. Each time a set of vectors is required to be multiplied by $\mathbf{A}$, control is returned to the user. The advantages of this approach are discussed in Section 5.1. The maximum number of matrix-vector products $\mathbf{Ax}$ is limited by a parameter that is held in a `COMMON` block.

Once `EB12` has successfully computed the required eigenvalues of $\mathbf{A}$, the user may call a separate subroutine, `EB12B`, to compute the corresponding (normalized) eigenvectors and, optionally, the scaled eigenvector residuals $\|\mathbf{Ay} - \theta\mathbf{y}\|_2/\|\mathbf{Ay}\|_2$.

4.4 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 parts.

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. As in EB12, 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. As for SRRIT, the user must supply a subroutine to form $\mathbf{Ax}$. The package offers the user the option of using Wielandt deflation (see Wilkinson, 1965, and Saad, 1992, for further details). As the individual Ritz values converge, the code forms the rank j modification $\mathbf{A}_j = \mathbf{A} - \mathbf{U}_j\mathbf{S}_j\mathbf{U}_j^T$, where $\mathbf{S}_j$ is a diagonal matrix of order j representing the dimension of the approximate invariant subspace that has already converged. The idea is to choose $\mathbf{S}_j$ so that $\mathbf{A}_j$ will converge to the remainder of the invariant subspace desired. Unfortunately, the eigenvectors of $\mathbf{A}_j$ are not those of $\mathbf{A}$. We remark that the column space $\mathbf{U}_j$ is invariant for $\mathbf{A}$, and thus it is possible for ARNOL to compute Ritz vectors when using Wielandt deflation. The extra computation is not trivial and must be carried out by the user. Moreover, there is no documentation to guide the user on how this could be done.[1]

4.5 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 Schur vectors corresponding to Ritz values that converge. At each iteration, the restart vector is taken to be the first unconverged approximate Schur vector. A Chebychev polynomial $p_l(\mathbf{A})$ on an ellipse containing the unwanted

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

Ritz values is applied to the restart vector in an attempt to accelerate convergence. The code automatically 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 orthogonalize the Arnoldi basis vectors. As with `EB12`, a reverse communication mechanism is used for computing matrix-vector products with $\mathbf{A}$.

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 (normalized) eigenvectors and, optionally, the scaled eigenvector residuals $\|\mathbf{Ay} - \theta\mathbf{y}\|_2/\|\mathbf{Ay}\|_2$.

4.6 ARPACK

The `ARPACK` software package (Lehoucq, Sorensen and Vu, 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. Suppose that ψ is a polynomial of degree $m - r$. A simple but tedious derivation shows that

$$\psi(\mathbf{A})\mathbf{X}_r \;=\; \mathbf{X}_m\psi(\mathbf{H}_m)\begin{bmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \cdots & \mathbf{e}_r \end{bmatrix}. \tag{4.1}$$

Partitioning the QR factorization of $\psi(\mathbf{H}_m)$ as

$$\mathbf{Q}_m\mathbf{R}_m = \begin{bmatrix} \mathbf{Q}_r & \bar{\mathbf{Q}}_{m-r} \end{bmatrix}\begin{bmatrix} \mathbf{R}_r & \mathbf{M}_r \\ 0 & \bar{\mathbf{R}}_{m-r} \end{bmatrix}$$

allows Equation (4.1) to 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 $\psi(\mathbf{A})\mathbf{X}_r$. In other words, an IRA-iteration is equivalent to performing subspace iteration with $\mathbf{X}_r$—while avoiding matrix-vectors products in $\mathbf{A}$.

Restarting the iteration involves post-multiplying the length m Arnoldi factorization with $\mathbf{Q}_m$ and then retaining the first r columns. 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 orthogonalize 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 are locked. As in `EB12` and `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 generalized eigenvalue problem, $\mathbf{Az} = \lambda\mathbf{Bz}$, when $\mathbf{B}$ is a symmetric positive semi-definite matrix.

Finally, analogous to the approach of `EB12` and `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.

5 Software Comparison

5.1 Matrix-Vector Products AX

A major implementation difference is the way in which the software copes with forming the product of $\mathbf{A}$ with sets of vectors. For large problems, this process can represent the dominant cost. Thus, it is important to minimize the number of times that $\mathbf{A}$ is applied and to ensure that the process is implemented efficiently.

The code `LOPSI` is the most restrictive of the codes in our study because it requires the user to pass the matrix to the routine by using a defined storage scheme. The matrix-vector products $\mathbf{AX}_m$ are all performed by a single subroutine within the code, and the authors comment in their paper that considerable savings can be obtained by converting this subroutine to machine code (see Stewart and Jennings, 1981*b*). Clearly this conversion involves intervention by the user. A user could also change the storage scheme used for $\mathbf{A}$ to one more suited to his or her problem, but this change would involve some effort, and no documentation is provided to assist with this.

`SRRIT` and `ARNCHEB` adopt a somewhat more flexible approach by requiring the user to supply the subroutine to perform the matrix-vector products. Even though $\mathbf{A}$ is not required to be held explicitly, for some problems it

can be inconvenient for the user to pass the matrix into this subroutine. For example, since both `SRRIT` and `ARNCHEB` use the `Fortran` programming language, the number of subroutine arguments is fixed. If a user needs additional descriptors to perform matrix-vector products, these must be passed through a `COMMON` block.

