RAL-TR-96-022

An evaluation of subspace iteration software for sparse nonsymmetric eigenproblems.*

by

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

Abstract

During the last decade there has been a rise in interest in numerical methods for computing eigenvalues (and eigenvectors) of large sparse nonsymmetric matrices. The research effort is now being accompanied by the development of high-quality mathematical software. One of the methods which has received attention is that of subspace iteration and several software packages implementing subspace iteration algorithms have become available. In this report, as part of an extensive study to evaluate state-of-the-art software for the sparse nonsymmetric eigenproblem, we review subspace iteration software. We look at the key features of the software, the main differences between the packages, and their ease of use. Then, using a wide range of test matrices arising from 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.

---

* 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

Consider the standard eigenvalue problem

$$\mathbf{Az} = \lambda\mathbf{z} \tag{1.1}$$

where $\mathbf{A}$ is a large sparse real nonsymmetric matrix. During the last decade there has been a considerable increase in interest in computing selected eigenvalues and eigenvectors of (1.1). One of the methods of solution that has received attention is the method of subspace (or simultaneous) iteration (see, for example, Saad, 1984, Duff and Scott, 1993, and Bai and Stewart, 1992). 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 generalisation of the power method. Several packages which employ subspace iteration techniques have been developed. In additional to numerous research codes and implementations of subspace iteration embedded within engineering packages, there are currently (to the authors' knowledge) three subspace iteration packages which are available either in the public domain or under licence. These are

- `LOPSI` (Stewart and Jennings, 1981*a*, 1981*b*)
- the Harwell Subroutine Library routine `EB12` (Duff and Scott, 1993)
- `SRRIT` (Stewart, 1978 and Bai and Stewart, 1992)

Details of the availability of the codes is given in Section 7.

The reports and papers which accompany each of these codes provide some numerical results illustrating their use but a only small number of test problems is generally used and almost no results comparing the performance of the different codes have been reported. In this study we review, compare, and evaluate the subspace iteration software by considering the algorithms implemented by the software, examining the main differences between the codes, looking at the user interfaces and documentation. By running the codes on a set of test problems, we compare their performance and assess their reliability and robustness. We also illustrate the limitations of current subspace iteration codes and highlight problems for which more sophisticated software and software employing other numerical techniques are needed. An evaluation of Arnoldi-based software is the subject of a separate report (Lehoucq and Scott, 1996*a*) and subspace iteration software is compared with Arnoldi-based software in Lehoucq and Scott (1996*b*).

This report is organised as follows. We briefly review subspace iteration in Section 2 then in Section 3 we look at the subspace iteration software which is currently available. We outline the algorithms used and discuss the main features of each of the codes. We highlight 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 our test matrices. In Section 6 we summarise our findings.

We end this section by introducing the notation which is used throughout this report.

- n denotes the order of $\mathbf{A}$.
- 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 subspace iteration algorithm.
- $\mathbf{X}_m$ denotes the matrix representation of this subspace.
- $\mathbf{X}_m^{(k)}$ denotes the matrix representation of this subspace on the kth iteration ($k \geq 0$).The superscript is dropped when doing so causes no confusion.
- $(\mathbf{s}, \theta)$ 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 the relative machine precision (that is, the smallest machine number such that $1 + u > 1$).
- ϵ denotes the user-prescribed convergence tolerance.

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

2 Subspace iteration

We briefly recall the main ideas behind the subspace iteration algorithm for computing the dominant eigenvalues of $\mathbf{A}$. Starting with an initial $n \times m$ matrix $\mathbf{X}_m^{(0)}$ with independent columns $x_1, x_2, \ldots, x_m$ (called the 'trial' vectors), the subspace iteration method computes the matrix $\mathbf{X}_m^{(k)} = \mathbf{A}^k\mathbf{X}_m^{(0)}$. If on each iteration k the columns of $\mathbf{X}_m^{(k)}$ are normalised so that the largest component of each column is unity, in general, as k increases, the columns will each converge to the eigenvector corresponding to the dominant eigenvalue of $\mathbf{A}$. The idea of Bauer (1957) was to reestablish the linear independence of the columns by using (for example) the QR algorithm. Thus, in its simplest form the subspace iteration algorithm is as follows:

1. *Start:* Choose an initial set of normalised vectors $\mathbf{X}_m = [x_1, x_2, \ldots, x_m]$.

2. *Iteration:* Until convergence **do**
   (a) Compute $\mathbf{X}_m \leftarrow \mathbf{A}\mathbf{X}_m$.
   (b) Form the QR factorisation $\mathbf{X}_m = \mathbf{Q}\mathbf{R}$ and set $\mathbf{X}_m \leftarrow \mathbf{Q}$.

The orthogonalisation of $\mathbf{X}_m$ maintains linear independence among its columns. Overheads can be reduced by only orthogonalising $\mathbf{X}_m$ when there is reason to believe that some columns have become linearly dependent. Thus, in practice, $\mathbf{A}\mathbf{X}_m$ is replaced by $\mathbf{A}^l\mathbf{X}_m$ for some $l \geq 1$ ($l = l(k)$, where k is the iteration number).

Assuming the eigenvalues are ordered so that

$$|\lambda_1| \geq |\lambda_2| \geq \ldots \geq |\lambda_n|,$$

it can be shown that if $|\lambda_m| > |\lambda_{m+1}|$, then under mild restrictions on the initial set of vectors, 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 a convergence ratio for the ith eigenvalue of $max(|\lambda_i/\lambda_{i-1}|, |\lambda_{i+1}/\lambda_i|)$, which can be intolerably slow. Efficiency can be enhanced by combining subspace iteration with a projection method. Stewart (1978) uses an orthogonal projection method while Stewart and Jennings (1981*b*) use an oblique projection method. For these methods convergence is still linear but, if $|\lambda_{i-1}| > |\lambda_i| > |\lambda_{i+1}|$, the convergence ratio for the ith column of $\mathbf{X}_m$ is $|\lambda_{m+1}/\lambda_i|$. Thus the first columns of $\mathbf{X}_m$ generally converge more rapidly than the later ones and, even if only one eigenvalue is wanted, m should be chosen larger than 1 to attain reasonable convergence.

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

3 Subspace iteration software

In this section we briefly review the software packages LOPSI, SRRIT, and EB12, which implement subspace iteration methods for nonsymmetric matrices.

3.1 LOPSI

The code LOPSI of Stewart and Jennings (1981*a*), which is based on work done in 1970's 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 'lop-sided' oblique projection to compute the r eigenvalues of largest modulus and the corresponding eigenvectors of $\mathbf{A}$. The algorithm used by LOPSI, which is described in Stewart and Jennings (1981*b*), has the following general structure.

1. *Start:* Choose the subspace dimension m and an $n \times m$ matrix $\mathbf{X}_m$ with orthonormal columns. Choose $l_{max} \geq 1$ and set $l = max(l_{max}/2, 1)$.

2. *Iteration:* Compute $\mathbf{V} = \mathbf{A}^l\mathbf{X}_m$.

3. *Projection:*
   (a) Compute $\mathbf{G} = \mathbf{X}_m^T\mathbf{X}_m$, $\mathbf{H} = \mathbf{X}_m^T\mathbf{V}$.

(b) Solve $\mathbf{GB} = \mathbf{H}$ by forming the Cholesky factorisation of $\mathbf{G}$.
(c) Find the eigenvalues and eigenvectors of $\mathbf{B}$ using the `QR` algorithm. Sort the eigenvalues and corresponding eigenvectors so that the eigenvalues are in descending order of their absolute values.
(d) Set $\mathbf{X}_m \leftarrow \mathbf{VZ}$, where $\mathbf{Z}$ is the matrix whose columns are the ordered eigenvectors of $\mathbf{B}$. Normalise the columns of $\mathbf{X}_m$.

4. *Convergence test:* If the first r columns of $\mathbf{X}_m$ satisfy the convergence criteria then **stop**, else determine l for the next iteration and **go to 2**.

The convergence criteria are discussed in Section 4.5.

After the first iteration, the number l of premultiplications between projections is chosen so that the dominant eigenvector components do not "swamp" the predictions for the lower eigenvectors and also to avoid unnecessary work being performed by carrying the iteration beyond the stage at which convergence is achieved. A 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.

The user is allowed to supply an initial estimate for one or more of the columns of $\mathbf{X}_m$. If the user supplies less than m columns, the remaining columns are generated using a random number generator.

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. This means that no further computations are carried out with this vector. Initially, the convergence test is applied to column 1 of $\mathbf{X}_m$ and, only if it satisfies the convergence criteria, is column 2 tested.

Matrix products $\mathbf{AX}_m$ are performed within `LOPSI` by an internal subroutine `PREMULT`. The matrix $\mathbf{A}$ must be passed to `LOPSI` 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 products which may be performed.

3.2 SRRIT

The recent code `SRRIT` of Bai and Stewart (1992) is a revised and updated version of a code by Stewart (1978) of the same name. It uses subspace iteration with an occasional application of a 'Schur-Rayleigh-Ritz step' (from which the code derives its name).The algorithm used by `SRRIT` can be summarised as follows.

1. *Start:* Choose the subspace dimension m and an $n \times m$ matrix $\mathbf{X}_m$ with orthonormal columns.

2. *SRR step:*
Compute $\mathbf{B} = \mathbf{X}_m^T \mathbf{A} \mathbf{X}_m$.
Reduce $\mathbf{B}$ to the real Schur form $\mathbf{T}_m = \mathbf{Z}^T \mathbf{B} \mathbf{Z}$, where each diagonal block

$(\mathbf{T}_m)_{ii}$ is either of order 1 or is a 2×2 matrix having complex conjugate eigenvalues, with the eigenvalues in descending order of their absolute values along the diagonal blocks. Set $\mathbf{X}_m \leftarrow \mathbf{X}_m\mathbf{Z}$.

3. *Convergence test:* If the first r columns of $\mathbf{X}_m$ satisfy the convergence criteria then accept the first r eigenvalues of $\mathbf{T}_m$ as the sought-after eigenvalues of $\mathbf{A}$ and **stop** else determine the interval l between orthogonalisations and the number k of orthogonalisations before the next SRR step.

4. *Iteration:*
   for $i = 1, \ldots, k$ **do**

   Compute $\mathbf{X}_m \leftarrow \mathbf{A}^l\mathbf{X}_m$
   Orthonormalise the columns of $\mathbf{X}_m$

   end do
   Go to 2.

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 $\mathbf{X}_m$. If $\mathbf{X}_m$ is supplied, the user sets a parameter to indicate whether or not the columns of $\mathbf{X}_m$ are orthonormal. This offers a potential saving in work. If $\mathbf{X}_m$ is not supplied, the initial basis is generated using a random number generator.

`SRRIT` requires the user to supply a subroutine `ATQ` to perform matrix products $\mathbf{A}\mathbf{X}_m$. 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 needed.

The columns of $\mathbf{X}_m$ are orthonormalised using the modified Gram-Schmidt algorithm with reorthogonalisation. An internal parameter controls the maximum number of reorthogonalisations that may be performed; this is set to 5. At each iteration the code computes the interval l between orthogonalisations and the number k of orthogonalisations before the next SRR step; no upper limits are imposed on these quantities. l is chosen so that the columns of $\mathbf{A}^l\mathbf{X}_m$ remain linearly independent and k is chosen to try and ensure an SRR step is only performed when it is anticipated that one or more of the columns of $\mathbf{X}_m$ will have converged.

3.3 EB12

Of the packages considered in this study, the Harwell Subroutine Library code `EB12` (Duff and Scott, 1993) is the most general since it is designed to calculate either the right-most or the left-most eigenvalues of $\mathbf{A}$, or the eigenvalues which are of largest modulus. In many practical applications the eigenvalues with the largest real parts (that is, the right-most eigenvalues) are the ones of importance. For example, bifurcation problems commonly involve computing the eigenvalue λ of largest real part of a stability matrix and then detecting when $Re(\lambda)$ crosses the imaginary axis as a parameter to the stability matrix varies (see, for example, the problem studied by Garratt, Moore and Spence, 1991).

`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 $p_l(\mathbf{A})$. Suppose the eigenvalues of $\mathbf{A}$ are ordered so that the right-most ones are $\lambda_1, \lambda_2, \ldots, \lambda_r$. Let

$E(d,c,a)$ denote an ellipse with centre d, foci $d \pm c$, major semi-axis a, and which is symmetric with respect to the real axis (since $\mathbf{A}$ is real, its spectrum is symmetric with respect to the real axis). Suppose $E(d,c,a)$ contains the set S of unwanted eigenvalues. `EB12` uses $p_l(\lambda)$ given by

$$p_l(\lambda) = \frac{T_l[(\lambda - d)/c]}{T_l[(\lambda_r - d)/c]}, \tag{3.2}$$

where $T_l(\lambda)$ is the Chebychev polynomial of degree l of the first kind. This choice is made since the maximum modulus of $p_l(\lambda)$ within the ellipse is small compared to its modulus on the sought-after eigenvalues. The use of Chebychev polynomials with subspace iteration was proposed by Saad (1984).