The reverse communication approach used by the remaining codes provides the most flexibility and gives the user the greatest degree of control. The user is able to exploit the sparsity and structure of the matrix and, by avoiding passing the matrix through a `COMMON` block, can take full advantage of parallelism and/or vectorization. Another obvious advantage of reverse communication is that the user is able to incorporate different preconditioning techniques in a straightforward way. For example, the user may wish to use a shift-and-invert transformation, in which a matrix of the form $(\mathbf{A} - \sigma\mathbf{I})^{-1}$ 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-vector products $\mathbf{w} = \mathbf{A}\mathbf{x}$. If a direct method of solution is used, the LU factorization of $(\mathbf{A} - \sigma\mathbf{I})$ need be performed only once. However, since reverse communication allows progress to be monitored, the user may choose to update σ as the computation progresses, thereby requiring a new factorization for each shift.

5.2 The Use of BLAS and LAPACK

Apart from the matrix-vector products $\mathbf{A}\mathbf{x}$, the subspace and Arnoldi algorithms require only dense linear algebra operations to be performed on matrices of order m. One way of achieving an efficient implementation and assisting with robustness, portability, and readability 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). The codes in our study use the BLAS to very different degrees. For instance, when `LOPSI` was developed, only the Level 1 BLAS were available, and `LOPSI`, in fact, makes no use of these kernels.

`EB12`, `ARNCHEB`, and `EB13` employ mainly Level 1 and Level 2 BLAS kernels. In addition, they use some EISPACK routines (Smith, Boyle, Garbow, Ikebe, Klema and Moler, 1976). `EB12` and `EB13` use modified versions of the EISPACK routines `ORTHES` and `ORTRAN` and of the routine `HQR3` given by Stewart (1976*a*).

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 to achieve much greater efficiency, where possible, on modern high-performance computers. This was accomplished by writing routines that call all three levels of the BLAS. The updating of the original 1978 version of `SRRIT` included using BLAS kernels and replacing EISPACK routines with ones from LAPACK. `ARPACK` also makes extensive use of BLAS and LAPACK, and we anticipate that this will be reflected in its performance. The use of BLAS and LAPACK also make the software potentially easier to maintain.

5.3 The Stopping Criteria

The codes all use different stopping criteria, thereby complicating performance comparisons (see Section 6). A useful discussion of stopping criteria for iterative eigensolvers is given by Scott (1995). Throughout this section, ϵ denotes a user-defined tolerance.

For subspace iteration, the dominant eigenvalues converge most rapidly so the codes test only the $(j+1)$-st eigenvalue after the jth one has converged. In the code `LOPSI`, a column of $\mathbf{X}_m$ is accepted as an approximation to an eigenvector of $\mathbf{A}$ when it becomes nearly stationary. Specifically, if $\mathbf{X}_m^{(k)}$ is $\mathbf{X}_m$ on the kth iteration ($k \geq 0$), the jth column of $\mathbf{X}_m^{(k)}$ is accepted if it satisfies the inequality

$$\|(\mathbf{X}_m^{(k)} - \mathbf{X}_m^{(k-1)})_j\|_\infty \quad < \quad \epsilon.$$

Such a stopping criterion may fail in the case of $\mathbf{A}$ having equal, or nearly equal, eigenvalues.

In an attempt to overcome this problem, `SRRIT`, `EB12`, and `EB13` follow Stewart (1978) and base their stopping criterion on demanding that $\mathbf{A}\mathbf{X}_r \approx \mathbf{X}_r\mathbf{T}_r$. In `SRRIT`, the jth column of $\mathbf{X}_m$ is said to have converged if

$$\|(\mathbf{A}\mathbf{X}_m - \mathbf{X}_m\mathbf{T}_m)_j\|_2 \quad < \quad |\theta_j|\epsilon,$$

where θ_j is the jth eigenvalue of $\mathbf{T}_m$. At each iteration, the residuals $\|(\mathbf{A}\mathbf{X}_m - \mathbf{X}_m\mathbf{T}_m)_j\|_2$ are computed for $j = i+1, \ldots, r$, where $i+1$ points to the first unconverged eigenvalue. The code groups eigenvalues that have nearly equal moduli. The eigenvalues computed on the previous iteration are also grouped. If the two groups have the same number of eigenvalues

and the average value of the eigenvalues has settled down, the residuals are averaged and tested against ϵ.

The convergence criterion used by `EB12` for the jth column of $\mathbf{X}$ requires

$$\|(\mathbf{A}\mathbf{X}_m - \mathbf{X}_m\mathbf{T}_m)_j\|_2 \quad < \quad \|(\mathbf{A}\mathbf{X}_m)_j\|_2\epsilon. \tag{5.1}$$

The residual $\|(\mathbf{A}\mathbf{X}_m - \mathbf{X}_m\mathbf{T}_m)_j\|_2$ is computed only if all the basis vectors $\mathbf{X}_i$ with $0 < i < j$ have already been accepted. `EB12` 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 to obtain the requested accuracy. Facilities are included for restarting the computation from the point at which the warning was issued.

`EB13` allows the user a choice of stopping criteria. In addition to offering (5.1), the user has the choice of requiring that $\|(\mathbf{A}\mathbf{X}_m - \mathbf{X}_m\mathbf{T}_m)_j\|_2 < \|\mathbf{A}\|\epsilon$ or $< \epsilon$. The advantage of using the norm of $\mathbf{A}$ is that the stopping criterion is based on the backward error. A disadvantage is that it requires $\|\mathbf{A}\|$ to be known. `EB13` requests the user to provide $\|\mathbf{A}\|$ (or an estimate of $\|\mathbf{A}\|$). If the user is unable to do this, the code will compute the Frobenius norm of $\mathbf{A}$ at the cost of n matrix-vector products.

The most compelling reason for possibly not wishing to use a stopping criterion that involves the norm of $\mathbf{A}$ 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 $\mathbf{A}$ 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 $\mathbf{A}$ is to be used, the user should take its size into account when selecting the convergence tolerance.

In a similar manner to `EB12`, `EB13` attempts to terminate the computation if convergence appears to have stagnated.

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

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

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 $\|\mathbf{A}\mathbf{X}_m\mathbf{s} - \mathbf{X}_m\mathbf{s}\theta\|$. ARPACK bases its stopping criterion on the Ritz estimate. Moreover, since only the last component of $\mathbf{s}$ 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

$$\|\mathbf{f}_m\| \, |\mathbf{e}_m^T \mathbf{s}| < |\theta|\epsilon.$$

We remark that since $\|(\mathbf{A}\mathbf{X}_m)_j\|_2 \leq ||\mathbf{A}||_2$ and $|\theta| \leq ||\mathbf{A}||_2$, ARPACK, EB12, and SRRIT also base their stopping criterion on the backward error. Moreover, the user should consider the size of $|\theta|$ when selecting ϵ for these codes.

Recent work by Chatelin (1993) and Bennani and Braconnier (1994) suggests that when $\mathbf{A}$ 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

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

respectively, where $\|\mathbf{A}\|_F$ denotes the Frobenius norm of $\mathbf{A}$. 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.

5.4 Storage Requirements

For large problems, the amount of storage needed can be an important consideration when selecting a code. In Table 2 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.

Assuming a block size $n_b = 1$, we see that EB12 and EB13 each need three arrays of length nm while SRRIT requires only two such arrays. There are two reasons why EB12 and EB13 demand 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, EB12 and EB13 use 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 **U** and **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, `EB12` and `EB13` demand three arrays of length nm.

Table 2: Storage requirements (n_b denotes the block size for `EB13`)

Code	Storage
`LOPSI`	$3n \times m + 4m^2 + O(m)$
`SRRIT`	$2n \times m + 2m^2 + O(m)$
`EB12`	$3n \times m + 2m^2 + O(m)$
`ARNCHEB`	$3n \times (m+5) + 2m^2 + O(m)$
`EB13`	$3n \times m \times n_b + 2m^2 + O(m)$
`ARPACK`	$n \times (m+4) + 3m^2 + O(m)$

5.5 User Interface

An important feature of any code written for general use is that it should be accompanied by straightforward but comprehensive documentation that allows the code to be used with a minimum of effort. The code and documentation should also offer assistance to the user in the event of the computation failing. Our numerical experiments have provided us with a feel for how easy the software and its documentation are to use, and in this section we comment briefly on our experience in using the software.

Our main findings are the following:

- Comprehensive and self-explanatory documentation is supplied with the codes `SRRIT`, `EB12`, `EB13`, and `ARPACK`.
- A particularly helpful feature of the documentation provided with `EB12` and `EB13` is that it includes simple sample programs. These sample programs would be of particular value to users who are unfamiliar with using reverse communication.

- EB12, EB13, ARNCHEB, and ARPACK 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. Both EB12 and EB13 are supplied with comprehensive testing programs: each comes with a driver that is designed to execute each line of the code at least once.

- ARNCHEB does not document all input and output parameters fully, and the code itself does not include comments to explain each of the parameters. Moreover, the code has no error flag and performs no error checking.

- The LOPSI documentation provides no assistance in the event of an error. LOPSI has an error flag that, on exit, indicates success or failure. If a failure is indicated, no help is given to the user as to what has gone wrong or what might be tried to achieve success. Furthermore, we found that the flag could be set to indicate all was well but when the computed eigenvalues were checked, they could be totally inaccurate. LOPSI does not check input parameters for errors.

- The codes SRRIT, EB12, EB13, and ARPACK all have error flags and check the parameters supplied by the user for errors. If an error is detected, EB12 and EB13 optionally print a message indicating what the error is. Both SRRIT and ARPACK set a flag and provide documentation for interpreting the flag.

- The codes ARNCHEB, EB13, SRRIT, and ARPACK 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, SRRIT, and ARPACK the monitoring printing is optional. EB12 offers the option of printing error and/or warning messages, but users can monitor convergence only by exploiting the reverse communication interface and displaying the information themselves.

- Our experiences suggest that LOPSI and ARNCHEB were not comprehensively tested. Both codes were found to contain bugs.[2]

[2]The authors of the respective codes were contacted with our findings.