The algorithm used by `EB12` is very similar to that outlined in Section 3.2 and is as follows.

1. *Start:* Choose the subspace dimension m and an $n \times m$ matrix $\mathbf{X}_m$ with orthonormal columns.

2. *SRR step:*
   Compute $\mathbf{B} = \mathbf{X}_m^T \mathbf{A} \mathbf{X}_m$.
   Reduce $\mathbf{B}$ to the real Schur form $\mathbf{T}_m = \mathbf{Z}^T \mathbf{B} \mathbf{Z}$, where each diagonal block $(\mathbf{T}_m)_{ii}$ is either of order 1 or is a 2×2 matrix having complex conjugate eigenvalues, with the eigenvalues ordered along the diagonal blocks. Set $\mathbf{X}_m \leftarrow \mathbf{X}_m \mathbf{Z}$.

3. *Convergence test:* If the first r columns of $\mathbf{X}_m$ satisfy the convergence criteria then accept the first r eigenvalues of $\mathbf{T}_m$ as the sought-after eigenvalues of $\mathbf{A}$ and **stop** else determine the iteration polynomial $p_l(\lambda)$ for the next iteration.

4. *Iteration:* Compute $\mathbf{X}_m \leftarrow p_l(\mathbf{A})\mathbf{X}_m$. Orthonormalise the columns of $\mathbf{X}_m$ and **go to 2**.

If the eigenvalues of largest modulus are required, `EB12` sets $p_l(\lambda) = \lambda^l$, otherwise p_l is the polynomial defined by (3.2) and the method is known as subspace iteration with Chebychev acceleration (Saad, 1984 and Duff and Scott, 1993) or as Chebychev subspace iteration (Meerbergen and Roose, 1994).

The evaluation of $p_l(\mathbf{A})\mathbf{X}_m$ with p_l defined by (3.2) is carried out using the three-term recurrence relation for Chebychev polynomials (see Saad, 1984 for details). Since λ_r is not known, in practice it is replaced by an approximation γ, which is updated at each iteration. It can be shown (see, for example, Saad, 1984) that the convergence ratio for the ith basis vector using (3.2) is η^l where

$$\eta = \frac{a + [a^2 - 1]^{\frac{1}{2}}}{a_i + [{a_i}^2 - 1]^{\frac{1}{2}}},$$

where $E(d,c,a_i)$ is the ellipse with centre d, foci $d \pm c$, and major semi-axis a_i which passes through λ_i. Thus, it is the 'gap' between $E(d,c,a)$ and $E(d,c,a_i)$ which determines how fast λ_i converges.

In EB12, the columns of $\mathbf{X}_m$ are orthonormalised using the modified Gram-Schmidt algorithm. On each iteration, the degree l of the iteration polynomial is chosen to try and ensure 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 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. If they are not supplied by the user, a starting basis is obtained by performing a single step of Arnoldi's method.

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. This allows 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 4.2.

Once EB12 has successfully computed the required eigenvalues of $\mathbf{A}$, the user may call a separate subroutine EB12B to compute the corresponding eigenvectors. EB12B computes the eigenvectors $\mathbf{s}$ of the quasi-triangular matrix $\mathbf{T}_m$ using back-substitution and then takes the approximate eigenvectors of $\mathbf{A}$ to be $\mathbf{y} = \mathbf{X}_m\mathbf{s}$. The computed eigenvectors $\mathbf{y}$ are normalised. EB12B optionally computes the scaled eigenvector residuals

$$\frac{\|\mathbf{A}\mathbf{y} - \theta\mathbf{y}\|_2}{\|\mathbf{A}\mathbf{y}\|_2}.$$

4 Key differences between the codes

Although LOPSI, EB12, and SRRIT are all subspace iteration-based codes written in FORTRAN 77, it is clear from the above descriptions that the codes differ from one another in a number of important ways. In this section we consider some of these differences.

4.1 Use of Schur vectors

A principal difference between the LOPSI algorithm and those used by EB12 and SRRIT is that LOPSI uses an oblique projection method rather than an orthogonal projection method and, in so doing, avoids any vector orthogonalisation. In addition, LOPSI computes approximations to eigenvectors of $\mathbf{A}$, whereas EB12 and SRRIT, by working with the Schur vectors, compute an approximation to the invariant subspace of $\mathbf{A}$ corresponding to the sought-after eigenvalues. According to Stewart (1976*a*), it is more stable to work with the Schur vectors since if $\mathbf{A}$ is defective, it may not have enough eigenvectors to span the subspace corresponding to the sought-after eigenvalues. Even when $\mathbf{A}$ is not defective, its eigenvectors may be nearly dependent so that they form a poor numerical basis for this subspace. EB12 offers the user the option of computing the eigenvectors once the eigenvalues

have converged. In their paper, Bai and Stewart (1992) recommend that, if the eigenvectors are desired, they may be obtained by first using the program DTREVC in LAPACK (Anderson, Bai, Bischof, Demmel, Dongarra, Croz, Greenbaum, Hammarling, McKenney, Ostrouchov and Sorensen, 1992) to compute eigenvectors $\mathbf{s}$ of the quasi-triangular matrix $\mathbf{T}_m$ and then forming $\mathbf{X}_m\mathbf{s}$.

4.2 Matrix products AX

A major implementation difference between the codes is the way in which they cope with forming the product of $\mathbf{A}$ with sets of vectors. If the order n of $\mathbf{A}$ is large, this can represent the dominant cost. Thus, it is important to minimise the number of times that $\mathbf{A}$ is applied and to ensure its application is implemented efficiently.

The code LOPSI is the most restrictive of the codes in our study since it requires the user to pass the matrix to the routine using a defined storage scheme. The matrix products $\mathbf{AX}$ 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 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 would involve considerable effort and no documentation is provided to assist with this.