- 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 the codes 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 codes as conforming to the `Fortran 77` standard.

5.6 Input Parameters

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, some of the codes require the user to decide which portion of the spectrum is to be computed.

Table 3: Input from the user

Code	Required Input
`LOPSI`	Matrix **A** in standard sparse format Maximum number of matrix-vector products Maximum number l matrix vector products between the oblique projections
`SRRIT`	Matrix-vector product routine Maximum number of iterations
`EB12`	None
`ARNCHEB`	Matrix-vector product routine Type of ellipse Degree of Chebychev polynomial Amount of orthogonalization Whether to perform deflation
`EB13`	Block size
`ARPACK`	Maximum number of iterations

Requiring the user to choose these parameters may appear reasonable

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

because the user is likely to know how many eigenvalues are required and how much accuracy is wanted. However, as discussed in Section 5.3, in order to select an appropriate value for ϵ, the user generally needs some knowledge of the problem, such as the norm of $\mathbf{A}$ 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. Some numerical results illustrating the effects of different choices for m may be found in Duff and Scott (1993).

All the codes, with the exception of `EB12`, require the user to supply values for at least one other input parameter. These parameters are listed in Table 3. Finite termination of the computation is ensured by specifying the maximum number of iterations and/or the maximum number of matrix-vector products. `EB12` and `EB13` avoid requiring the user to set these parameters by having default values that the user is able to reset.

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 `EB12` and `EB13` user friendly while at the same time providing flexibility.

6 Numerical Experiments

In this section and in the Appendix, results are presented for a range of problems arising from scientific and industrial applications. Our aims are to compare the performance of the software discussed in this paper and to acquire a better understanding of the practical behavior of the methods and of the importance of different implementation details.

6.1 The Test Matrices

Table 4: The matrices used for performance testing (∗ indicates matrix from the collection of Bai, Barratt, Day and Dongarra 1995a)

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 modeling
NNC1374	1374	8606	Nuclear reactor core modeling
WEST0156	156	371	Chemical engineering plant model
WEST0167	167	507	Chemical engineering plant model
WEST02021	2021	7353	Chemical engineering plant model
CK400∗	400	2860	Not available
CK656∗	656	3884	Not available
PDE2961∗	2961	14585	Model PDE eigenvalue problem
RW5151∗	5151	20199	Markov chain modeling: random walk
CDDE∗			2-D convection diffusion problem
TOLOSA∗			Stability of aircraft in flight
BW2000∗	2000	7996	Chemical engineering model

The matrices we have used to evaluate the performance of the software are taken either from the well-known Harwell-Boeing collection of sparse matrices (Duff, Grimes and Lewis, 1992) or from the collection of large eigenvalue problems of Bai, Barrett, Day and Dongarra (1995*a*). Many of the problems were chosen because they have appeared elsewhere in the literature on solving large sparse nonsymmetric eigenvalue problems (for example, Sadkane, 1993, uses the matrices GRE1107 and PORES3 when testing his block Arnoldi-Chebychev method and Saad, 1984, uses the random walk problem in his tests on Arnoldi-based methods). The Harwell-Boeing

matrices arise from linear systems of equations and are not nonsymmetric eigenvalue problems. Nevertheless, computing eigenvalues for some of the matrices in the collection can provide useful tests for the software. The problems in the collection of Bai et al. are ideal for our study because the primary purpose in developing the collection was to provide a testbed of practical problems for use in testing numerical algorithms for solving eigenproblems. However, this test set is still under development, and only part of the collection is currently available.

6.2 Verification

It is important when testing software that an attempt be made to check the correctness of the computed results. For example, an important consideration is whether any of the sought-after eigenvalues have been missed. In the symmetric case, a factorization is performed, and an inertia count then provides a check for missing eigenvalues (see Grimes et al., 1994, and Parlett, 1980, for details). There is no analogous procedure for nonsymmetric matrices.

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

$$FE_{max} = \max_{1 \le i \le r} \frac{|\lambda_i - \theta_i|}{|\lambda_i|}, \tag{6.1}$$

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 by using the QR algorithm.

We also check results by computing the r eigenvector residuals

$$\|\mathbf{A}\mathbf{y} - \theta\mathbf{y}\|_2 \tag{6.2}$$

and the real and imaginary portions of the Rayleigh quotient errors

$$\|\mathbf{y}^T\mathbf{A}\mathbf{y} - \theta\mathbf{y}^T\mathbf{y}\|_2. \tag{6.3}$$

`SRRIT` does not compute the eigenvectors of $\mathbf{A}$, but the code is designed so that it is straightforward to do this computation by using the LAPACK routine `DTREVC`. We have done this in our tests with `SRRIT`.

For all the codes except ARNCHEB and LOPSI, we check the orthogonality of the computed Schur basis and quality of the Schur projection by computing

$$\|\mathbf{X}_r^T\mathbf{X}_r - \mathbf{I}_r\|_\infty \quad \text{and} \quad \|\mathbf{X}_r^T\mathbf{A}\mathbf{X}_r - \mathbf{T}_r\|_\infty, \tag{6.4}$$

respectively.