SRRIT adopts a somewhat more flexible approach by requiring the user to supply the subroutine to perform the matrix 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 SRRIT uses the FORTRAN 77 programming language, the number of subroutine arguments is fixed. If a user needs additional descriptors to perform matrix products, these must be passed through a COMMON block.

The reverse communication approach followed by EB12 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, full advantage may be taken of parallelism and/or vectorisation. Another obvious advantage of reverse communication is that the user is able 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 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 products $\mathbf{W} = \mathbf{AX}$. 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.

Another advantage of reverse communication is that, in the case of unacceptably slow convergence, EB12 is able to offer the option of restarting the computation with, for example, an increased subspace dimension but taking advantage of the approximations to the sought-after eigenvalues and Schur vectors already computed.

4.3 Orthogonalisation of sets of vectors

The code LOPSI avoids the need to orthogonalise sets of vectors but one of the important implementation differences between SRRIT and EB12 concerns the orthogonalisation of the columns of $\mathbf{X}_m$, how frequently this is done, and how often an SRR step is performed. As already mentioned, SRRIT employs the modified Gram-Schmidt algorithm with reorthogonalisation and, in general, performs more than one orthogonalisation between SRR steps. EB12, however, adopts the somewhat different approach of attempting to keep the columns of $\mathbf{X}_m$ linearly independent by limiting the degree l of the iteration polynomial and using the modified Gram-Schmidt algorithm without reorthogonalisation. In the Harwell Subroutine Library Release 11 version of EB12, only one orthogonalisation was performed between each SRR step. In the latest Release 12 version, experience with practical problems has led to the code being modified slightly. Björck (1967) showed that the modified Gram-Schmidt algorithm applied to a matrix $\mathbf{X}$ produces a computed matrix $\hat{\mathbf{X}}$ which satisfies

$$\hat{\mathbf{X}}^T\hat{\mathbf{X}} = \mathbf{I} + \mathbf{E}, \quad \|\mathbf{E}\|_2 \approx \mathbf{u}\kappa_2(\mathbf{X}), \tag{4.3}$$

where $\kappa_2(\mathbf{X})$ is the condition number of $\mathbf{X}$ and u is the relative machine precision. On iteration k, EB12 uses (4.3) to obtain an estimate of $\kappa_2(\mathbf{X}_m)$ and this in turn is used to restrict the degree $l(k+1)$ of the iteration polynomial for the next iteration. If either $\kappa_2(\mathbf{X}_m) > 10^3$ or $l(k+1) \geq q$, the code immediately moves to performing an SRR step. In the Release 11 version, q is equal to 1. This tends to result in an SRR step being performed too early, that is, before any of the columns of $\mathbf{X}_m$ have converged. In the Release 12 version, q is set to 10 and, if the degree $l(k+1)$ is less than 10, the columns of $\mathbf{X}_m$ are reorthogonalised, $\mathbf{X}_m \leftarrow p_{l(k+1)}(\mathbf{A})\mathbf{X}_m$ is computed, its columns orthogonalised, and $\kappa_2(\mathbf{X}_m)$ recomputed. The process is repeated, if necessary, until at least q matrix products $\mathbf{A}\mathbf{X}_m$ have been performed on iteration k. Thus, step 4 of the EB12 algorithm given in Section 3.3 becomes
Set $q = 0$
while $q < 10$ **do**

$q = q + l(k+1)$
Compute $\mathbf{X}_m \leftarrow p_{l(k+1)}(\mathbf{A})\mathbf{X}_m$
Orthonormalise the columns of $\mathbf{X}_m$
Compute $\kappa_2(\mathbf{X}_m)$ and update $l(k+1)$

end do

This delaying of the SRR step can give significant improvements in performance. This is illustrated in Table 4.1. The results in this table are for a convergence tolerance $\epsilon = u * 10^3$ and the runs were performed on a SUN SPARCstation 10. The number of iterations is the number of SRR steps. All the examples in Table 4.1 are taken from the Harwell-Boeing collection (Duff, Grimes and Lewis, 1992). In Table 4.1, a † indicates the requested accuracy was not achieved before the stopping criteria terminated the computation (see Section 4.5).

Identifier	r	m	Matrix-vector products hsl11	Matrix-vector products hsl12	Iterations hsl11	Iterations hsl12	CPU time hsl11	CPU time hsl12
PORES3	8	16	3127	2827	12	9	7.25	6.78
GRE1107	1	8	3135	3135	13	13	12.93	12.93
IMPCOLA	8	16	1745	1653	33	12	3.59†	2.52
IMPCOLB	8	16	3095	3569	62	25	3.31†	2.75
IMPCOLC	8	16	1471	1569	26	10	1.74†	1.13
NNC666‡	8	16	3518	3263	24	19	13.71	12.95
WEST0156‡	4	10	4365	1709	217	15	8.46†	2.62
WEST0167	8	24	659	461	16	9	2.56	1.74

Table 4.1: A comparison of Release 11 and Release 12 versions of the Harwell Subroutine Library routine `EB12` (hsl11 denotes Release 11 and hsl12 denotes Release 12). † indicates the requested accuracy not achieved. ‡ indicates Chebychev acceleration is used.

4.4 The use of BLAS

Apart from the matrix products $\mathbf{AX}_m$, the subspace iteration algorithm requires 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 make use of the BLAS to very different degrees. When `LOPSI` was developed, only the Level 1 BLAS were available and `LOPSI`, in fact, makes no use of these kernels.

`EB12` makes some use of Level 1 and Level 2 BLAS kernels, as well as using the Level 3 kernel `GEMM` for performing matrix-matrix multiplications of the form $\mathbf{B} = \mathbf{X}^T\mathbf{W}$, where $\mathbf{W} = \mathbf{AX}$ has been computed by the user. In the SRR step, `EB12` uses modified versions of the EISPACK routines `ORTHES` and `ORTRAN` (Smith, Boyle, Garbow, Ikebe, Klema and Moler, 1976) and of the routine HQR3 given by Stewart (1976*b*). Since the release of `EB12` in 1991, LAPACK (Anderson et al., 1992) has become available. LAPACK was designed to supersede EISPACK, by developing new routines and by restructuring the EISPACK software to achieve much greater efficiency, where possible, on modern high-performance computers. To help do this, LAPACK routines are written so that as much as possible of the computation is performed by calls to the BLAS. The updating of the original 1978 version of `SRRIT` included using BLAS kernels and replacing EISPACK routines with LAPACK routines. Therefore, of the three codes, `SRRIT` makes the most use of the BLAS and we anticipate that this will be reflected in the performance of `SRRIT`. The only part of the codes where `EB12` makes more use than `SRRIT` of the higher level BLAS is in orthonormalising the columns of $\mathbf{X}_m$. `SRRIT` uses a Level 1 BLAS implementation of the modified Gram-Schmidt algorithm with reorthogonalisation;

EB12 has a Level 2 implementation of the modified Gram-Schmidt algorithm.

4.5 The stopping criteria

The codes all use different stopping criteria, which adds to the difficulties associated with trying to compare their performance (see Section 5). Useful 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.

For subspace iteration, the dominant eigenvalues converge most rapidly so the $(j+1)$-st eigenvalue needs to be tested only once 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 and EB12 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 which 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 criteria used by EB12 for the jth column of $\mathbf{X}_m$ requires

$$\|(\mathbf{A}\mathbf{X}_m - \mathbf{X}_m\mathbf{T}_m)_j\|_2 \quad < \quad \|(\mathbf{A}\mathbf{X}_m)_j\|_2\epsilon. \tag{4.4}$$

To save work, 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 and obtain the requested accuracy and facilities are included for restarting the computation from the point at which the warning was issued.

We remark that recent work by Chatelin and Fraysée (1993) recommends that stopping criteria for iterative methods should be based on the backward error. The idea of backward error is to measure the shortest distance between the original problem with computed solution $\mathbf{z}$ and a perturbed problem with exact solution $\mathbf{z}$. The normwise backward error associated with $(\mathbf{T}_m, \mathbf{X}_m)$ is defined by

$$\eta = min\{\delta > 0 : \|\Delta\mathbf{A}\| \leq \delta\|\mathbf{A}\|, (\mathbf{A} + \Delta\mathbf{A})\mathbf{X}_m = \mathbf{T}_m\}. \tag{4.5}$$

In the 2-norm, it can be shown that

$$\eta = \frac{\|\mathbf{A}\mathbf{X}_m - \mathbf{X}_m\mathbf{T}_m\|_2}{\|\mathbf{A}\|_2}. \tag{4.6}$$

Computing $\|\mathbf{A}\|_2$ (or a bound on $\|\mathbf{A}\|_2$) is often very expensive. Since $\|(\mathbf{A}\mathbf{X}_m)_j\|_2 \leq ||\mathbf{A}||_2$ and $|\theta| \leq ||\mathbf{A}||_2$, the stopping criteria used by `SRRIT` and `EB12` are related to the backward error.

4.6 Storage requirements

For large problems, the amount of storage needed can be an important consideration when selecting which code to use. In Table 4.2, we compare the storage requirements of the three codes. We observe that `LOPSI` requires the most storage and that `EB12` needs 3 real arrays of length nm while `SRRIT` requires only 2 such arrays. There are two reasons why `EB12` demands an extra array. Firstly, to use the 3-term recurrence relation for Chebychev polynomials to compute $p_l(\mathbf{A})\mathbf{X}_m$, three arrays of length nm are needed. Secondly, as already discussed, `EB12` uses reverse communication. To try and ensure against the user overwriting the latest approximation $\mathbf{X}_m$ to the Schur vectors, the user forms matrix products 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, `EB12` demands 3 arrays of length nm.

Code	Storage
`LOPSI`	$3nm + 4m^2 + O(m)$
`SRRIT`	$2nm + 2m^2 + O(m)$
`EB12`	$3nm + 2m^2 + O(m)$

Table 4.2: Storage requirements for the subspace iteration software.

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 experience.

- Each of the codes came with a well-commented parameter list which allowed us to use them on straightforward problems without any difficulties.
- A particularly helpful feature of the documentation provided with `EB12` was that it included a simple sample program. This would be of particular value to users who are unfamiliar with using reverse communication.

- Each of the codes includes a parameter in its argument list which is employed as an error flag. On exit from `LOPSI`, the error flag should have the value 0 if the requested number of eigenvalues have satisfied the convergence criteria and the value 1 otherwise. `LOPSI` does not provide any useful information in the event of a failure. A warning message is issued if the size of a pivot was increased to maintain stability during a Cholesky decomposition but the significance of this warning (which we found could be issued many times during the computation) is not adequately explained. Furthermore, we found the error flag could be set to indicate all was well but when the computed eigenvalues were checked, they could be totally inaccurate. We conclude that the code has not been comprehensively tested. [1]

- `LOPSI` does not check input parameters for errors. `SRRIT` and `EB12` check the user's data and, in the event of an error, set the error flag and return control to the user. The documentation explains the meaning of the possible different values for the error flag. In addition, `EB12` optionally prints a self-explanatory error message.

- During the computation, `SRRIT` and `EB12` monitor for fatal errors and terminate the computation if such an error is encountered. Again, `EB12` optionally prints an error message. `EB12` also warns the user if convergence has been judged to be unacceptably slow and returns the convergence tolerance which the computed Schur vectors actually satisfy.

- A nice feature of `SRRIT` is an input parameter which can be set to allow information on the course of the computation to be printed. After each SRR step, this printing includes the values of the sought-after eigenvalues and the norm of the corresponding residuals. This allows the user to see how fast the eigenvalues are converging but, as reverse communication is not used, the user is unable to take action to terminate or restart the computation if convergence is not proving satisfactory. It is confusing that the parameter which must be set if printing is required doubles-up as the error flag. The current version of `EB12` does not offer the user the option of printing on each iteration: the user must take advantage of reverse communication if printing is required.

- The documentation for `SRRIT` points to parameters within the code which are given values at the start of the computation. It is suggested that the knowledgeable user may wish to alter these values to improve efficiency for particular problems. This is not a friendly interface. If these really are control parameters which the user may want to alter, it would be advisable to give them default values by calling an initialisation subroutine and then passing them as input arguments to the main routine. The user could alter one or more of the control parameters after the call to the initialisation routine and prior to the call to the main routine.

- At the end of the computation, `SRRIT` and `EB12` return some useful information, including the number of matrix products performed. For our numerical

[1] The author was contacted with our findings.

experiments, it was necessary to modify LOPSI so that information needed for comparing the performance of the codes could be collected.

- When checked with a FORTRAN code analyser[2], each code was passed as conforming to the FORTRAN 77 Standard.