The checks (6.2)–(6.4) are designed to test the backward stability of the software.

6.3 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 employs different stopping critera. Therefore, even if we supply each code with the same convergence tolerance and if 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} (u denotes the relative machine precision).

In designing their software, the authors have all attempted to produce codes that can be used as black boxes. It should be recognized that, in doing so, the authors 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 that have been made, we use only the default values (or values recommended by the authors in their documentation) for all parameters in our numerical experiments.

As discussed in Section 5.6, given the use of default parameters, there remain other parameters that must be chosen by the user. The choices made for these parameters can significantly affect the performance of the software. When testing the software on a particular problem, we use the same values of r and m for each code. This choice allows us to compare the relative performance of the codes.

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 the next two subsections we present detailed results for the TOLOSA matrix and the two-dimensional convection-diffusion problem (CDDE). Further results for other matrices in our test set are given in the Appendix and in the reports by Lehoucq and Scott (1995*a*, 1995*b*).

6.4 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.

Table 5: CPU times (in seconds) and matrix-vector products for the TOLOSA matrix of order 1000 ($*$ denotes that code did not converge within $4000m$ matrix-vector products)

	Subspace dimension m		
Algorithm	8	16	32
SRRIT	33/20488	141/64016	$*$
EB12	34/17719	120/35087	65/10271
ARNCHEB	1.0/1867	2.4/2583	7.3/4294
EB13	5.6/5545	0.8/625	8.2/3917
ARPACK	15/5120	4.5/1082	3.2/482

The code LOPSI was not used for this problem since the matrix is not stored explicitly. We employed each of the other codes discussed in Section 4 to compute the eigenvalue of largest imaginary part and the corresponding eigenvector of the TOLOSA matrix with orders up to 2000. Our findings for $n = 1000$ and $n = 2000$ are summarized in Tables 5 and 6.

We observe that, for this problem, the subspace iteration codes are much slower to converge than the Arnoldi codes. With $m = 16$ and $n = 1000$,

Table 6: CPU times (in seconds) and matrix-vector products for the TOLOSA matrix of order 2000 ($*$ denotes that code did not converge within $4000m$ matrix-vector products)

	Subspace Dimension m		
Algorithm	8	16	32
SRRIT	$*$	$*$	$*$
EB12	117/26255	72/9919	341/26111
ARNCHEB	5.0/4201	6.3/4100	15/5484
EB13	2.8/1451	6.4/2468	7.1/1657
ARPACK	16/2528	3.4/422	5.0/422

SRRIT found 16 eigenvalues with the requested accuracy. We also see that the value of m giving the best ARPACK results is larger than that giving the best ARNCHEB and EB13 reresult sults. This suggests that the figures given in Table 2 for the storage requirements of the different codes should be treated with caution. However, if m is sufficiently large, the number of matrix-vector products required by ARPACK is significantly less than the numbers used by the other codes.

6.5 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 discretized using centered finite differences. The eigenvalues and eigenvectors of the resulting matrix are known explicitly (see, for example, Bai et al., 1995*a*). 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 diagonalizable.
- 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. Again, `LOPSI` was not used for this problem because the matrix is not held explicitly. Tables 7 to 10 illustrate that `SRRIT` can be much slower to converge than `EB12` and may require an unacceptably large number of matrix-vector products. For the smaller values of ρ and n used in the tests, `EB12` can be competitive with the Arnoldi software. We also found that, as n was increased, `EB13` and `ARPACK` required much smaller convergence tolerances than did the subspace iteration codes if missing multiple eigenvalues were to be avoided. In each of our tests on this problem, `ARNCHEB` failed to compute the required eigenvalues with the requested accuracy (although it was successful when used to compute a single eigenpair).

Table 7: 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	18	36
`SRRIT`	83/29583	106/27465
`EB12`	9.6/5312	27/14450
`ARNCHEB`	†	†
`EB13`	7.2/2116	8.2/1358
`ARPACK`	8.7/639	9.1/532

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. `SRRIT` and `EB12` did not converge with the required accuracy within $4000m$ matrix-vector products. Using convergence tolerances of $\sqrt{u}$ and 10^3u, `ARPACK` ran without an error flag being set, but complex eigenvalues were returned. When 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 (6.2)–(6.4) were suitably small. With the same convergence tolerances, `EB13` returned real eigenvalues but missed the multiplicities.

Table 8: CPU times (in seconds) and matrix-vector products for the 2-D Laplacian ($\rho = 0$) matrix of order $10,000$. Note that with $m = 36$, SRRIT found 11 eigenvalues with the requested accuracy ($*$ denotes that code did not converge within $4000m$ matrix-vector products).

	Subspace Dimension m	
Algorithm	18	36
SRRIT	$*$	2326/92196
EB12	62/8631	153/20099
ARNCHEB	$*$	$*$
EB13	81/4781	115/4263
ARPACK	171/2625	79/1081

Table 9: CPU times (in seconds) and matrix-vector products for the CDDE matrix with $\rho = 10$ of order 2500. Note that with $m = 36$, SRRIT found 11 eigenvalues with the requested accuracy ($\dagger$ denotes that one or more of the requested eigenvalues was missed).

	Subspace Dimension m	
Algorithm	18	36
SRRIT	127/46098	332/88356
EB12	13/5971	33/9668
ARNCHEB	$\dagger$	$\dagger$
EB13	41/12178	46/3383
ARPACK	8.3/602	11/613

Table 10: 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	18	36
SRRIT	$*$	$*$
EB12	284/41857	292/34268
ARNCHEB	$*$	$*$
EB13	727/20004	436/7263
ARPACK	61/991	80/1095

6.6 General Findings

In this subsection we summarise our findings based on running the codes on all the test problems listed in Table 4.

First, our conclusions for the subspace iteration codes are :

- SRRIT can be much slower than EB12 and LOPSI. The reason appears to be that, although SRRIT uses only a small number of iterations, it generally performs many more reorthogonalizations than do the other codes. SRRIT also allows a large number of matrix-vector products to be performed between projections.

- An attractive feature of SRRIT is that it displays monotonic consistency; that is, as the convergence tolerance decreases, so does the size of the computed residuals.

- LOPSI can be the fastest subspace iteration code for computing a single eigenvalue, but if $r > 1$, it can return spurious eigenvalues. We believe there is probably a bug in the code, but we have not yet located the source of the problem.

- SRRIT has a useful property of being able to recognize clusters of eigenvalues. This can lead to the code's returning more eigenvalues than requested. Our experience suggests that some adjustment to the values of the (internal) parameters used in detecting a cluster may be beneficial.

- Use of a Chebychev polynomial may improve the performance of `EB12` but it can also degrade it. This cannot be anticipated unless some knowledge of the distribution of the eigenvalues is already known.
- All the subspace iteration codes are extremely slow when used to solve the most difficult test problems in our test set.

Our findings for the Arnoldi codes are the following:

- `ARPACK` is generally the fastest and most dependable of the codes studied, especially for small convergence tolerances and large departures from normality.
- Like `SRRIT`, `ARPACK` displays monotonic consistency.
- The code `EB13` offers many options and, as these options are fully documented, they enable the user to experiment with different choices.
- `ARNCHEB` gives reasonable results for computing a single eigenpair but it can struggle on problems for which several eigenvalues are requested.
- On some of the examples, `ARPACK` uses dramatically fewer matrix-vector products than the code. However, its restarting strategy can be more expensive. This is typically the case when the cost of a matrix-vector product is inexpensive.
- For some problems, `EB13`'s blocking option gives disappointing results and appears to need further work. The results given in Sadkane (1993) reflect a much larger subspace being constructed than was attempted in our study.

Comparing subspace iteration with Arnoldi's method and considering the suitability of the codes to solve industrial eigenproblems, we draw the following general conclusions:

- For many of the test problems, the best Arnoldi result is better than the best subspace iteration result.
- There is a tendency of the Arnoldi methods to miss multiple eigenvalues when the matrix is highly non-normal. The subspace iteration code `SRRIT` was generally the most reliable code for problems with clustered and/or multiple eigenvalues, but it could be unacceptably slow.

- None of the codes succeeds on *all* the test problems. In particular, all the codes experience difficulties for problem BWM2000. This problem is described in Saad (1992) and Bai et al. (1995*a*). The eigenvalues of practical importance are the right-most ones, but none of the codes studied successfully computed them. However, overall, we found `ARPACK` to be the most successful of the codes studied.

- All the codes are sensitive to the choice of input parameters, and they all leave important decisions, such as the choice of the subspace dimension m, to the user.

- The performance of the subspace iteration and Arnoldi methods is extremely dependent on implementation details within the software.

7 Conclusions

We end our study by briefly discussing features found in the current software that should be part of any revisions to existing software or should be incorporated in new attempts at high-quality software. Also addressed are new areas of research for improved algorithms and the construction of software implementing them.

- The grouping of clustered eigenvalues appears to be an important feature for problems with equal or nearly equal eigenvalues. This is attempted by `SRRIT` but needs further study.

- A well-engineered strategy for detecting stagnating convergence may prevent the code from performing unnecessary work. Although not always successful, `EB12` and `EB13` attempt to do this; further investigation is needed. More generally, as examined in Section 5.3, further research and testing need to be undertaken to improve the ways in which the software decides to terminate the computation.

- For codes implemented in `Fortran 77`, we recommend the use of reverse communication for carrying out matrix-vector products. However, perhaps the time has now come to consider a more modern programming language. This could remove the requirement for a reverse communication interface.

- Improved polynomial restarting methods are needed. The success of `ARNCHEB`, `ARPACK`, `EB12`, and `EB13` depends upon such strategies. The

latter two codes include 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.

- The ability to deflate a converged invariant subspace can substantially reduce the amount of work performed when more than one eigenpair is wanted. Deflation strategies may also improve the reliability of the Arnoldi-based software. The deflation strategies proposed by Lehoucq and Sorensen (1995) need to be implemented in `ARPACK` and their effects studied.

- Automatic verification procedures are sorely needed. At present, no software is available for determining whether *the* given eigenvalue problem was solved. Successful convergence implies only that *an* eigenvalue problem was solved. This situation is to be contrasted with the situation when $\mathbf{A}$ is symmetric. Some work on a possible approach in the nonsymmetric case has been done by Meerbergen, Spence and Roose (1994).