To summarise, LOPSI is easy to use if **A** is held in the required form and if no difficulties are experienced. However, the user is given no help if **A** is supplied in any other form, nor if the method fails to converge. SRRIT offers printing for monitoring progress but offers only limited assistance if a non-zero error flag is returned. EB12 provides error messages and help if an error is encountered but could be improved by offering printing at each iteration.

5 Numerical experiments

In this section we present the results of using the subspace iteration software to compute a few eigenvalues and eigenvectors of a range of problems arising from real scientific and industrial applications.

5.1 The test matrices

The test problems are taken either from the well-known Harwell-Boeing collection of sparse matrices (Duff et al., 1992) or from the collection of large eigenvalue problems of Bai, Barrett, Day and Dongarra (1995). Many of the problems we chose 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). We employ the same set of test problems in our evaluation of Arnoldi based software (Lehoucq and Scott, 1996*a*). We remark that 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 since the primary purpose in developing the collection was to provide a testbed of practical problems for use in testing numerical algorithms for solving eigenvalue problems. However, this test set is still under development and the whole set is not yet available.

The CDDE problem 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 Bai et al., 1995). 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

[2]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.

Table 5.3: The matrices used for performance testing (* indicates matrix from the collection of Bai, Barratt, Day and Dongarra, 1995).

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.

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 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, Lewis and Simon, 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

$$FE_{max} = \max_{1 \le i \le r} \frac{|\lambda_i - \theta_i|}{|\lambda_i|}, \tag{5.7}$$

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

$$\|\mathbf{A}\mathbf{y} - \theta\mathbf{y}\|_2, \tag{5.8}$$

and the real and imaginary portions of the Rayleigh Quotient errors

$$\|\mathbf{y}^T\mathbf{A}\mathbf{y} - \theta\mathbf{y}^T\mathbf{y}\|_2. \tag{5.9}$$

As mentioned in Section 4.1, `SRRIT` does not compute the eigenvectors of $\mathbf{A}$, but they can be computed using the LAPACK routine `DTREVC`. We have done this in our tests with `SRRIT`.

For `SRRIT` and `EB12`, we check the orthogonality of the computed Schur basis and quality of the Schur projection by computing

$$\|\mathbf{X}_r^T\mathbf{X}_r - \mathbf{I}_r\|_\infty \quad \text{and} \quad \|\mathbf{X}_r^T\mathbf{A}\mathbf{X}_r - \mathbf{T}_r\|_\infty, \tag{5.10}$$

respectively.

The checks (5.8)–(5.10) 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 they have 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).

Even if we use the default parameters, there remain parameters which must be chosen by the user. In particular, each of the codes in our study requires the user to choose the number r of required eigenvalues, the subspace dimension m, and the convergence tolerance ϵ. Requiring the user to choose these parameters may appear reasonable since 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 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 is strongly dependent 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). When testing the software on a particular problem, we use the same value of r and m for each code. This allows use to compare the relative performances of the codes.

Code	Required input
`LOPSI`	Matrix $\mathbf{A}$ in standard sparse format Maximum number of products $\mathbf{AX}_m$ Maximum number l products $\mathbf{AX}_m$ between the oblique projections
`SRRIT`	Routine for performing $\mathbf{AX}_m$ Maximum number of iterations
`EB12`	Which eigenvalues

Table 5.4: Input from the user

In addition to the parameters m, r, and ϵ, all the codes require the user to supply values for at least one other input parameter. These parameters are listed in Table 5.4. For `EB12` it is only necessary to specify whether the right-most eigenvalues, the left-most eigenvalues, or eigenvalues of largest modulus are required. Finite termination of the computation is ensured by specifying the maximum number of iterations and/or the maximum number of matrix products. `EB12` avoids requiring the user to set these parameters by using default values, which are held in fully documented `COMMON` block and which the user is able to reset.

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

5.4 Results

The results of our numerical experiments are lengthy so we only summarise our findings here. The complete set of results for computing a single eigenpair are given in Appendix A and for several eigenpairs the results are in Appendix B. For each of the test examples, results are only given for the codes which converged with the requested accuracy to the correct eigenvalues. No results are given for the matrix BWM2000 since, for a range of values of m, the codes all failed to achieve the required accuracy within the limit of $4000m$ matrix-vector products. The codes also failed to converge for the CDDE problem as n and ρ were increased. In particular, for $n = 10000$ and $\rho = 40$, convergence was not achieved. The code `LOPSI` could only be run on the test matrices which are held explicitly (which includes all the Harwell-Boeing matrices but not TOLOSA, RWK5151, and CDDE). For the test matrices for which the eigenvalues of largest modulus are also the right-most (or left-most) eigenvalues, results are given for `EB12` with and without Chebychev acceleration. If Chebychev acceleration is used, this indicated by a ‡ symbol.

Our general findings from the numerical experiments were the following.

- `LOPSI` can be the fastest subspace iteration code for computing a single eigenvalue but if $r > 1$, it can return spurious eigenvalues, without warning. We believe there is probably a bug in the code but we have not yet located the source of the problem.

- For some examples, `EB12` and `LOPSI` require a large number of iterations to achieve convergence (see, for instance, `EB12` applied to TOLOSA and `LOPSI` applied to NNC666).

- `SRRIT` can be much slower than `EB12` and `LOPSI`. The reason for this appears to be that it generally performs many more reorthogonalisations than the other codes. The extra cost incurred by the reorthogonalisations is illustrated by our results for the problems IMPCOLD and NNC1374 with $r = 5$. Both `EB12` and `SRRIT` perform a similar number of iterations and matrix-vector products but the CPU time for `SRRIT` is much greater than for `EB12`.

- 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. The stopping criteria used by `EB12` which attempts to detect stagnating convergence means that `EB12` does not always display this property. However, `SRRIT` may return eigenvalues which have much greater accuracy than was requested by the user. For example, for the problem TOLOSA with $n = 1000$ and $m = 16$, the amount of work required for convergence using a convergence tolerance of 10^{-4} is the same as for a convergence tolerance of $\sqrt{u}$; both tolerances yield computed eigenvalues with a maximum forward error of order 10^{-9}.

- In general, the subspace iteration codes were found to be good at detecting multiple eigenvalues and at not missing any of the sought-after eigenvalues. In particular, `SRRIT` has a useful property of being able to recognise clusters of eigenvalues. For problems such as CK400 and CK656, which have multiple and clustered eigenvalues, `SRRIT` was the most efficient code. The ability to detect clusters can lead to `SRRIT` returning more eigenvalues than was requested. Our experience suggests 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. An example for which Chebychev acceleration led to substantial improvements was CDDE. For this problem, as n and ρ where increased, convergence was not achieved without Chebychev acceleration. However, it cannot be anticipated whether Chebychev acceleration will be beneficial unless some knowledge of the distribution of the eigenvalues is already known.

- All the codes were either slow to converge or failed to converge for matrices with a large departure for non-normality. The eigenvalues of the TOLOSA matrix 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 so the subspace iteration codes may be used. The matrix is non-normal and its departure from normality increases with the order n of the matrix. We found that the time taken and the number of matrix-vector products performed to achieve convergence increases with n so that, for sufficiently large n, the subspace iteration codes become too expensive to be of practical use. For this problem and the CDDE problem with n large an alternative approach, such as Arnoldi's method or the Lanczos method, may be needed.

6 Concluding remarks and comments

In this final section, we briefly highlight some of the main weaknesses of the current subspace iteration software. We also look at features found in the software that should be part of any revisions to existing software or should be incorporated in new attempts at high quality sparse eigenvalue software.

- The software could be made more user-friendly by improving the error handling and printing options.
- The grouping of clustered eigenvalues would appear to be an important feature for problems with equal or nearly equal eigenvalues. This is attempted by `SRRIT` but further investigation is needed.
- A well-engineered strategy for detecting stagnating convergence may prevent unnecessary work being performed. Although not always successful, `EB12` attempts to do this; further investigation is needed. More generally, as examined in Section 4.5, further research and testing needs to be undertaken to improve the ways in which the software decides to terminate the computation.
- All the codes we looked at were written in `FORTRAN 77`. For such codes, we recommend the use of reverse communication for carrying out matrix products $\mathbf{AX}$. 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 acceleration methods are needed. `EB12` offers Chebychev acceleration but no other acceleration methods are currently incorporated within the software. `EB12` has no mechanism for automatically selecting or deselecting Chebychev acceleration as the computation proceeds. Reverse communication allows `EB12` to be used with shift-and-invert but no shift selection strategy is offered.
- The ability to deflate a converged invariant subspace can substantially reduce the amount of work performed when more than one eigenpair is wanted. In common with the existing codes, deflation should be incorporated within any new software.

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

- The automated selection of software parameters needs further work. 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 dynamic allocation of storage is allowed as well as optional arguments.

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 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. can be obtained 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`.

- `EB12` is included in 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.

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.

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.

Å. Björck. Solving linear least squares problems by Gram–Schmidt orthogonalization. *BIT*, **7**, 1–21, 1967.

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.

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.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 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.

T. J. Garratt, G. Moore, and A. Spence. Two methods for the numerical detection of Hopf bifurcations. *In* R. Seydel, F. W. Schneider and H. Troger, eds, 'Bifurcation and Chaos: Analysis, Algorithms and Applications', pp. 119–123. Birkhauser, 1991.

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.

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.

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 and J. A. Scott. An evaluation of Arnoldi based software for sparse nonsymmetric eigenproblems. Technical Report RAL-TR-96-023, Rutherford Appleton Laboratory, Chilton, England, 1996*a*.

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*b*. Submitted to ACM Transactions on Mathematical Software.

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.

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

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.

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.

A Appendix: Tables of results for a single eigenpair

A superscript of the form $(k\lambda)$ indicates the code returned k eigenpairs ($k > 1$, or 2 if the dominant eigenpair is complex).

PORES2, $r = 1, m = 8$

Code	Matrix-vector products	Iterations	Time (secs)
LOPSI	121	4	0.18
SRRIT	168	$4^{(3\lambda)}$	0.38
EB12	111	2	0.20

‡ indicates Chebychev acceleration used.

PORES3, $r = 1, m = 8$

Code	Matrix-vector products	Iterations	Time (secs)
LOPSI	769	11	0.26
SRRIT	128	$7^{(2\lambda)}$	0.81
EB12	1735	7	0.56
EB12‡	815	4	0.35

‡ indicates Chebychev acceleration used.

GRE1107, $r = 1, m = 8$

Code	Matrix-vector products	Iterations	Time (secs)
LOPSI	1761	24	1.41
SRRIT	2032	$8^{(2\lambda)}$	2.58
EB12	2159	12	1.59
EB12‡	3367	14	3.13

‡ indicates Chebychev acceleration used.

HOR131, $r = 1, m = 8$

Code	Matrix-vector products	Iterations	Time (secs)
LOPSI	177	5	0.08
SRRIT	168	4	0.11
EB12	127	3	0.08
EB12‡	119	2	0.08

‡ indicates Chebychev acceleration used.

IMPCOLC, $r = 1, m = 8$

Code	Matrix-vector products	Iterations	Time (secs)
LOPSI	2753	36	0.16
SRRIT	2248	$8^{(8\lambda)}$	0.40
EB12	183	3	0.02

IMPCOLD, $r = 1, m = 8$

Code	Matrix-vector products	Iterations	Time (secs)
LOPSI	†	†	†
SRRIT	3976	9	1.61
EB12	3295	10	0.53

† indicates sought-after eigenvalue was missed.

NNC666, $r = 1, m = 8$

Code	Matrix-vector products	Iterations	Time (secs)
LOPSI	1849	40	0.94
SRRIT	1288	$7^{(2\lambda)}$	1.23
EB12	1095	5	0.40

‡ indicates Chebychev acceleration used.

NNC1374, $r = 1, m = 8$

Code	Matrix-vector products	Iterations	Time (secs)
LOPSI	3457	58	3.72
SRRIT	5128	$9^{(2\lambda)}$	8.24
EB12	3079	8	2.61
EB12‡	583	5	0.76

‡ indicates Chebychev acceleration used.

WEST0156, $r = 1, m = 8$

Code	Matrix-vector products	Iterations	Time (secs)
LOPSI	177	9	0.02
SRRIT	200	$5^{(7\lambda)}$	0.06
EB12	119	2	0.03

WEST0167, $r = 1, m = 8$

Code	Matrix-vector products	Iterations	Time (secs)
LOPSI	137	4	0.20
SRRIT	1288	$7^{(6\lambda)}$	0.35
EB12	119	2	0.02

WEST2021, $r = 1, m = 8$

Code	Matrix-vector products	Iterations	Time (secs)
LOPSI	49	2	0.10
SRRIT	72	3	0.34
EB12	15	1	0.08

B Appendix: Tables of results for several eigenpairs

A superscript of the form $(k\lambda)$ indicates the code returned k eigenpairs ($k > r$ where r is the requested number of eigenpairs.