- The automated selection of software parameters is another area where research is needed. For example, given that the user requests r eigenvalues, the software should attempt to determine the appropriate size m of the subspace needed during each iteration. There could be advantages here in using, for example, the `Fortran 90` programming language, since it allows dynamic allocation of storage.

Given the results of this study, we believe that an implicitly restarted Arnoldi iteration appears a promising way forward.

8 Availability of the Codes and Tests

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

- The test matrices of Bai et al. are available by anonymous ftp to `ftp.ms.uky.edu` in the directory `pub/misc/bai/Collection`.

- 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`.

- `LOPSI` is available by anonymous ftp to `netlib.att.com` in the directory `netlib/toms` as the compressed `Fortran` file `570.Z`.
- `SRRIT` is available by anonymous ftp to `ftp.ms.uky.edu` in the directory `pub/misc/bai/SRRIT`.
- `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.
- `EB12` and `EB13` are 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.

Acknowledgments

The authors are grateful to Zhaojun Bai and David Day for supplying some of the test problems, to Thierry Braconnier for helpful discussions on `ARNCHEB`, and to Iain Duff and Danny Sorensen for their interest in 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. Progress in the numerical solution of the nonsymmetric eigenvalue problem. *Numerical Linear Algebra with Application*, **2**, 219–234, 1995.

Z. Bai and G. W. Stewart. SRRIT — A FORTRAN subroutine to calculate the dominant invariant subspace of a nonsymmetric matrix. Technical Report 2908, Department of Computer Science, University of Maryland, 1992. Submitted to *ACM Transactions on Mathematical Software*.

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

Z. Bai, D. Day, and Q. Ye. ABLE: An adaptive block Lanczos method for non-Hermitian eigenvalue problems. Research report Research Report 95-04, Department of Mathematics, University of Kentucky, May 1995*b*.

F. L. Bauer. Das Verfahren der Treppeniteration und verwandte Verfahren zur Losung algebraischer Eigenwertproblem. *Z. Angew. Mat. Phys.*, **8**, 214–235, 1957.

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.

M. Clint and A. Jennings. The evaluation of eigenvalues of real unsymmetric matrices by simultaneous iteration method. *Journal of the Institute of Mathematics and its Applications*, **8**, 111–121, 1971.

J. Cullum. Lanczos algorithms for large scale symmetric and nonsymmetric matrix eigenvalue problems. *In* J. D. Brown, M. T. Chu, D. C. Ellison and R. J. Plemmons, eds, 'Proceedings of the Cornelius Lanczos International Centenary Conference', pp. 11–31, SIAM, Philadelphia, PA., 1994.

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.

J. J. Dongarra and E. Grosse. Distribution of mathematical software via electronic mail. *Communications of the ACM*, **30**, 403–407, 1987.

I. S. Duff and J. A. Scott. Computing selected eigenvalues of large sparse unsymmetric matrices using subspace iteration. *ACM Transactions on Mathematical Software*, **19**(2), 137–159, June 1993.

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.

R. W. Freund, M. N. Gutknecht, and N. M. Nachtigal. An implementation of the look-ahead Lanczos algorithm for non-Hermitian matrices. *SIAM Journal on Scientific and Statistical Computing*, **14**(1), 137–158, 1993.

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.

Harwell Subroutine Library. *A catalogue of subroutines (release 12)*. Advanced Computing Department, Harwell Laboratory, Harwell, UK, 1996.

A. Jennings and W. J. Stewart. Simultaneous iteration for partial eigensolution of real matrices. *Journal of the Institute of Mathematics and its Applications*, **15**, 351–361, 1975.

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

L. Komzsik. *Numerical Methods Users Guide.* The MacNeal-Schwendler Corporation, Los Angeles, CA, 1995.

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 Arnoldi software for sparse nonsymmetric eigenvalue problems. Technical report, Rutherford Appleton Laboratory, Chilton, England, 1995*a*. In preparation.

R. B. Lehoucq and J. A. Scott. An evaluation of subspace iteration software for sparse nonsymmetric eigenvalue problems. Technical report, Rutherford Appleton Laboratory, Chilton, England, 1995*b*. In preparation.

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, and P. Vu. 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.

Karl Meerbergen and Dirk Roose. Preconditioners for computing rightmost eigenvalues of large sparse nonsymmetric matrices. Technical Report TW206, Katholieke Universitet Leuven, Leuven, Belguim, November 1994. Submitted to *IMA Journal Numerical Analysis.*

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. ALGORITHM 506: HQR3 and EXCHANG: Fortran subroutines for calculating and ordering the eigenvalues of a real upper Hessenberg matrix [F2]. *ACM Transactions on Mathematical Software*, **2**(3), 275–280, 1976*a*.

G. W. Stewart. Simultaneous iteration for computing invariant subspaces of non-Hermitian matrices. *Numerische Mathematik*, **25**, 123–136, 1976*b*.

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.

W.J. Stewart and A. Jennings. ALGORITHM 570: LOPSI a simultaneous iteration method for real matrices [F2]. *ACM Transactions on Mathematical Software*, **7**(2), 230–232, June 1981*a*.

W.J. Stewart and A. Jennings. A simultaneous iteration algorithm for real matrices. *ACM Transactions on Mathematical Software*, **7**(2), 184–198, June 1981*b*.

D. S. Watkins and L. Elsner. Convergence of algorithms of decomposition type for the eigenvalue problem. *Linear Algebra and Its Applications*, **143**, 19–47, 1991.

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

A Appendix: Tables of Results

In this section, the accuracy of the computed eigenvalues is at least $\sqrt{u}$. A superscript of the form $(k\lambda)$ indicates that the code actually found k $(k > r)$ eigenvalues.

Table A-1: CPU times (in seconds) and matrix-vector products for computing the eigenvalues of largest modulus of PORES2 ($\ddagger$ denotes spurious eigenvalues returned)

Algorithm	$r = 1, m = 8$	$r = 4, m = 20$
LOPSI	0.2/121	$\ddagger$
SRRIT	0.4/168	$2.6/750^{(8\lambda)}$
EB12	0.2/111	1.0/246
EB13	0.1/17	0.2/41
ARPACK	0.1/16	0.3/34

Table A-2: CPU times (in seconds) and matrix-vector products for computing the right-most eigenvalues of PORES2

Algorithm	$r = 1, m = 12$	$r = 4, m = 20$
EB12	0.6/423	9.1/2890
ARNCHEB	3.4/1401	4.7/1712
EB13	0.4/119	1.3/305
ARPACK	0.5/90	1.3/151

Table A-3: CPU times (in seconds) and matrix-vector products for computing the eigenvalues of largest modulus of PORES3

Algorithm	$r = 1, m = 8$	$r = 5, m = 20$
LOPSI	0.3/769	0.3/481
SRRIT	$0.4/648^{(4\lambda)}$	$1.3/1364^{(13\lambda)}$
EB12	0.6/1735	0.7/869
EB13	0.2/113	0.8/325
ARPACK	0.1/36	0.2/48

Table A-4: CPU times (in seconds) and matrix-vector products for computing the right-most eigenvalues of PORES3 ($*$ denotes convergence not reached within $2000m$ matrix-vector products)

Algorithm	$r = 1, m = 8$	$r = 4, m = 20$
EB12	$*$	$*$
ARNCHEB	$*$	$*$
EB13	$*$	$*$
ARPACK	$*$	38/11684

Table A-5: CPU times (in seconds) and matrix-vector products for computing eigenvalues of GRE1107 (the eigenvalues of largest modulus are also the right-most eigenvalues) ($\ddagger$ denotes spurious eigenvalues returned, and $*$ denotes convergence not reached within $2000m$ matrix-vector products)

Algorithm	$r = 1, m = 8$	$r = 5, m = 20$
LOPSI	1.4/1761	$\ddagger$
SRRIT	2.6/2032	4.5/2198
EB12	1.6/2159	4.3/4774
ARNCHEB	4.0/2204	$*$
EB13	1.1/465	2.5/1449
ARPACK	1.3/302	1.4/260

Table A-6: CPU times (in seconds) and matrix-vector products for computing eigenvalues of NNC1374 (the eigenvalues of largest modulus are also the right-most eigenvalues) († denotes one of the required eigenvalues was missed, and $\diamond$ denotes the code was not suitable because it was not designed to compute the requested portion of the spectrum)

Algorithm	$r = 1, m = 8$	$r = 5, m = 20$ (largest modulus)	$r = 5, m = 20$ (right-most)
LOPSI	3.7/3457	7.9/5211	$\diamond$
SRRIT	8.2/5128	12/5600	$\diamond$
EB12	1.4/1151	5.4/5755	4.2/1633
ARNCHEB	4.1/1758	$\diamond$	140/37935†
EB13	0.4/117	4.0/659	2.1/557
ARPACK	0.6/112	1.4/167	1.2/145

Table A-7: CPU times (in seconds) and matrix-vector products for computing the right-most eigenvalues of WEST2021 ($*$ denotes convergence not reached within $2000m$ matrix-vector products)

Algorithm	$r = 1, m = 8$	$r = 5, m = 20$
EB12	$*$	98/20930
ARNCHEB	8.6/3233	71/15921
EB13	17/4869	18/4149
ARPACK	3.7/401	2.1/167

Table A-8: CPU times (in seconds) and matrix-vector products for computing $r = 8$ eigenpairs of CK400 and CK656 with $m = 20$ (the eigenvalues of largest modulus are also the right-most eigenvalues) (‡ denotes spurious eigenvalues returned, and † denotes multiple eigenvalues were missed)

Algorithm	CK400	CK656
LOPSI	‡	‡
SRRIT	0.8/1008	1.3/994
EB12	0.8/943	1.6/1035
ARNCHEB	†	†
EB13	0.3/228	0.6/238
ARPACK	0.2/61	0.3/70

Table A-9: CPU times (in seconds) and matrix-vector products for computing the right-most eigenvalues of PDE2961

Algorithm	$r = 1, m = 8$	$r = 6, m = 30$
EB12	2.1/791	17/5225
ARNCHEB	13/3161	14/3357
EB13	0.8/119	3.9/405
ARPACK	1.3/104	3.0/137