PORES2, $r = 4, m = 20$

Code	Matrix-vector products	Iterations	Time (secs)
LOPSI	$\diamond$	$\diamond$	$\diamond$
SRRIT	2704	7	7.88
EB12	695	5	2.31

$\diamond$ indicates spurious eigenvalues were returned.
‡ indicates Chebychev acceleration used.

PORES3, $r = 5, m = 20$

Code	Matrix-vector products	Iterations	Time (secs)
LOPSI	841	6	0.47
SRRIT	1364	$6^{(13\lambda)}$	1.34
EB12	869	6	0.67
EB12‡	717	4	0.51

‡ indicates Chebychev acceleration used.

GRE1107, $r = 5, m = 20$

Code	Matrix-vector products	Iterations	Time (secs)
LOPSI	†	†	†
SRRIT	2198	10	4.48
EB12	4774	14	4.35
EB12‡	5207	18	6.27

† indicates eigenvalues were missed.
‡ indicates Chebychev acceleration used.

HOR131, $r = 5, m = 20$

Code	Matrix-vector products	Iterations	Time (secs)
LOPSI	281	4	0.21
SRRIT	279	5	0.41
EB12	483	4	0.38
EB12‡	719	5	0.52

‡ indicates Chebychev acceleration used.

IMPCOLC, $r = 5, m = 20$

Code	Matrix-vector products	Iterations	Time (secs)
LOPSI	581	7	0.08
SRRIT	620	5	0.20
EB12	279	2	0.06

IMPCOLD, $r = 5, m = 20$

Code	Matrix-vector products	Iterations	Time (secs)
LOPSI	◇	◇	◇
SRRIT	8500	$10^{(8\lambda)}$	6.36
EB12	8233	11	1.69

◇ indicates spurious eigenvalues were returned

NNC666, $r = 5, m = 20$

Code	Matrix-vector products	Iterations	Time (secs)
LOPSI	17429	152	12.67
SRRIT	1620	$6^{(6\lambda)}$	2.23
EB12	3347	8	1.63

NNC1374, $r = 5, m = 20$

Code	Matrix-vector products	Iterations	Time (secs)
LOPSI	5211	43	7.87
SRRIT	5600	$8^{(6\lambda)}$	12.48
EB12	5755	10	5.45

WEST0156, $r = 5, m = 20$

Code	Matrix-vector products	Iterations	Time (secs)
LOPSI	$\diamond$	$\diamond$	$\diamond$
SRRIT	340	$4^{(7\lambda)}$	0.20
EB12	39	$1^{(6\lambda)}$	0.02

$\diamond$ indicates spurious eigenvalues were returned

WEST0167, $r = 5, m = 20$

Code	Matrix-vector products	Iterations	Time (secs)
LOPSI	421	6	0.07
SRRIT	420	$4^{(8\lambda)}$	0.23
EB12	383	4	0.05

WEST2021, $r = 5, m = 20$

Code	Matrix-vector products	Iterations	Time (secs)
LOPSI	$\diamond$	$\diamond$	$\diamond$
SRRIT	3070	$7^{(9\lambda)}$	19.19
EB12	724	5	6.44

$\diamond$ indicates spurious eigenvalues were returned.

CK400, $r = 8, m = 20$

Code	Matrix-vector products	Iterations	Time (secs)
LOPSI	◇	◇	◇
SRRIT	994	7	0.83
EB12	1401	8	0.90
EB12‡	943	6	0.83

◇ indicates spurious eigenvalues were returned.
‡ indicates Chebychev acceleration used.

CK656, $r = 8, m = 20$

Code	Matrix-vector products	Iterations	Time (secs)
LOPSI	◇	◇	◇
SRRIT	994	21	1.29
EB12	1508	21	2.15
EB12‡	1065	7	1.61

◇ indicates spurious eigenvalues were returned.
‡ indicates Chebychev acceleration used.

RWK5151, $r = 2, m = 10$

Code	Matrix-vector products	Iterations	Time (secs)
SRRIT	15140	$11^{(4\lambda)}$	125.10
EB12	14609	18	56.19

TOLOSA, $r = 2$

n	m	Code	Matrix-vector products	Iterations	Time (secs)
500	8	SRRIT	20488	$11^{(6\lambda)}$	15.8
		EB12	6367	73	5.9
500	16	SRRIT	64016	$11^{(14\lambda)}$	41.9
		EB12	11327	65	15.3
1000	8	SRRIT	20488	$11^{(4\lambda)}$	33.2
		EB12	17719	202	34.1
1000	16	SRRIT	64016	$12^{(14\lambda)}$	141.3
		EB12	35087	200	119.8
2000	8	SRRIT	*	*	*
		EB12	26255	299	117.0
2000	16	SRRIT	*	*	*
		EB12	9919	57	72.0
3000	8	SRRIT	*	*	*
		EB12	*	*	*
3000	16	SRRIT	*	*	*
		EB12	14847	85	159.4

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

CDDE, $r = 6$

ρ	n	m	Code	Matrix-vector products	Iterations	Time (secs)
0	2500	18	SRRIT	29583	11	83.2
			EB12	29205	25	29.0
			EB12†	5312	11	9.6
0	2500	36	SRRIT	27465	10	106.4
			EB12	26527	18	37.5
			EB12†	14450	12	27.5
10	2500	18	SRRIT	46098	11	126.9
			EB12	48296	33	45.8
			EB12†	5971	22	13.5
10	2500	36	SRRIT	88356	$11^{(19\lambda)}$	331.6
			EB12	45192	25	55.5
			EB12†	9668	19	32.8
0	10000	18	SRRIT	*	*	*
			EB12	*	*	*
			EB12†	8631	13	62.2
0	10000	36	SRRIT	92196	$11^{(11\lambda)}$	2326
			EB12	93509	35	426.4
			EB12	20099	14	153.0
15	10000	18	SRRIT	*	*	*
			EB12	*	*	*
			EB12†	41857	54	283.7
15	10000	36	SRRIT	*	*	*
			EB12	*	*	*
			EB12†	34268	36	291.7

* denotes that code did not converge within $4000m$ matrix-vector products.
‡ indicates Chebychev acceleration used.